Cognos(R)
Application Development Tools
PowerHouse(R) 4GL

VERSION 8.4E

POWERHOUSE RULES

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 Global Customer Services Web site (http://support.cognos.com).

Copyright

Table of Contents

About this Book *13*
- Overview *13*
- Conventions in this Book *13*
- Getting Help *13*
- Cognos PowerHouse 4GL Documentation Set *14*
- Cognos PowerHouse Web Documentation Set *15*
- Cognos Axiant 4GL Documentation Set *16*

Chapter 1: Running PowerHouse *17*
- Before Running PowerHouse *17*
- Getting Help *17*
- Setting Up the PowerHouse Environment *18*
  - MPE/iX *18*
  - OpenVMS *18*
  - UNIX *18*
  - Windows *19*
- Running QDESIGN *20*
- Running QUICK *21*
- Running QUIZ *22*
- Running QTP *24*
- Running PDL *27*
- Running PHDPDL (OpenVMS) *28*
- Running QSHOW *29*
- Running QUTIL *30*
- PowerHouse Commands (OpenVMS) *31*
- @SETPOWERHOUSE (OpenVMS) *32*
- POWERHOUSE (OpenVMS) *33*
- SETDICTIONARY (OpenVMS) *36*
- SHOWDICTIONARY (OpenVMS) *38*
- SHOWPOWERHOUSE (OpenVMS) *39*
- SHOWQUOTA (OpenVMS) *40*
- Locating Files *41*
  - Locating the Data Dictionary *41*
  - How the BUILD and SAVE Statements Locate Files *41*
  - How the EXECUTE and USE Statements and auto Program Parameter Locate Files *42*
  - Locating Start Screens or QKGO files in QUICK *43*
  - How the GO Statement Locates Files *44*
  - Locating Subfiles *44*
  - Locating ODS5 File Names (OpenVMS) *46*
- Designated Files *48*
- PDC Shared Dictionary (OpenVMS) *53*
  - Introduction *53*
  - Requesting Dictionary Installations *53*
  - Shared Memory Configuration *53*
- PHD Shared Dictionary (OpenVMS) *55*
- PDL Shared Dictionary (UNIX) *56*
  - Installing Your Dictionary *56*
  - Shared Memory Management *58*
- Mailbox Support in PowerHouse (OpenVMS) *59*

Creating a Temporary or Permanent Mailbox 59
Temporary Mailbox Application 59
Permanent Mailbox Application 60
Using Mailboxes to Pass Source Statements 62
Mailboxes and System Crashes 62
sitehook (OpenVMS) 63
Large File Support (UNIX, Windows) 64
DISAM Data Storage (Windows) 65

Chapter 2: Program Parameters 67

About Program Parameters 67
Summary of Program Parameters 67
auto 75
autodetach|noautodetach 76
blockmode (MPE/iX) 77
broadcast (OpenVMS) 78
bulkfetch 79
cc 80
charmode 81
checksum710 (OpenVMS) 82
close_detach 84
columnowner 85
commitpoints 87
compress_buffers 88
confirmer 89
consolekeys|noconsolekeys (Windows) 90
createall 91
createbase (MPE/iX) 92
createfile 93
cursorowner 94
dbaudit 96
dbdetach|nodbdetach 98
dbwait|nodbwait 99
dcl|nodcl (OpenVMS) 100
debug (QDESIGN) 101
debug (QUICK) 102
deleteall 103
deletebase (MPE/iX) 104
deletefile 105
designer_noretain 106
detail|nodetail 107
dictionary|dict 108
dicttype|dt (OpenVMS) 109
direct_file_base_zero (OpenVMS) 110
disable_nulls 111
dont_store_module 112
downshift|upshift|noshift 113
entryrecall 114
errlist 115
fastread (OpenVMS) 116
fdl|nofdl (OpenVMS) 117
initnulls|noinitnulls 118
intsize6|nointsize6 (OpenVMS) 119
jcwbase (MPE/iX) 120
lineread (MPE/iX) 121
list|nolist 122
lockword (MPE/iX) 123

moduleext (MPE/iX) *124*
moduleloc (MPE/iX) *125*
nls (no line split) (MPE/iX, UNIX) *126*
noblobs *127*
nobreakset (MPE/iX) *128*
nonportable *129*
nontermcompat (Windows) *130*
noowner *131*
noprefix_ownername *132*
nosetwarnstatus (OpenVMS) *133*
nouicbrackets (OpenVMS) *134*
nxl (no extra line) *135*
obsolete *136*
omnidex|noomnidex (MPE/iX) *137*
osaccess|noosaccess *138*
owner *139*
parmfile (OpenVMS, UNIX, Windows) *140*
parmprompt *141*
patch *142*
pollspeed (MPE/iX) *143*
pre_chooseall *144*
procloc *145*
prompt *146*
qktrace *147*
quotedproccall *149*
read (MPE/iX) *150*
resetbindvar|noresetbindvar *151*
resource *152*
restore *153*
retainmark|noretainmark *154*
reuse_screen_buffers|noreuse_screen_buffers *155*
search *156*
secured *157*
setjobshow|nosetjobshow (Windows) *158*
statistics|nostatistics *159*
subdictionary|subdict *160*
subformat *161*
term *162*
termpoll|notermpoll (MPE/iX, OpenVMS) *165*
timezone|notimezone (MPE/iX) *166*
tpi|notpi (MPE/iX, HP-UX, Windows) *167*
trusted|notrusted (OpenVMS) *168*
update *170*
version *171*
vmsdate (OpenVMS) *172*

Chapter 3: Resource File Statements *173*

About Resource File Statements *173*
Summary of Resource File Statements *173*
ALLBASE MODULE EXTENSION (MPE/iX) *180*
AUTODETACH *181*
BROADCAST (OpenVMS) *182*
BULKFETCH n *183*
CC *184*
CHECKSUM710 *185*
CLOSE DETACH *187*
COLUMNOWNER *188*

COMMITPOINTS OBSOLETE *190*
COMPRESS BUFFERS *191*
CONSOLE KEYS (Windows) *192*
DATABASE *193*
DBAUDIT *195*
DBDETACH *197*
DBWAIT *198*
DEBUG *199*
DEFAULT CURSOR OWNER *200*
DESIGNER NORETAIN *201*
DICTIONARY *202*
DIRECTORY (UNIX, Windows) *204*
DISABLE NULLS *205*
ENTRY RECALL *206*
EXIT *207*
HPSLAVE EXTRA LINE *208*
HPSLAVE SPLIT LINES (MPE/iX, UNIX) *209*
INITIALIZE NULLS *210*
INTEGER SIZE 6 (OpenVMS) *211*
JCWBASE (MPE/iX) *212*
LIST *213*
LOCATION MODULE (MPE/iX) *214*
LOCATION PROCESS *215*
LOCKWORD (MPE/iX) *216*
NOBLOBS *217*
NONPORTABLE *218*
NOOWNER *219*
NOSET WARN STATUS (OpenVMS) *220*
OBSOLETE *221*
OMNIDEX (MPE/iX) *222*
OSACCESS *223*
OWNER *224*
PREFIX ORACLE OPEN NAME *225*
PROMPT *226*
QUIT *227*
RESET BIND VARIABLES *228*
RESTORE LINES *229*
RETAIN MARK *230*
REUSE SCREEN BUFFERS *231*
RMS FAST READ (OpenVMS) *232*
RMS FILE BASE (OpenVMS) *233*
SET *234*
SETJOBSHOW (Windows) *235*
SHIFT *236*
STATISTICS *237*
STORE MODULES *238*
SUBDICTIONARY *239*
SUBFORMAT n *240*
TERMINAL *241*
TERMINAL BLOCKMODE (MPE/iX) *244*
TERMINAL CHARACTERMODE *245*
TERMINAL CONFIRMER *246*
TERMINAL POLLING SPEED (MPE/iX) *247*
TERMINAL READ (MPE/iX) *248*
TERMPOLL (MPE/iX, OpenVMS) *249*
TIC RESOURCE FILE (UNIX, Windows) *250*
TIME ZONE (MPE/iX) *251*

TPI (MPE/iX, HP-UX, Windows) *252*
TRUNCATE PARM VALUES *253*
TRUSTED (OpenVMS) *254*
UIC BRACKETS (OpenVMS) *256*
UPDATE ORDER *257*
USE *258*
VMSDATE *259*

Chapter 4: Messages in PowerHouse *261*
PowerHouse 4GL Messages *261*
Service Layer Messages *265*
How the Service Layer Locates Message Files *265*
Format of a Default Message File *266*
Service Message Compiler *268*
Designer Messages *270*
Text Order Numbering *271*

Chapter 5: PowerHouse Language Rules *273*
Syntax Symbols in PowerHouse *273*
Uppercase and Lowercase *273*
Square Brackets *273*
Braces *274*
Ellipsis *274*
Or-Bars *274*
Stacked Syntax *274*
Indented Syntax *274*
General Terms in PowerHouse *275*
Entering Statements *281*
Abbreviating Keywords *281*
Avoiding Conflicts Between Keywords and Record or Item Names *281*
What Happens When You Enter Statements *281*
Entering Comments *281*
Entering Conditional Compile Statements *282*
Operating System Commands *283*
Arrays in PowerHouse *284*
Using Arrays in QDESIGN *284*
Subscripting in QUIZ and QTP *285*
Using Arrays in QUIZ *285*
Using Arrays in QTP *286*
Conditions in PowerHouse *289*
Logical Function *289*
Logical Expression *289*
Predefined Conditions in QDESIGN *290*
Predefined Conditions in QTP *293*
Predefined Conditions in QUIZ *295*
Simple Conditions *295*
Compound Conditions *296*
Modifying Simple and Compound Conditions *296*
Conditional Command List *296*
Conditions and NULL Values *297*
Conditions in SQL *298*
sql-expression operator {sql-expression|subquery} *298*
sql-expression operator {ALL|SOME|ANY} subquery *298*
columnspec [NOT] LIKE 'sql-pattern' [ESCAPE 'character'] *298*
columnspec IS [NOT] NULL *299*
sql-expression [NOT] IN (value, value[...])|subquery *299*
[NOT] EXISTS subquery *299*
Expressions in PowerHouse *300*

String Expressions *300*
Numeric Expressions *300*
Date Expressions *301*
Conditional Expressions *301*
Case Processing *302*
Expressions in SQL *303*
String Expressions *303*
Numeric-Expressions *303*
Date Expressions *303*
SQL Case Processing *303*
Expressions within Program Variables *304*
SQL Summary Operations *304*
Items and Datatypes in PowerHouse *306*
Defined Items *306*
Global Temporary Items (QTP) *306*
Predefined Items (QDESIGN) *306*
Record Items *307*
Temporary Items *308*
How QDESIGN Searches for Items *308*
How QTP Searches for Items *308*
Item Types *309*
Item Datatypes *309*
Item Sizes *310*
Non-Relational PowerHouse Datatypes *311*
Relational PowerHouse Datatypes (Part 1) *313*
Relational PowerHouse Datatypes (Part 2) *315*
Relational Datatypes Specifics *316*
BLOB Datatype *317*
CHARACTER Datatype *317*
DATE Datatype *317*
DATETIME Datatype *318*
FLOAT Datatype *318*
FREEFORM Datatype *320*
INTEGER Datatype *320*
INTERVAL Datatype *321*
JDATE Datatype *321*
NUMERIC Datatype *321*
PACKED Datatype *321*
PHDATE Datatype *322*
VARCHAR Datatype *322*
VMSDATE Datatype (OpenVMS) *322*
ZDATE Datatype *323*
ZONED Datatype *323*
User-Defined Datatypes *324*
ORACLE Synonyms in PowerHouse *325*
Limitations to PowerHouse Statements *325*
Attributes of Numeric Elements *326*
The Input Conversion Process *326*
Default Assumptions Governing Input *327*
The Output Conversion Process *327*
Default Assumptions for Display Attributes *328*
Specifying Decimal Currencies *328*
Displaying Negative Values *329*
Multiplication and Percentage Calculations *329*
Decimal Alignment and Scaling *330*
Conditions and Scaled Values *331*
VALUES Options and Scaled Values *331*

Calculations and Scaled Values *331*
Floating Point Calculations *332*
Examples of Calculations *333*
Notes on Scaling Efficiency *333*
QUICK Screen Commands *334*
Using Screen Commands in Command Lists *334*
Action Commands: *335*
Data Commands: *340*
Action Bar Commands *342*
Field Marking Commands *342*
Line Edit Commands *342*
Menu/List/Selection Box Commands *343*
Popup Commands *343*
System Commands *343*
Text Edit Commands *343*
Blob Support in PowerHouse *345*
Using Blobs in PowerHouse Expressions *345*
Using Blobs *345*
Restrictions on Blobs *346*
Null Value Support in PowerHouse *347*
Enabling Null Value Item Initialization *347*
Automatic Item Initialization *347*
Entering and Displaying Null Values *347*
Assigning Null Values *348*
Testing for Null Values *348*
Operating on Null Values in PowerHouse *348*
Selective Record Retrieval Based on Null Values *349*
Controlling Null Value Entry in QDESIGN *349*
Pattern Matching in PowerHouse *351*
Types of Characters Used in Pattern Matching *351*
Types of Patterns *353*
Formal Pattern Matching Syntax *354*
Example Patterns *355*
Pattern Matching in SQL *356*
Using the SOUNDEX Option *357*
SOUNDEX Option Rules *357*

Chapter 6: Functions in PowerHouse *359*

About Functions in PowerHouse *359*
Summary of PowerHouse Functions *359*
ABSOLUTE *367*
ADDCENTURY *368*
ASCII *369*
ATTRIBUTE *370*
AUDITSTATUS *371*
Bit Extract *372*
BITEXTRACT *373*
CEILING *374*
CENTER|CENTRE *375*
CENTURY *376*
CHARACTERS *378*
CHARACTER_LENGTH|CHAR_LENGTH *379*
CHECKSUM *380*
COMMANDCODE *383*
COMMANDMESSAGE *384*
COMMANDSEVERITY (OpenVMS) *385*
COMMANDSTATUS (OpenVMS) *386*

CONTENTS *387*
DATE *388*
DATEEXTRACT *390*
DAYS *391*
DECIMALTIME *392*
DECRYPT *393*
DELETESYSTEMVAL (MPE/iX, UNIX, and Windows) *394*
DELETESYSTEMVAL (OpenVMS) *397*
DOWNSHIFT *401*
ENCRYPT *402*
EXTRACT *404*
FIRST *405*
FLOOR *406*
FORMATNUMBER *407*
GETSYSTEMVAL (MPE/iX, UNIX, and Windows) *412*
GETSYSTEMVAL (OpenVMS) *413*
INDEX *414*
HEXDECODE Function *415*
HEXENCODE Function *416*
INTERVAL *417*
JCW (MPE/iX) *418*
LASTDAY *419*
LEFT JUSTIFY | LJ *420*
LINKVALUE *421*
LOGONID *422*
LOWER *423*
MATCHPATTERN *424*
MATCHUSER *425*
MISSING *426*
MOD *427*
NCONVERT *428*
NULL *429*
OCCURRENCE *430*
OCTET_LENGTH *432*
OLDVALUE *433*
PACK *434*
PORTID *435*
POSITION *436*
PROCESSLOCATION *437*
RANDOM *438*
RECORDLOCATION *439*
REMOVECENTURY *440*
REVERSE *441*
RIGHT JUSTIFY | RJ *442*
ROUND *443*
SCREENLEVEL *446*
SETSYSTEMVAL (MPE/iX, UNIX, and Windows) *447*
SETSYSTEMVAL (OpenVMS) *449*
SHIFTLEVEL *451*
SIGNONACCOUNT (MPE/iX) *452*
SIGNONGROUP (MPE/iX) *453*
SIGNONUSER *454*
SIZE *455*
SOUNDEX *456*
SPREAD *458*
SQLCODE *459*
SQLMESSAGE *461*

SUBSTITUTE *462*
SUBSTRING *464*
Substring Extract *465*
SUM *466*
SYSDATE *468*
SYSDATETIME *469*
SYSNAME *470*
SYSPAGE *471*
SYSTIME *472*
TERMTYPE *473*
TRUNCATE *474*
UIC (OpenVMS, UNIX) *475*
UPPER *476*
UPSHIFT *477*
VALIDPATTERN *478*
VMSTIMESTAMP (OpenVMS) *479*
WEBLOGONID *480*
ZEROFILL *481*

Glossary *483*

Index *497*

About this Book

Overview

This book is intended for experienced PowerHouse users who require a concise summary of PowerHouse rules and statements.

Chapter 1, "Running PowerHouse", tells you how to run every component of PowerHouse, including QDESIGN, QUICK, QUIZ, QTP, and PDL and its utilities, QSHOW and QUTIL. It also tells you how to locate PowerHouse Data Dictionary and Source Statement files, and describes designated files that are reserved for use as PowerHouse default files.

Chapter 2, "Program Parameters", describes the program parameters that control attributes such as determining which dictionary PowerHouse runs.

Chapter 3, "Resource File Statements", describes the statements you can use to specify program parameters and other system characteristics for use with PowerHouse applications.

Chapter 4, "Messages in PowerHouse", provides information about PowerHouse 4GL messages, service layer messages, and designer messages for PowerHouse applications.

Chapter 5, "PowerHouse Language Rules", contains detailed discussions of PowerHouse syntax and its presentation, general terms that appear in syntax, and related topics.

Chapter 6, "Functions in PowerHouse", describes PowerHouse functions, providing syntax descriptions, function discussions, and examples, where applicable.

The book also contains a "Glossary" of PowerHouse terms.

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.

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
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
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
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: Running PowerHouse

Overview

This chapter describes

- how to run the PowerHouse components: QDESIGN, QUICK, QUIZ, QTP, PDL, PHDPDL, QSHOW, and QUTIL
- how to locate PowerHouse data dictionary and other files
- the designated files that are reserved for use as PowerHouse default files
- how to installed shared dictionaries

Before Running PowerHouse

The appropriate user-defined commands (UDCs) (**MPE/iX**), logicals and symbols (**OpenVMS**), or environment variables (**MPE/iX, UNIX, Windows**) should be set before you run PowerHouse. For further information, see the *Cognos PowerHouse 4GL and PowerHouse Web Getting Started* book for your platform.

Getting Help

As you're entering statements in PowerHouse, you can see a brief list of the allowed syntax of a statement by entering a question mark (**?**). For example, entering

```
> ACCESS ?
```

produces a list of the keywords that you can enter after the ACCESS statement.

Setting Up the PowerHouse Environment

MPE/iX

Access to PowerHouse 4GL is achieved through the PowerHouse 4GL UDC catalog, PH<version>UDC.PH<*version*>.COGNOS. When client/server applications are built and run through Axiant 4GL, some of the same setup is done using a command file which is restored from the tape as SETPOW.PH<*version*>.COGNOS.

If you have certain HP or third party software installed on your computer, you will have to modify these two files.

For example, to enable PowerHouse 4GL 8.39C, sign on to the account in which PowerHouse 4GL is to be used and enter

```
SETCATALOG PH839UDC.PH839.COGNOS
```

OpenVMS

Once PowerHouse has been installed according to the instructions in the *Getting Started* book, you usually set it up by entering the command @SETPOWERHOUSE at the operating system prompt ($), like this:

```
$ @SETPOWERHOUSE
```

Your computer may be set up so that you don't have to use this command, or so that you have to use some other procedure instead. See your system manager for more information.

There are options for the @SETPOWERHOUSE command that let you specify which version of PowerHouse you want to use and in what language you want to use it. For more information about the @SETPOWERHOUSE command, see (p. 32).

If, when you run PowerHouse, it prompts for your terminal type, or if the screens are displayed incorrectly on your terminal, use the DCL command SET TERMINAL, which allows OpenVMS to determine the correct terminal type. Entering the commands

```
$ SET TERMINAL/UNKNOWN
$ SET TERMINAL/INQUIRE
```

sets up VT series terminals and most DEC terminal emulators. If you are prompted for a terminal type again when you are working in PowerHouse, enter a question mark (?) to view a listing of supported terminal types.

You can also use the SET TERMINAL command to establish your terminal's keypad for numeric data-entry or function key use. The command

```
$ SET TERMINAL/NUMERIC_KEYPAD
```

sets up the keypad as a numeric keypad. The DCL command

```
$ SET TERMINAL/APPLICATION_KEYPAD
```

sets up the keypad for function key use.

UNIX

Once PowerHouse has been installed according to the instructions in the *Getting Started* book, you usually set it up by entering the command

```
setenv PH <powerhouse version>
```

Once the Powerhouse version is set you must resource your environment to pick up the correct version.

Your system may be set up so that you don't have to use this command, or so that you have to use some other procedure instead. See your system manager for more information.

To set a dictionary on the UNIX platform, use the following command at the operating system prompt:

```
setdict <dictionary compiled name>
```

To show which dictionary you have set, use the following command:

```
showdict
```

By default, PDL, QDESIGN, QSHOW, QTP, QUIZ and QUTIL create a temporary directory in the current working directory named phnnnn.tmp where nnnn is the process id. This temporary directory is used to store temporary files, such as the source statement save file. If one component is started from within another, the original component's temporary directory is used. A temporary directory is deleted when the component that created it exits.

If the PHTEMP environment variable is set, temporary files are created in the PHTEMP location and the temporary directories are not created.

If you require a unique name for the PHTEMP location, use $$ in the following technique:

- Bourne shell or Korn shell users:

```
PHTEMP=phtemploc_$$.tmp
export PHTEMP
```

- C shell users:

```
setenv PHTEMP phtemploc_$$.tmp
```

Windows

Once PowerHouse has been installed according to the instructions in the *Getting Started* book, you can start a PowerHouse component from

- a shortcut in the Start menu
- the Run dialog in the Start menu
- a Command Prompt window

By default, the shortcuts in the Start menu have the install location as the 'start in' or working location. Typically, you would start a Command Prompt session, navigate to the working directory, and type the component name to launch it.

By default, PDL, QDESIGN, QSHOW, QTP, QUIZ, and QUTIL create a temporary directory in the current working directory named phnnnn.tmp where nnnn is the process id. This temporary directory is used to store temporary files such as the source statement save file. If one component is started from within another, the second component creates its own temporary directory. A temporary directory is deleted when the component that created it exits.

If the PHTEMP environment variable is set, the temporary directories are created in the PHTEMP location.

Running QDESIGN

To run QDESIGN from the operating system prompt, enter:

```
qdesign
```

To run QDESIGN from within another PowerHouse component, enter:

MPE/iX:	: qdesign
OpenVMS:	$ qdesign or ! qdesign
UNIX, Windows:	! qdesign

To run QDESIGN on Windows:

1. Select the QDESIGN option from the PowerHouse <*version*> option of the Start Menu.
2. From the Programs option of the Start Menu, select the Command Prompt option. A Command Prompt window appears. From the Command Prompt window, enter:
   ```
   qdesign
   ```

Entering QDESIGN Program Parameters

You can include program parameters when running QDESIGN from the operating system prompt or from within another component.

All valid program parameters can be accessed as in

MPE/iX:	QDESIGN INFO="DICT=MYDICT"
OpenVMS, UNIX, Windows:	qdesign dict=mydict

MPE/iX

Some of the more common program parameters have been included in the PowerHouse UDC, which allow you to say, for example:

```
QDESIGN DICT=MYDICT
```

HELP QDESIGN at the operating system prompt shows the program parameters included in the UDC.

Exiting QDESIGN

To return to the operating system when you finish a QDESIGN session, enter the EXIT or QUIT statement.

Running QUICK

To run QUICK from the operating system prompt, enter:

```
quick
```

To run QUICK from within QDESIGN, enter:

```
go filename
```

To run QUICK from within another PowerHouse component, enter:

MPE/iX:	: quick
OpenVMS:	$ quick or ! quick
UNIX, Windows:	! quick

To run QUICK on Windows:

1. Select the QUICK option from the PowerHouse <*version*> option of the Start Menu.

   A valid dictionary must be specified using the PHD environment variable. If the environment variable is not specified, QUICK tries to locate a dictionary named PHD.PDC in the current directory. If a valid dictionary is not found, the Console window closes. If a valid dictionary is found, you enter the QUICK program and are prompted for a screen name.
2. From the Programs option of the Start Menu, select the Command Prompt option. A Command Prompt window appears. From the Command Prompt window, enter:
   ```
   quick
   ```

Entering QUICK Program Parameters

You can include program parameters when running QUICK from the operating system prompt or from within another component.

All valid program parameters can be accessed as in

MPE/iX:	QUICK INFO="DICT=MYDICT"
OpenVMS, UNIX, Windows:	quick dict=mydict

MPE/iX

Some of the more common program parameters have been included in the PowerHouse UDC, which allow you to say, for example:

```
QUICK DICT=MYDICT
```

HELP QUICK at the operating system prompt shows the program parameters included in the UDC.

Exiting QUICK

To return to the operating system when you finish a QUICK session, enter the Return to Previous Screen command (^) in the Action field. For a complete list of QUICK screen commands, see Chapter 2, "QUICK User Interface", in the *QDESIGN Reference* book.

Running QUICK Debugger

For information about running QUICK Debugger, see Chapter 9 "Debugger", and Chapter 10, "Debugger Commands", in the *QDESIGN Reference* book.

Running QUIZ

To run QUIZ from the operating system prompt, enter:

```
quiz
```

To run QUIZ from within another PowerHouse component, enter:

MPE/iX:	: quiz
OpenVMS:	$ quiz or ! quiz
UNIX, Windows:	! quiz

To run QUIZ on Windows:

1. Select the QUIZ option from the PowerHouse <*version*> option of the Start Menu.
2. From the Programs option of the Start Menu, select the Command Prompt option. A Command Prompt window appears. From the Command Prompt window, enter:
   ```
   quiz
   ```

Entering QUIZ Program Parameters

You can include program parameters when running QUIZ from the operating system prompt or from within another component.

All valid program parameters can be accessed as in

MPE/iX:	QUIZ INFO="DICT=MYDICT"
OpenVMS, UNIX, Windows:	quiz dict=mydict

MPE/iX

Some of the more common program parameters have been included in the PowerHouse UDC, which allow you to say, for example:

```
QUIZ DICT=MYDICT
```

HELP QUIZ at the operating system prompt shows the program parameters included in the UDC.

Exiting QUIZ

To return to the operating system when you finish a QUIZ session, enter the EXIT or QUIT statement.

QUIZ Error Status Settings (MPE/iX, UNIX, Windows)

Conditional error status codes are set when you exit QUIZ. The error status setting depends on events that occur during the QUIZ session.

Setting (MPE/iX)	Setting (UNIX, Windows)	Meaning
-	0	Normal completion
WARN1	1	Parsing error
WARN2	2	Exception error
WARN3	3	Calculation error
WARN4	4	Edit error

Setting (MPE/iX)	Setting (UNIX, Windows)	Meaning
WARN5	*5*	File output error

To check the status, use:

MPE/iX: `SHOWJCW`

UNIX: `echo $status` (C shell) or `echo $?` (korn shell)

Windows: echo %ERRORLEVEL%

QUIZ Error Status Settings (OpenVMS)

Conditional error status codes are set when you exit QUIZ. The error status setting depends on events that occur during the QUIZ session. QUIZ sets the DCL symbol $STATUS to the error code that represents the last type of error encountered.

Setting	Meaning
19008008	An internal error prevented continuation of the program
19008010	An exception error has been encountered
19008018	A calculation error has been encountered
19008020	A data or parameter editing error has been encountered
19008028	A file access error has been encountered
19008030	Errors with parameters have been detected
19008038	An error was detected while outputting to a subfile
19008042	Errors have been encountered
19009000	Warnings have been encountered
00000001	Successful completion

Testing Error Status Settings

You can test whether the QUIZ session succeeded with the DCL command:

```
$IF $STATUS
```

You can test for particular values, as in

```
$ QUIZ AUTO=BUDGET_RUN
$ IF $STATUS .EQ. %X1900801A
$ THEN
$ WRITE SYS$OUTPUT "Calculation error encountered."
$ ENDIF
```

Running QTP

To run QTP from the operating system prompt, enter:

```
qtp
```

To run QTP from within another PowerHouse component, enter:

MPE/iX:	: qtp
OpenVMS:	$ qtp or ! qtp
UNIX, Windows:	! qtp

To run QTP on Windows:

1. Select the QTP option from the PowerHouse *<version>* option of the Start Menu.
2. From the Programs option of the Start Menu, select the Command Prompt option. A Command Prompt window appears. From the Command Prompt window, enter:
   ```
   qtp
   ```

Entering QTP Program Parameters

You can include program parameters when running QTP from the operating system prompt or from within another component.

All valid program parameters can be accessed as in

MPE/iX:	QTP INFO="DICT=MYDICT"
OpenVMS, UNIX, Windows:	qtp dict=mydict

MPE/iX

Some of the more common program parameters have been included in the PowerHouse UDC, which allow you to say, for example:

```
QTP DICT=MYDICT
```

HELP QTP at the operating system prompt shows the program parameters included in the UDC.

Exiting QTP

To return to the operating system when you finish a QTP session, enter the EXIT or QUIT statement.

QTP Error Status Settings (MPE/iX, UNIX, Windows)

Conditional error status codes are set when you exit QTP. The error status setting depends on events that occur during the QTP session.

Setting (MPE/iX)	Setting (UNIX, Windows)	Meaning
-	0	Normal completion
FATAL1	1	Parsing error
FATAL2	2	Exception error
FATAL3	3	Calculation error
FATAL4	4	Edit error

Setting (MPE/iX)	Setting (UNIX, Windows)	Meaning
FATAL5	*5*	File output error
FATAL6	*6*	Conditional termination

To check the status, use:

MPE/iX: `SHOWJCW`

UNIX: `echo $status` (C shell) or `echo $?` (korn shell)

Windows: echo %ERRORLEVEL%

QTP Error Status Settings (OpenVMS)

Conditional error status codes are set when you exit QTP. The error status setting depends on events that occur during the QTP session. QTP sets the DCL symbol $STATUS to the error code that represents the last type of error encountered.

Setting	Meaning
1900800A	An internal error prevented continuation of the program
19008012	An exception error has been encountered
1900801A	A calculation error has been encountered
19008022	A data or parameter editing error has been encountered
1900802A	A file access error has been encountered
19008032	Errors with parameters have been detected
1900803A	An error was detected while outputting to a subfile
19008042	Errors have been encountered
19009000	Warnings have been encountered
00000001	Successful completion

Testing Error Status Settings

You can test whether the QTP session succeeded with the DCL command:

```
$IF $STATUS
```

You can test for particular values, as in

```
$ QTP AUTO=BUDGET_RUN
$ IF $STATUS .EQ. %X1900801A
$ THEN
$ WRITE SYS$OUTPUT "Calculation error encountered."
$ ENDIF
```

Request Statistics

By default, QTP displays statistics, any errors, and the resulting action at the end of each request. In addition, a detailed update report is available as an option. You can override the default by using the **nostatistics** program parameter, the SET NOSTATISTICS statement, or the SET REPORT statement.

Standard request statistics contain a count of all data records read, processed, and updated in tabular form, as in

```
Executing request PROJECT-UPDATE ...
```

```
Records read:
   BILLINGS            14
Transactions processed: 14
Records processed: Added Updated Unchanged Deleted
   PROJECTS         0       1        13        0
Finished.
```

If the request has no name, QTP reports its number. (Each request within a run is numbered consecutively.)

Record-structures are listed under the "Records read:" heading in the order that they appear in the ACCESS statement. The counts under the "Records read:" heading include only data records that are actually read. If a transaction fails selection while being built, only the data records read to the point of failure are included in the counts. Data records read for lookup editing purposes aren't included.

The "Transactions processed:" heading lists the number of transactions that passed selection.

The "Records processed:" heading appears only if the request has an output phase. The counts under the "Records processed:" heading report the output actions that QTP performed. If a conditional update isn't performed, the count isn't incremented. The unchanged count reflects data records for which an update action was specified but not performed because no item values changed.

Record-structures are listed in the sequence in which the OUTPUT and SUBFILE statements are specified in the request. If a record-structure has an alias, the alias is listed. The last line, "Finished.", appears after all requests in a run have completed execution.

If the statistics are produced on a printer, the date, time, and system title are displayed at the top of each page. The run and request names, if any, are also displayed.

Detailed Update Report

You can request a detailed report of update activity using the SET REPORT DETAIL statement. Every addition, update, or deletion is listed, as in

```
Record updated.   [9]
   File: EMPLOYEES
      Linkitem: EMPLOYEE 00017
      Linkitem:LASTNAME  ARBA
Record updated.   [14]
   File: EMPLOYEES
      Linkitem: EMPLOYEE 00002
      Linkitem: LASTNAME ANDERSON
```

The first line shows the action taken and the current transaction count. In the example, the EMPLOYEES record-structure was updated at control breaks, which occurred at transaction number 9 and 14. The record-structure name is shown along with its indexes and their values.

Determining Request Status at Execution Time

A snapshot of the current status of a request is obtained by entering a user break, (Ctrl-Y, **MPE/iX**; Ctrl-C, **UNIX**, **Windows**, **OpenVMS**), during request execution. When you enter a user break, QTP displays one of the following messages:

```
In GLOBAL phase, no records have been processed.
In LOAD phase, no records have been processed.
In INPUT phase, the following is a report of status.
In SORT phase, the following is a report of status.
In OUTPUT phase, the following is a report of status.
In WRAPUP phase, no statistics are available.
```

If the request is in the global or load phase, either the request tables are still being loaded or actual processing hasn't started. If the request is in the input, sort, or output phase, a status report is displayed that's identical in format to the statistics produced at the end of a request. The wrap-up phase occurs immediately after request processing has been completed and statistics have been displayed.

Following the display, QTP asks if you want to continue execution. A negative response (or no response within three minutes) results in an exception error, and in the termination of the run or request. Run or request termination also occurs if there's no response within three minutes.

Running PDL

To run PDL from the operating system prompt, enter:

```
pdl
```

To run PDL from within another PowerHouse component, enter:

MPE/iX:	: pdl
OpenVMS:	$ pdl or ! pdl
UNIX, Windows:	! pdl

To run PDL on Windows:

1. Select the PDL option from the PowerHouse <*version*> option of the Start Menu.
2. From the Programs option of the Start Menu, select the Command Prompt option. A Command Prompt window appears. From the Command Prompt window, enter:
   ```
   pdl
   ```

For more information about PDL, see Chapter 1, "Introducing the PowerHouse Dictionary", in the *PDL and Utilities Reference* book.

Entering PDL Program Parameters

You can include program parameters when running PDL from the operating system prompt or from within another component.

All valid program parameters can be accessed as in

MPE/iX:	PDL INFO="DICT=MYDICT"
OpenVMS, UNIX, Windows:	pdl dict=mydict

MPE/iX

Some of the more common program parameters have been included in the PowerHouse UDC, which allows you to say, for example:

```
PDL DICT=MYDICT
```

HELP PDL at the operating system prompt shows the program parameters included in the UDC.

Exiting PDL

To return to the operating system when you finish a PDL session, enter the EXIT or QUIT statement.

Running PHDPDL (OpenVMS)

To run PHDPDL from the operating system prompt, enter

```
phdpdl
```

To run PHDPDL from within another component, enter

```
$ phdpdl
```

or

```
! phdpdl
```

For more information about PHDPDL, see Chapter 1, "Introducing the PowerHouse Dictionary", in the *PDL and Utilities Reference* book.

Entering PHDPDL Program Parameters

You can include program parameters when running PHDPDL from the operating system prompt or from within another component.

All valid program parameters can be accessed as in

```
phdpdl dict=mydict
```

Exiting PHDPDL

To return to the operating system when you finish a PHDPDL session, enter the EXIT or QUIT statement.

Running QSHOW

To run QSHOW from the operating system prompt, enter:

```
qshow
```

To run QSHOW from within QDESIGN, QUIZ or QTP, enter QSHOW from the product prompt or:

MPE/iX:	: qshow
OpenVMS:	$ qshow or $phshow
UNIX, Windows:	! qshow

To run QSHOW on Windows:

1. Select the QSHOW option from the PowerHouse <*version*> option of the Start Menu.
2. From the Programs option of the Start Menu, select the Command Prompt option. A Command Prompt window appears. From the Command Prompt window, enter:
   ```
   qshow
   ```

Entering QSHOW Program Parameters

You can include program parameters when running QSHOW from the operating system prompt or from within another component.

All valid program parameters can be accessed as in

MPE/iX:	QSHOW INFO="DICT=MYDICT"
OpenVMS, UNIX, Windows:	qshow dict=mydict

MPE/iX

Some of the more common program parameters have been included in the PowerHouse UDC, which allows you to say, for example:

```
QSHOW DICT=MYDICT
```

HELP QSHOW at the operating system prompt shows the program parameters included in the UDC.

Exiting QSHOW

To return to the invoking program when you finish a QSHOW session, enter the EXIT or QUIT statement.

Running QUTIL

To run QUTIL at the operating system prompt, enter

```
qutil
```

To run QUTIL from within QDESIGN, QUIZ or QTP, enter:

MPE/iX:	: qutil
OpenVMS:	$ qutil or ! qutil
UNIX, Windows:	! qutil

To run QUTIL on Windows:

1. Select the QUTIL option from the PowerHouse <*version*> option of the Start Menu.
2. From the Programs option of the Start Menu, select the Command Prompt option. A Command Prompt window appears. From the Command Prompt window, enter:
   ```
   qutil
   ```

Entering QUTIL Program Parameters

You can include program parameters when running QUTIL from the operating system prompt or from within another component.

All valid program parameters can be accessed as in

MPE/iX:	QUTIL INFO="DICT=MYDICT"
OpenVMS, UNIX, Windows:	qutil dict=mydict

MPE/iX

Some of the more common program parameters have been included in the PowerHouse UDC, which allows you to say, for example:

```
QUTIL DICT=MYDICT
```

HELP QUTIL at the operating system prompt shows the program parameters included in the UDC.

Exiting QUTIL

To return to the operating system when you finish a QUTIL session, enter the EXIT or QUIT statement.

PowerHouse Commands (OpenVMS)

The PowerHouse commands are:

@SETPOWERHOUSE

POWERHOUSE

SETDICTIONARY

SHOWDICTIONARY

SHOWPOWERHOUSE

SHOWQUOTA

The syntax of the PowerHouse commands is presented on the following pages.

You can obtain information about a command by entering a question mark after it, as in

```
$ SETDICTIONARY ?
```

The question mark works for PowerHouse command procedures such as SETDICTIONARY but not for PowerHouse images like QUIZ or QTP.

Any PowerHouse command can be placed in your login file so that it executes automatically when you log on.

Limit: You must first execute the @SETPOWERHOUSE command in order to use any of the other PowerHouse commands.

@SETPOWERHOUSE (OpenVMS)

Executes a PowerHouse command procedure that sets up the environment for using PowerHouse.

Syntax

@SETPOWERHOUSE [version] [LANGUAGE=E|F|D]
[LICENSE=DEVELOPMENT|RUNTIME|RT_REPORTING|
REPORTING_ONLY] [VERIFY|NOVERIFY]

@

Indicates to OpenVMS that the command invokes a command procedure. You only have to use ampersand (@) once in a session.

Limit: You must use @ when you execute the SETPOWERHOUSE command for the first time in a session.

version

Specifies the version of PowerHouse to be run. Check with your system manager to find out which version to use. The logical name PH_DEFAULT_VERSION points to the default version.

LANGUAGE=E|F|D

Specifies which message files to use. The language may be set to one of English (E), French (F), or German (D).

Default: English (E)

LICENSE=DEVELOPMENT|RUNTIME|RT_REPORTING|REPORTING_ONLY

Overrides the license type that is set up by the logical PH_DEFAULT_LICENSE.

VERIFY|NOVERIFY

Specifies whether process quotas are checked or not.

Default: NOVERIFY

Discussion

The @SETPOWERHOUSE command invokes a command procedure that sets up the environment for using PowerHouse. It establishes the logical names and global symbols that PowerHouse needs. This command is typically included in your login file.

The command procedure can be used more than once in a session to change to a different version of PowerHouse or to restore global symbols and logical names that have been changed during the session. After the first time this command procedure is executed, the command to invoke it can be abbreviated to

```
SETPOW [VERSION]
```

When the command procedure is run the first time, a global symbol for SETPOWERHOUSE is established. The @ is no longer required in subsequent commands.

By default, the @SETPOWERHOUSE command selects the EDT editor. You can change the default editor by changing the PHEDIT symbol.

POWERHOUSE (OpenVMS)

The POWERHOUSE command initiates the PowerHouse Menu.

Syntax

POWERHOUSE [option]...

ACTIONBAR

Runs all the PowerHouse Menu screens in Action Bar mode, rather than the default Action field mode.

CHECKSUM710[=ON|OFF]

ON means PowerHouse uses unsigned input for checksum calculations. OFF means PowerHouse uses signed input.

Specifying CHECKSUM710 without an option is the same as specifying CHECKSUM710=ON.

Default: ON (version 7.10); OFF (versions 8.00 and up)

If the logical CHECKSUM710 is set to the option you require, then the PHDMAINT parameter does not need to be used. For further information on this parameter, see the corresponding CHECKSUM710 program parameter in the *PowerHouse Rules* document.

[TERM=] termtype

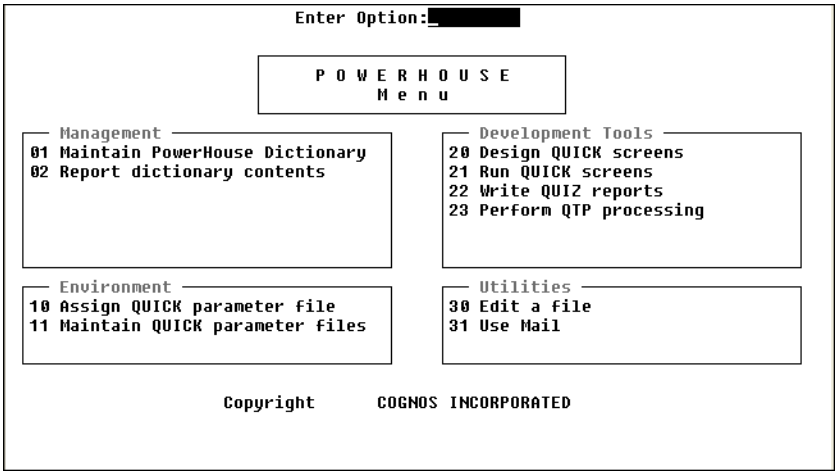
Identifies your terminal type. If you omit this parameter, you will be prompted for your terminal type as PowerHouse can't recognize it automatically.

USER|AMGR|DMGR

Instructs PowerHouse to skip over the PowerHouse Menu and move you directly to the User Menu (USER), the Application Management Menu (AMGR), or the Dictionary Management Menu (DMGR) provided you have access to the dictionary at the necessary security level. If you are denied access to a menu, the PHD Screen System displays an appropriate message.

The PowerHouse command initiates PowerHouse. If you have not previously used the SETDICTIONARY command in the current session, you are prompted for the name of the dictionary to be used. When you enter the name of your dictionary, the PowerHouse Menu or the menu appropriate to your dictionary security class appears. You can abbreviate POWERHOUSE to POW.

The PowerHouse Menu looks like this:



To use a component or utility, type the option number in the Action field and then press [Return].

The options on the PowerHouse menu are:

01 Maintain PowerHouse Dictionary

Moves you to the menu appropriate to your dictionary security class. If your security class permits it, you can override this automatic assignment by entering one of the following commands in the action field of the PowerHouse menu.

- USER moves you to the User Menu.
- AMGR moves you to the Application Management Menu.
- DMGR moves you to the Dictionary Management Menu.

02 Report Dictionary Contents

Runs QSHOW so you can report the contents of your dictionary. Reports can be displayed on your terminal or directed to a printer.

10 Assign QUICK Parameter File

Establishes the QKGO file to be used by QUICK during this session. A QKGO file sets many of the parameters QUICK uses when running your applications.

11 Maintain QUICK Parameter Files

Moves to the QKGO Construction and Maintenance Screen to create, view, or change QKGO files for QUICK. For more information about QKGO, see Chapter 6, "Customizing QUICK with QKGO", in the *QDESIGN Reference* book.

20 Design QUICK Screens

Runs QDESIGN, the QUICK screen builder. Refer to the *QDESIGN Reference* for complete details.

21 Run QUICK Screens

Runs the QUICK screens that you have designed. QUICK accesses the data defined in your data dictionary.

22 Write QUIZ Reports

Reports the data stored in your files.

23 Perform QTP Processing

Runs QTP for large volume data processing, such as bulk data validation and other batch-oriented tasks.

30 Edit a File

Invokes your system editor. You can use the editor for writing and changing statements used by the PowerHouse components.

31 Use Mail

Invokes OpenVMS mail.

Discussion

The USER, AMGR, and DMGR Action Field Commands

The PowerHouse Menu provides three Action field commands - USER, AMGR and DMGR - which allow dictionary users to choose which of the three menus they want to use.

- The AMGR command invokes the Application Management Menu.
- The DMGR command invokes the Dictionary Management Menu.
- The USER Action field command can be used to move to the basic User Menu when PHD Screen System automatically initiates a different menu.

Users whose dictionary security grants them application manager or dictionary manager status can enter AMGR or DMGR, respectively, in the Action field of the PowerHouse Menu instead of choosing option 01. Users who don't have the appropriate dictionary access can't use the AMGR and DMGR commands.

SETDICTIONARY (OpenVMS)

Establishes a dictionary for a PowerHouse session.

Syntax

SETDICTIONARY [dictionaryname [TYPE PHD|PDC] [NOCHECK]]

dictionaryname

The name of the dictionary to be used for the session. The dictionary name does not include the automatically assigned numbers (00 through 03) or the file extension (.PHD) of the dictionary files.

Limit: The dictionary name can be up to 37 characters long.

TYPE PHD|PDC

Specifies the default dictionary type. 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 for a PDC dictionary.

NOCHECK

Prevents the SETDICTIONARY procedure from checking that the dictionary files exist and are valid. You can use this option to save machine overhead.

Discussion

The SETDICTIONARY command establishes which dictionary PowerHouse is to use. You can abbreviate SETDICTIONARY to SETDICT.

The SETDICTIONARY command sets the logical name assignments for the dictionary files. If you enter it without a dictionary name, as in

```
$ SETDICT
```

you will be prompted for the dictionary name. If you then press [Return] without entering a name, the logical names for the dictionary files will be de-assigned.

To change dictionaries for the PHD Screen System or QUICK, you must leave PowerHouse and then reissue the SETDICTIONARY command, as in

```
$ SETDICT DICT2
```

In QDESIGN, QUIZ, QTP, PDL, and QSHOW, you can either change dictionaries by leaving PowerHouse and using the SETDICTIONARY command, or you can use the SET DICTIONARY statement while you are still in the component, as in

```
> SET DICT DICT2
```

Once you set a dictionary, all the PowerHouse components that you initiate from the same process level will use that dictionary. You are on one process level when you run any PowerHouse component from the DCL prompt ($), but you create a subprocess when you run a PowerHouse component from the PowerHouse Menu. For example, if you follow these steps:

1. $ SETDICT DICTA
2. $ QUIZ
3. > SET DICT DICTB
4. > EXIT
5. $ QSHOW

QSHOW will use DICTB. Similarly, if you follow these steps:

1. $ SETDICT DICTA
2. $ POWERHOUSE
3. select option 4 to run QUIZ
4. > SET DICT DICTB

5. > EXIT
6. select option 9 to run QSHOW

QSHOW will use DICTB, because both QUIZ and QSHOW were run as subprocesses. However, if you follow these steps:

1. $ SETDICT DICTA
2. $ POWERHOUSE
3. select option 4 to run QUIZ
4. > SET DICT DICTB
5. > EXIT
6. exit from the PowerHouse Menu
7. $ QSHOW

QSHOW will use DICTA, because DICTB was set at a subprocess level but QSHOW is not a subprocess.

If you follow these steps:

1. $ SETDICT DICTA
2. $ QUIZ
3. > SET DICT DICTB
4. > EXIT
5. $ SHOWDICT

the SHOWDICT command shows the dictionary currently in use as DICTB. If QUIZ is run using the **dictionary** program parameter as follows, the current dictionary is DICTB:

1. $ SETDICT DICTA
2. $ QUIZ DICT=DICTB
3. > EXIT
4. $ SHOWDICT

Limit: In PHDPDL you cannot set a dictionary to one that is already in use.

SHOWDICTIONARY (OpenVMS)

Displays the name of the dictionary currently in use.

Syntax

SHOWDICTIONARY

Discussion

The SHOWDICTIONARY command displays the name and type of the dictionary currently in use for a PowerHouse session. You can abbreviate SHOWDICTIONARY to SHOWDICT.

SHOWPOWERHOUSE (OpenVMS)

Displays the name of the active version of PowerHouse.

Syntax

SHOWPOWERHOUSE

Discussion

The SHOWPOWERHOUSE command displays the name of the version of PowerHouse that you are running. If the active version is different from the default version, the default version is also displayed, as in

```
DISK: PATH$RELEASE:
ROOT DIRECTORY: (RELEASE)
ACTIVE VERSION: 830C
DEFAULT VERSION: 830C
ACTIVE LICENSE: DEVELOPMENT
```

You can abbreviate SHOWPOWERHOUSE to SHOWPOW.

SHOWQUOTA (OpenVMS)

Displays the remaining process quotas, and the minimum PowerHouse requirements for them.

Syntax

SHOWQUOTA

Discussion

The SHOWQUOTA command displays the process quotas and the minimum PowerHouse requirements for BYTLM, ENQLM, FILLM, and PRCLM.

Locating Files

The term "location" in this section means:

MPE/iX:	group, or group and account
OpenVMS:	node, device, and/or directory
UNIX, Windows:	path possibly including the root or device

Qualifying, or partially qualifying a file, means adding a location or partial location to the file name. Partial locations are always relative to the current working location.

If specified, the **procloc** program parameter takes precedence over the LOCATION PROCESS resource file statement.

Locating the Data Dictionary

For PowerHouse components that access a dictionary file, the component may be run with the **dictionary** program parameter to specify the name of a dictionary file. The parameter can be abbreviated to **dict**. For example,

```
qdesign dict=mydict
```

OpenVMS, UNIX, Windows: If the name supplied by the **dictionary** program parameter, logical name (**OpenVMS**), the environment variable (**UNIX, Windows**), or SET DICT statement doesn't end with the extension .pdc, PowerHouse appends this extension before using it. **OpenVMS:** This applies only to PDC-type dictionaries. For PHD-type dictionaries, it will append the appropriate file number (00-03) and the extension .phd.

If the **dictionary** program parameter is not specified on startup:

MPE/iX, UNIX, Windows:	PowerHouse first looks for the designated file PHD. If not found, PowerHouse looks for the file PHD in the current location. If not found, PowerHouse issues an error.
OpenVMS:	If no dictionary type is specified, PowerHouse first looks for a PHD-type dictionary by looking for • the logical name PHD00 • and if not found, the files PHD00-PHD03.PHD in the current location If a PHD-type dictionary is not found, PowerHouse looks for a PDC-type dictionary by looking for • the logical name PHD • and if not found, the file PHD.PDC in the current location If the file is still not found, PowerHouse issues an error. If a dictionary type is specified, PowerHouse performs the same steps as indicated when no dictionary type is specified, but restricts the search to the specified dictionary type, PHD or PDC.

How the BUILD and SAVE Statements Locate Files

The BUILD statement causes QDESIGN to create compiled screens. The BUILD statement causes QUIZ and QTP to create compiled files.

The BUILD statement in QDESIGN references the file named in the SCREEN statement. If the BUILD statement in QTP does not have a file name specified, it references the file named in the RUN statement.

The SAVE statement causes PDL, PHDPDL, QDESIGN, QSHOW, QTP, QUIZ, and QUTIL to create a file of source statements.

If you qualify or partially qualify the file name, the component only searches in the specified location. A partially qualified location is relative to the current working location. If you reference a file without a qualifier, then the component tries to locate the file in the current working location.

MPE/iX:	Only files with the appropriate file type are processed.
OpenVMS, UNIX, Windows:	The component appends the appropriate file extension to the name only if none is given in the entered name. Only files with the appropriate extension are processed.

If the component finds the file, then it replaces it (**MPE/iX, UNIX, Windows**) or creates a new version of the existing file (**OpenVMS**). The component prompts for confirmation before replacing or creating a new version of the existing file if SET VERIFY DELETE is specified. If SET NOVERIFY DELETE is specified, the component replaces or creates a new version of the file automatically. If you choose not to replace or create a new version of the existing file, then the component leaves the existing file intact and creates a temporary copy of the new file.

For qualified or partially-qualified files, if the location exists but the file is not found, then the component creates the named file in the specified location. If the location is not found, the component issues an error.

For unqualified files, if the file is not found, then the component creates the named file in the current working location.

How the EXECUTE and USE Statements and auto Program Parameter Locate Files

The EXECUTE statement in QTP and QUIZ reads compiled run and report files. The USE statement in PDL, PHDPDL, QDESIGN, QSHOW, QTP, QUIZ, and QUTIL reads source statement files. The USE statement in QTP and QUIZ also reads compiled run and report files. The **auto** program parameter in PDL, PHDPDL, QDESIGN, QSHOW, QTP, QUIZ, and QUTIL reads source statement files. The **auto** program parameter in QTP and QUIZ also reads compiled run and report files.

MPE/iX:	Only files with the appropriate file type are processed.
OpenVMS, UNIX, Windows:	The component appends the appropriate file extension to the name only if none is given in the entered name. Only files with the appropriate extension are processed.

In QTP and QUIZ, if a file extension is specified (**OpenVMS**, **UNIX**, **Windows**), the USE statement and the **auto** program parameter look for that type of file only. If no file extension is specified, the component first looks for a source statement file. If none is found, the component then looks for compiled run or report files. If the compiled file is not found, the component issues an error.

Locating Specified Files

If neither the **procloc** program parameter nor the LOCATION PROCESS resource file statement is used, then the component looks for the file in the location specified if qualified, or, if partially or not qualified, relative to the current working location. If the file is not found, the component issues an error.

If the file is fully qualified or partially qualified (**MPE/iX**, **OpenVMS**), any **procloc** program parameter or LOCATION PROCESS resource file statement is ignored. The component looks for the file in the specified location. A partially qualified location is relative to the current working location. If the file is not found, the component issues an error.

If the file is unqualified and either the **procloc** program parameter or the LOCATION PROCESS resource file statement is used, the component looks for the file in the **procloc** or the LOCATION PROCESS location. If the file is not found, it is searched for relative to the current working location. If the file is still not found, the component issues an error.

UNIX, Windows: If the file is partially qualified and the **procloc** program parameter or LOCATION PROCESS resource file statement is used, then the component looks for the file in the partial location relative to the **procloc** or LOCATION PROCESS location. If the file is not found, the component looks for the file relative to the current working location. If the file is still not found, the component issues an error.

Locating Files When the Component Starts

If the **auto** program parameter is not used, the component looks for its designated USE file. For a list of designated files, see "Designated Files" (p. 48). The file is located as if it had been specified in the **auto** program parameter, except that if the component does not locate the file, it does not issue an error.

Locating Start Screens or QKGO files in QUICK

If no file extension is specified on the file name, QUICK looks for a QKI file first, then a QKGO file, and lastly for a compiled screen file. The default extension for QKGO files is .qkg (**OpenVMS, UNIX, Windows**). The default extension for QKI files is .qki (**OpenVMS, UNIX, Windows**). The default extension for compiled screen files is .qkc (**OpenVMS, UNIX, Windows**).

If a QKGO file is specified and it contains a start or first screen, that screen is located in the same manner as if the screen had been specified in the **auto** program parameter. If there is no start screen, QUICK prompts for a screen name.

If neither the **procloc** program parameter nor the LOCATION PROCESS resource file statement is used, then QUICK looks for the screen or QKGO file specified in the **auto** program parameter in the specified location if qualified, or, if partially or not qualified, relative to the current working location. If the file is not found, QUICK issues an error.

Locating Qualified or Partially Qualified Files

If the screen or QKGO file specified in the **auto** program parameter is fully qualified or partially qualified (**MPE/iX, OpenVMS**), any **procloc** program parameter or LOCATION PROCESS resource file statement is ignored. QUICK looks for the file in the specified location. A partially qualified location is relative to the current working location. If the file is not found, QUICK issues an error.

UNIX, Windows: If the screen or QKGO file is partially qualified and the **procloc** program parameter or LOCATION PROCESS resource file statement is used, then QUICK looks for the file in that location. If the file is not found, QUICK looks for the file relative to the current working location. If the file is still not found, QUICK issues an error.

Locating Unqualified Files

If the screen or QKGO file specified in the **auto** program parameter is unqualified and either the **procloc** program parameter or the LOCATION PROCESS resource file statement is used, QUICK looks for the file in the **procloc** or the LOCATION PROCESS location. If the file is not found, QUICK looks for the file relative to the current working location. If the file is still not found, QUICK issues an error.

Locating the Start Screen or QKGO File When No File is Specified

If the **auto** program parameter is not used, QUICK looks for the designated file, QKGO. QUICK looks for the file as if it had been specified in the **auto** program parameter, except that if QUICK does not locate the file or there is no start screen, it prompts for a screen name rather than issuing an error.

OpenVMS: If there is no designated file QKGO specified, QUICK looks for the logical PH_DEFAULT_QKGO.

PH_DEFAULT_QKGO May Take Precedence Over QKGO (OpenVMS)

If the **auto** program parameter is specified for a compiled QUICK screen file and the logical PH_DEFAULT_QKGO points to a QKGO file, then the specified screen is run and the QKGO file referenced by the PH_DEFAULT_QKGO logical is used, except for its First Screen setting. This overrides any QKGO file that may be pointed to by the logical QKGO.

How the GO Statement Locates Files

The GO statement in QDESIGN invokes QUICK and passes a screen file or QKGO file using the **auto** program parameter. If the **procloc** program parameter was specified to QDESIGN, it is also passed to QUICK.

If the GO statement does not name a screen, then QUICK attempts to locate the screen file named in the SCREEN statement. If QDESIGN finds the file, it creates a temporary version of the screen to pass to QUICK.

If you name a screen or a QKGO file on the GO statement, QUICK processes the **auto** program parameter in the same manner as if QUICK had been started on its own.

For example, if in QDESIGN you enter the following statement:

```
> GO MYSCREEN
```

QDESIGN invokes QUICK with the **auto** program parameter as follows:

```
> QUICK AUTO=MYSCREEN
```

If there is no current screen when you enter the GO statement, and you do not name a screen or QKGO file, then QUICK prompts you for a screen name. This also happens if you specify a QKGO file and the QKGO file does not specify a start screen.

Locating Subfiles

When creating a subfile, PowerHouse first attempts to locate a like-named subfile. When locating subfiles, PowerHouse looks first for a temporary subfile and then for a permanent subfile.

Creating a Subfile With the Same Name as an Existing Subfile

If you create a temporary subfile with the same name and structure as an existing permanent subfile, PowerHouse clears the permanent subfile of data and uses the permanent subfile. If the structures are different, PowerHouse issues an error.

The following table shows what happens when you create temporary subfiles:

If you specify ...	and a temporary subfile with the same name exists ...	and a temporary subfile with the same name doesn't exist ...	and a non-PowerHouse temporary file with the same name exists ...
TEMPORARY and NOAPPEND	PowerHouse deletes the temporary subfile, issues a warning, and creates a new temporary subfile	PowerHouse creates a temporary subfile	PowerHouse issues an error
TEMPORARY and APPEND	PowerHouse appends to the temporary subfile	PowerHouse creates a temporary subfile	PowerHouse issues an error

If you create a permanent subfile with the same name as an existing temporary subfile, PowerHouse deletes the temporary subfile.

The following table shows how PowerHouse creates permanent subfiles:

If you specify ...	and a permanent subfile with the same name exists ...	and a permanent subfile with the same name doesn't exist ...	and a non-PowerHouse permanent file with the same name exists ...
KEEP and NOAPPEND	PowerHouse prompts you: • to delete the existing permanent subfile (**MPE/iX, UNIX, Windows**) • to create a new version (**OpenVMS**)	PowerHouse creates a permanent subfile	PowerHouse issues an error
KEEP and APPEND	PowerHouse appends to the permanent subfile	PowerHouse creates a permanent subfile	PowerHouse issues an error

Locating Subfiles (MPE/iX)

PowerHouse looks for permanent subfiles in the permanent domain. PowerHouse looks for temporary subfiles in the temporary domain.

If a subfile is qualified, then PowerHouse looks for the subfile in the specified location. If a subfile is partially qualified, then PowerHouse looks for the subfile in the specified location relative to the current working location. If a subfile is unqualified, then PowerHouse looks for the subfile in the current working location.

Locating Subfiles (OpenVMS)

If a permanent subfile is qualified, then PowerHouse looks for the subfile in the specified location. If a permanent subfile is partially qualified, then PowerHouse looks for the subfile in the specified location relative to the current working location. If a permanent subfile is unqualified, then PowerHouse looks for the subfile in the current working location.

PowerHouse looks for temporary subfiles in the SYS$SCRATCH area regardless of path qualification. Any path qualification is ignored for temporary subfiles.

Creating Temporary Directories (UNIX, Windows)

When a component starts, PowerHouse creates a temporary directory to hold temporary files, such as the source statement save file and interim and temporary subfiles. The name of the directory is phnnnn.tmp where nnnn is the process-id. **UNIX**: If a component is started from within another component, the temporary directory is shared. **Windows**: Each component creates its own temporary directory regardless of where it was started.

UNIX: By default, the temporary directory is created in the current working location. If the PHTEMP environment variable is set, temporary files are created in the location specified by PHTEMP and the temporary directories are not created.

Windows: By default, the temporary directory is created in the Windows temporary directory established by checking the following in order: the TEMP environment variable, the TMP environment variable, the USERPROFILE environment variable, the current working directory for the user. If the PHTEMP environment variable is set, the temporary directory is created in the location specified by PHTEMP.

Locating Interim and Temporary Subfiles (UNIX, Windows)

Any path qualification is ignored for temporary subfiles.

If the PHTEMPKEEP environment variable is not set, PowerHouse looks for interim and temporary subfiles in the phnnnn.tmp directory.

If both the PHTEMPKEEP and PHTEMP environment variables are set, interim and temporary subfiles are created and looked for in the PHTEMP location rather than in the phnnn.tmp directory. In this case, PowerHouse does not delete interim and temporary subfiles when the components exit, and they are thus available for subsequent components to use. If you use this feature, ensure that you delete interim and temporary subfiles once they are no longer needed. Other temporary files are deleted as are any temporary directories. The PHTEMPKEEP environment variable can be set to any value.

If the PHTEMP environment variable is not set, the PHTEMPKEEP environment variable is ignored with one exception. When a component starts, it sets the PHTEMP location as part of its processing. That component is aware that the PHTEMP setting is not due to the environment variable; however, if that component calls another component, it sets the PHTEMP environment variable so that the second component can use the same location. If PHTEMPKEEP is set, results can be unpredictable. Avoid this situation by ensuring that PHTEMPKEEP is only set if PHTEMP is also set.

Locating Permanent Subfiles (UNIX, Windows)

If a permanent subfile is qualified, then PowerHouse looks for the subfile in the specified location. If a permanent subfile is partially qualified, then PowerHouse looks for the subfile in the specified location relative to the current working location.

UNIX: If a permanent or portable subfile is unqualified, PowerHouse looks for the subfile in the current working location.

Windows: If a permanent or portable subfile is unqualified, PowerHouse looks for the subfile in the location specified by the PH_SBF_LOC environment variable. If the PH_SBF_LOC environment variable is not set, PowerHouse looks for the subfile in the current working directory.

Locating ODS5 File Names (OpenVMS)

PowerHouse 8.40 partially supports ODS5 file names. As well, the maximum length of a filename or file specification remains 70 characters. Due to the change in OpenVMS system services, PowerHouse can no longer validate a file name as being restricted to the ODS2 format. Due to the flexibility of ODS5 file names, complete testing is not practical. Using ODS5 file names in a manner that is not specifically mentioned below may not work as expected.

Dictionary names

PHDPDL

Embedded periods are not supported. Embedded blanks and non-ODS2 characters are supported.

PDL

Embedded blanks are not supported. Embedded periods are supported but the extension .PDC must be included for complete support. Without the .PDC extension, SETDICT will not work with filenames with embedded periods. Non-ODS2 characters are supported.

POW

ODS5 file names are not supported.

Data file names

Include the .DAT extension if you are using embedded periods. Embedded blanks are not supported if you are using QUTIL to create the file.

Compiled screen names

If you use ODS5 file names SYS$SCRATCH must point to an ODS5 drive as well so that the temporary files can be created correctly. Embedded periods are not supported.

Source file names and compiled run and request names

Embedded periods, blanks and non-ODS2 characters are supported based on our testing. As previously mentioned, we cannot guarantee complete support. If embedded periods are used and are not preceded by "^" then the last period is considered to begin the extension. If in doubt, specify the extension. If the embedded periods are preceded by "^" then PowerHouse adds the appropriate extension.

Subfile names

ODS5 file names are not supported.

Designated Files

Designated files are reserved for use as PowerHouse default files. Some of these names are reserved for use as file equations (**MPE/iX**), logical names (**OpenVMS**) or environment variables (**UNIX, Windows**) by PowerHouse. For **UNIX,** this is the uppercase versions of the names. (**MPE/iX**, **OpenVMS**, and **Windows** are not case sensitive.)

Where there is an entry in the File Equation/Env.Variable/Logical Name column in the table below, the product looks for a file equation/environment variable/logical with that name. If the variable is defined, the product uses its value in place of the lowercase file name.

OpenVMS, UNIX, Windows: PowerHouse uses file extensions when searching for files. For example, when you use the SAVE statement in QTP, PowerHouse creates a file containing the source code in the current dictionary and appends the file extension .qts to the filename. When PowerHouse searches for the file, it first searches for a filename that has the extension .qts appended.

File	File Ext.[1]	File Equation/ Env. Variable/ Logical Name[2]	Component	Purpose
filenameb filenameb	.dat .idx		QUICK	The context binding files in QKGO file sets (exist when TIC information is specified).
filenamek filenamek	.dat .idx		QUICK	The key sequence files in QKGO file sets (exist when TIC information is specified).
filenamet filenamet	.dat .idx		QUICK	The terminal-group files in QKGO file sets (exist when TIC information is specified).
pdlmsg	.txt	PDLMSG	PDL	Optional message file that, if present, contains messages to be used by PDL.
pdlsave	.pdl	PDLSAVE	PDL	Temporary save file to which entered PDL statements are written.
pdluse	.pdl	PDLUSE	PDL	If this optional USE file exists, it is processed automatically before opening the system input file (unless the **auto** program parameter has been specified).
phd	.pdc	PHD	All	The default PDC type dictionary file
phd00-phd03 (**OpenVMS**)	.phd	PHD00-PHD03	All	The default PHD type dictionary files
phdpdlmsg (**OpenVMS**)	.txt	PHDPDLMSG	PHDPDL	Optional message file that, if present, contains messages to be used by PHDPDL.
phdpdlsave (**OpenVMS**)	.pdl	PHDPDLSAVE	PHDPDL	Temporary save file to which entered PHDPDL statements are written.

File	File Ext.[1]	File Equation/ Env. Variable/ Logical Name[2]	Component	Purpose
phdpdluse (**OpenVMS**)	.pdl	PHDPDLUSE	PHDPDL	If this optional USE file exists, it is processed automatically before opening the system input file (unless the **auto** program parameter has been specified).
PHFORM	.rpo		QDESIGN	The default forms library when creating Axiant Client forms.
PHRS (**MPE/iX**) ph (**OpenVMS, Windows**) .phrc (**UNIX**)	. .rc	PHRS	All	The default resource file containing program parameters and other system characteristics.
qkdmsg	.txt	QKDMSG	QDESIGN	An optional message file that, if present, contains messages to be used by QDESIGN.
qkdebug	.txt	QKDEBUG	QUICK	The file containing the trace log of a Debugger session.
qkecho	.txt	QKECHO	QUICK	The file to which user responses are echoed. The echo file is created only if a file equation (**MPE/iX**), logical name (**OpenVMS**) or environment variable (**UNIX, Windows**) has been assigned to QKECHO. This file is useful for building QKIN files including batch updates or automated testing for future use.
qkgo	.qkg	QKGO	QUICK	The execution-time parameters file governing QUICK's operating characteristics.
qkin	.txt	QKIN	QUICK	An optional QUICK input file. Rerouting default input should be done through QKIN alone.If QKIN isn't specified, input is taken from: **MPE/iX**: $SDTINX. If this is not $STDINX, trailing blanks are ignored. If trailing blanks are required, they should be followed by the semicolon separator character (;). **OpenVMS**: SYS$INPUT. QKIN allows QUICK to be run from command procedures.

File	File Ext.[1]	File Equation/ Env. Variable/ Logical Name[2]	Component	Purpose
qklist	.txt	QKLIST	QUICK	The default QUICK output file for the LIST and LISTALL command. If a logical name for this file is not defined, then the output is sent to standard printer.
qkmsg	.txt	QKMSG	QUICK	An optional message file that, if present, contains messages to be used by QUICK.
qkmsgdes	.txt	QKMSGDES	QUICK	An optional designer message file.
qkout	.txt	QKOUT	QUICK	An optional QUICK output file that contains text destined for the terminal. If a logical name for QKOUT isn't specified, terminal output is directed to the standard output.
qksave	.qks	QKSAVE	QDESIGN	A temporary save file to which entered QDESIGN statements are written.
qk_termtype	.txt	QK_TERMTYPE	QUICK	The optional message file used to set the default terminal and its attributes.
qkuse	.qks	QKUSE	QDESIGN	An optional USE file processed automatically by QDESIGN before opening the system input file (unless the **auto** program parameter has been specified).
qshogen	.pdl	QSHOGEN	QSHOW	Default output file for PDL statement generation.
qsholist	.txt	QSHOLIST	QSHOW	Default output file for QSHOW reports that are sent to disk.
qshomsg	.txt	QSHOMSG	QSHOW	Optional message that, if present, contains messages to be used by QSHOW.
qshosave	.qss	QSHOSAVE	QSHOW	Temporary save file to which entered QSHOW statements are written.
qshouse	.qss	QSHOUSE	QSHOW	Optional use file processed automatically by QSHOW before opening the system input file (unless the **auto** program parameter has been specified).
qtpjob	.txt	QTPJOB	QTP	An optional file that, if present, has its contents copied to QTPSAVE when the SET JOB statement is specified.

File	File Ext.[1]	File Equation/ Env. Variable/ Logical Name[2]	Component	Purpose
qtplist	.txt	QTPLIST	QTP	The default report output file when SET REPORT is specified.
qtpmsg	.txt	QTPMSG	QTP	An optional message file that, if present, contains messages to be used by QTP.
qtpmsgdes	.txt	QTPMSGDES	QTP	An optional message file that, if present, contains designer messages to be used by QTP.
qtpsave	.qts	QTPSAVE	QTP	A temporary save file to which entered QTP statements are written.
qtpuse	.qts	QTPUSE	QTP	An optional use file processed automatically by QTP before opening the system input file (unless the **auto** program parameter has been specified).
quizjob	.qzs	QUIZJOB	QUIZ	An optional file that, if present, has its contents copied to QUIZSAVE when the set job statement is specified.
quizlist	.txt	QUIZLIST	QUIZ	The default report output file.
quizmsg	.txt	QUIZMSG	QUIZ	An optional message file that, if present, contains messages to be used by QUIZ.
quizobj	.qzc	QUIZOBJ	QUIZ	Temporary save file to which the current QUIZ report is compiled.
quizsave	.qzs	QUIZSAVE	QUIZ	Temporary save file to which entered QUIZ statements are written.
quizuse	.qzs	QUIZUSE	QUIZ	An optional use file processed automatically by QUIZ before opening the system input file (unless the **auto** program parameter has been specified).
quizwork	.sf		QUIZ	The default name of the file containing the data portion of a subfile that has a minidictionary
quizwork	.sfd		QUIZ	The default name for the file containing the minidictionary portion of a subfile.
quizwork	.ps		QUIZ	The default name of the file containing the data portion of the portable subfile.

File	File Ext.[1]	File Equation/ Env. Variable/ Logical Name[2]	Component	Purpose
quizwork	.psd		QUIZ	The default name of the file containing the dictionary portion of the portable subfile.
qzmsgdes	.txt	QZMSGDES	QUIZ	An optional message file that, if present, contains designer messages to be used by QUIZ.
srvcmsgs	.msg	SRVCMSGS	All	Service layer messages file.
STDERROR (**MPE/iX**)		STDERROR	All	The error report file
STDIN (**MPE/iX**)		STDIN	All	The primary input file.
STDLIST (**MPE/iX**)		STDLIST	All	The primary output list file.
STDPRINT (**MPE/iX**)		STDPRINT	QUIZ QTP QSHOW	The print file used with SET PRINT.
SYSPRINT (**OpenVMS**)	.txt	SYSPRINT	QUIZ QTP PDL QSHOW	In QUIZ, an optional file that, if present, contains the source listing when SET PRINT is specified. In QTP, PDL, and QSHOW, the default name of the file created when SET PRINT is specified.
		PHSHOW	All except QUICK	**OpenVMS:** The symbol for the program file used by the components to run QSHOW.
		QSHOW	All except QUICK	**MPE/iX**: The program file used by the components to run QSHOW.
		quick	QDESIGN	**UNIX, Windows:** The environment variable used by QDESIGN to run QUICK
		QUICK	QDESIGN	**MPE/iX**: The program file used by QDESIGN to run QUICK. **OpenVMS:** The symbol used by QDESIGN to run QUICK.
ABDBA			All	**MPE/iX**: Allbase/SQL services executable library.
ORDBA			All	**OpenVMS:** ORACLE services executable library.

[1] Extensions apply only to **OpenVMS**, **UNIX**, and **Windows**.

[2] **MPE/iX** uses file equations, **OpenVMS** uses logical names, and **UNIX/Windows** use environment variables.

PDC Shared Dictionary (OpenVMS)

Introduction

PHD dictionaries are implemented as a set of five indexed files in the PHD file format. Reading these dictionaries is done through index access as needed. A PDC dictionary is implemented as a single stream file in common with the other platforms. Because of the format, the whole dictionary is read into memory when the dictionary is opened so that all of it is easily accessible. If your dictionary is very large and you have many users on the system, this can consume a great deal of physical memory.

On OpenVMS, screens are shared using global sections. Dictionary access is also available in this manner. This applies to products that open the dictionary with READ access only, that is, not PDL.

Requesting Dictionary Installations

The OpenVMS shared dictionary uses a combination of a PH_CREATE_SHARED logical and the QKGO setting for SCREEN SECTION in the execution time parameters. For QUICK, if a QKGO is being used (either through the AUTO=<qkgo file>, the QKGO logical or the PH_DEFAULT_QKGO logical), then the setting for SCREEN SECTION is used as the indicator to use a shared section of a particular type for the dictionary. If no QKGO is being used for QUICK and for all other products, the logical PH_CREATE_SHARED is evaluated. If PH_CREATE_SHARED is set, it indicates that a shared dictionary should be used, and what type of sharing is to be employed.

Executing SETPOWERHOUSE defines this logical with the "G"(Group) qualifier. If you want to change the type of sharing globally, or remove it, you must modify the file, PH_LOCATION:setph.com. You could also override the logical by setting it locally, after SETPOWERHOUSE has been executed.

The types of sharing are "P"(private), "G"(group), "S"(system). The default PowerHouse user privileges, as defined in the installation guide allow you to create/map Private and Group global sections. The SYSGBL privilege is required to create and map a System global section.

If the dictionary open/map fails, then you will get the specific OpenVMS error message that is returned by the system services being employed to do the job. This will be followed by one of the following messages:

QUICK will return the message

```
"No valid data dictionary was specified".
```

QUIZ, QTP and QSHOW will return the message

```
"*E* You have been denied access to the specified dictionary."
```

This error could mean one of the following situations exists:

- the dictionary file does not exist
- you do not have access to detect or read the dictionary file
- you do not have the privilege to create the global section (System primarily)
- you are using the PH_CREATE_SHARED logical and have assigned it an invalid value. Only "P","G", and "S" are valid.
- you don't have enough GBLPAGES or GBLSECTIONS or some other system resource to support the dictionaries being loaded into memory (see below for guidelines)

Shared Memory Configuration

The configuration of an OpenVMS system has two requirements for the use of shared memory. These are:

- GBLSECTIONS – one section per dictionary that is to be shared
- GBLPAGES – one page per file block + 2 for each dictionary to be shared. (for example, dictionary size is 1500 blocks – you need 1502 GBLPAGES).

See the *OpenVMS System Management Utilities Reference Manual* for more information about system tuning.

PHD Shared Dictionary (OpenVMS)

Each PowerHouse PHD dictionary on OpenVMS is contained in five RMS data files with the extension, PHD. You can make your dictionary shareable by enabling global buffers for these files. Sharing dictionaries is most useful during development of a PowerHouse application when the dictionary is frequently accessed. Enabling global buffers can also be useful in a runtime environment since some dictionary access is performed while executing compiled PowerHouse programs. Enabling global buffers for both PowerHouse dictionaries and RMS data files, and using screen global sections can improve application startup times and screen transition times. For more information, see "The Execution-Time Parameter Values Screen" in Chapter 6, "Customizing QUICK with QKGO" in the *QDESIGN Reference* book.

Sharing of PHD dictionaries can be enabled with the following set of DCL commands:

```
SET FILE/GLOBAL_BUFFER=n<PHD>00.PHD
SET FILE/GLOBAL_BUFFER=n<PHD>01.PHD
SET FILE/GLOBAL_BUFFER=n<PHD>02.PHD
SET FILE/GLOBAL_BUFFER=n<PHD>03.PHD
```

where <PHD> is the name of your dictionary and n is a number between 0 and 32,767.

Note: The value 0 disables global buffers.

Global buffers on OpenVMS are implemented as temporary global sections in non-paged dynamic memory. Your system manager may have to adjust the GLBLSECTIONS and NPAGEDYN parameters to use RMS global buffers. The exact amount of memory consumed by the dictionary will depend on the number of buffers allocated and the size of each buffer. The buffer size is the same as the file bucket size. For example, if a file has a bucket size of three (that is, three blocks of 512 bytes) then each buffer will be 1536 bytes. These buffers are shared by all system users that access the particular file(s). Without global buffers, each user would have local buffers buffering the same data and using overall memory. The objective of global buffers is to maintain shared data in memory where the first user accessing the data pays the cost of doing a physical disk I/O and other users accessing the same data only incur memory access. The net result is that less disk I/O will occur.

The number of global buffers to allocate for your dictionary or any other RMS data files used in your application will depend on many factors, including index depth of the file type of data access to the file (that is, sequential or random), amount of memory you want to reserve for global buffering the point at which you want OpenVMS and RMS to start re-using buffers.

We strongly urge you to read the "Guide to OpenVMS File Applications" in the OpenVMS documentation set in order to determine the optimum settings in your environment.

PDL Shared Dictionary (UNIX)

To reduce the possible overhead of having several users loading their own version of a dictionary into memory, PowerHouse allows a compiled dictionary to be installed into shared memory on UNIX systems. This method retains the efficient access that PowerHouse's PDL dictionary provides, while permitting the same dictionary to be shared by all of its users in memory and avoiding the load time for the dictionary for all but the very first user of that dictionary.

Installing Your Dictionary

PowerHouse is able to install dictionaries as shared sections all the time, only for selected dictionaries, or not at all.

Requesting Dictionary Installations

Any PowerHouse component will install a dictionary into shared memory when instructed to do so and when it is not already installed. This feature is controlled by an environment variable, PH_CREATE_SHARED. If PH_CREATE_SHARED is set, and if PowerHouse cannot locate a shared-memory section for a dictionary that it opens, then PowerHouse will automatically create a shared-memory section for that dictionary when it opens it.

There may be instances when PowerHouse determines that it should install a dictionary, but it is unable to do so. This may happen, for example, if there are insufficient shared memory IDs or memory remaining on your system, or if an existing shared memory section has the same key as that generated by PowerHouse. In these instances, PowerHouse will issue a warning message describing the problem but continue processing using a local version of the dictionary.

Selective Installation

If shared memory sections are tightly controlled at your installation, you may want to install only certain dictionaries into shared memory sections. This can be done by using the PH_CREATE_SHARED variable together with a PowerHouse component to install the desired dictionaries at system startup, but leaving the PH_CREATE_SHARED variable unset in user environments.

Here is a piece of Bourne shell script which installs two dictionaries with this method.

```
PH_CREATE_SHARED= /usr/cognos/ph703c/bin/qshow ENDINPUT
SET DICT /disk1/gen1/phd.pdc
SET DICT /disk2/othersys/sysdict.pdc
ENDINPUT
```

Automatic Installation

If your site has ample shared memory for its PowerHouse users, then it may be more suitable for PowerHouse dictionaries to be installed on an as-required basis. This can be done by setting the PH_CREATE_SHARED variable for all users of PowerHouse. The first user of any particular dictionary will then cause that dictionary to be moved into a shared memory section.

This can be done by a modification to the supplied setpow.csh and setpow.sh scripts, or by an addition to each user's .login or .profile files. The command for csh is the following:

```
setenv PH_CREATE_SHARED ""
```

For the Bourne shell, the commands are the following:

```
PH_CREATE_SHARED=""
export PH_CREATE_SHARED
```

Access to Installed Dictionaries

If PowerHouse is asked to open a dictionary that already has a corresponding shared-memory section, then it will use the shared memory version of the dictionary even if the PH_CREATE_SHARED environment variable is not set. This allows certain dictionaries to be installed into shared memory but not all dictionaries. PowerHouse will not create a second shared-memory section if one already exists for a given dictionary.

Access to Uninstalled Dictionaries

When PowerHouse accesses a dictionary which is not installed, and when the PH_CREATE_SHARED environment variable is not set (or if there are insufficient shared memory resources left to create a shared copy), PowerHouse will copy the dictionary into its own local memory for access. For small to moderate sized dictionaries, most users find this quite acceptable. If the dictionary is large or used by many users, then one of the shared memory options described above should be used to avoid loading the dictionary into local memory.

Deleting Shared Memory Sections

Normally, it should not be necessary to delete a PowerHouse dictionary from shared memory once it is created. However, there may be circumstances when it is necessary.

Note that it's perfectly safe to delete a shared memory section at any time. That is because a shared memory section will not actually be deleted as long as any process has the section mapped (it is turned into a private section until such time as no process is attached to it any more).

Following are three ways in which a PowerHouse shared dictionary can be de-installed.

Via PDL

PDL deletes shared memory sections as a normal course of doing a dictionary update. This avoids a shared memory section from becoming "stale" when a dictionary is updated. The recreated dictionary will need its own shared memory section, which can be created using the methods above. PDL will install the dictionary itself, provided that the PH_CREATE_SHARED environment variable is set.

Via ipcs and ipcrm

The UNIX ipcrm program can be used to remove any IPC resource, including a shared memory section. This can only be done via the creator or owner.

The ipcs program can be used to identify shared memory sections by key and owner (the actual pathname is not given). All PowerHouse shared memory sections are given an identifier of "C" which appears in the high order byte of the ipcs shared memory key field as a hex character 0x43. The owner and group of the section will be the same as that of the base dictionary (even if the section is created by another user). The permissions will permit read access to the same users as the original dictionary but no write or execute access. For example, if the file's permissions are -rw-r----- then the shared section will be -r--r-----.

Via a System Reboot

When a UNIX system is shut down, all shared memory sections are removed. Therefore, a system reboot is an effective way to clear out shared memory sections.

PowerHouse Consistency Checks

PowerHouse does two separate consistency checks when mapping to an installed PowerHouse dictionary: that the section is in fact an installed PowerHouse file; and that it matches the corresponding physical file.

Valid Section Check

PowerHouse checks any section it maps to in order to ensure that it is a valid PowerHouse compiled dictionary. It does this by looking for a known string within the first few bytes of the file. If this identifier isn't found, then PowerHouse assumes that some other system has created the shared memory section, and will bypass mapping to it. In this case, PowerHouse will use its default strategy of reading the dictionary into memory instead.

Out of Date Section

The second consistency check looks at a timestamp and other information within the compiled file and within the mapped section. If any of this information doesn't match, then PowerHouse assumes that the section is out of date. In this case, it will attempt to delete and recreate the compiled section. This can happen, for example, if you use cp to copy a dictionary over top of an existing dictionary.

PowerHouse will not be able to purge a compiled section if the user is not either the creator of the compiled section, the effective owner of the compiled section (which is also the owner of the dictionary file) or the root user. In this case, it will again revert to the default strategy of reading in the dictionary.

If you replace a dictionary of a running system which was installed, it's a good idea to ensure it's reinstalled by immediately running QSHOW, specifying the new dictionary. You should be running as either the creator user or the effective user of the shared memory section or as root. The effective user will be the same as the previous file owner; the creator user will be the user that caused the shared memory section to be created.

Shared Memory Management

Shared memory sections are a resource provided by the UNIX environment, and as such need some amount of configuration and management.

Shared Memory Basics

Shared memory is provided on most System V-based UNIX systems by means of the system calls shmget, shmctl, shmat and shmdt. A shared memory section is identified by a 32-bit key. This key is visible when shared memory sections are displayed with the ipcs command, and is used by PowerHouse to attach to shared memory sections.

Shared memory sections are virtual memory sections, and are paged in to real memory on an as-required basis (just like process memory). Therefore, memory dedicated to shared memory will require an equivalent amount of swap space, but will only require physical memory when pages of it are actually required in a running process.

PowerHouse generates a shared memory key to identify shared memory sections by means of the **ftok** system call. This provides an identifier based on physical disk file information (device ID and inode), as well as a one-byte project code. PowerHouse goes by the project code "C" (for Cognos). A specific dictionary will generate the same key even if it is accessed via a symbolic or hard link, or via a relative rather than an absolute path. However, once a dictionary is rebuilt via PDL, it will normally generate a different key even though the pathname didn't change.

Shared Memory Configuration

The configuration of a UNIX system specifies three limits upon the use of shared memory. Changing these values is described in the *UNIX System Administration* manual.

For PowerHouse, these parameters must allow sufficient space for the shared dictionaries that are to be installed, taking into consideration other users of the shared memory such as a database system. A given PowerHouse dictionary will require one shared memory identifier, and as many bytes of shared memory as the disk file occupies (rounded up to a page boundary of course).

The relevant parameters are the following.

SHMMAX - shared memory maximum

This parameter defines the maximum overall size of the shared memory area.

SHMMNI - shared memory maximum number of identifiers

This parameter defines the total number of shared memory identifiers (and hence the total number of sections) that can be created on the system.

SHMSEG - shared memory segments

This parameter defines the maximum number of segments that can be attached to any single process. Since PowerHouse Dictionary only attaches to a single segment at one time, this parameter should not need modification.

Mailbox Support in PowerHouse (OpenVMS)

A mailbox must be a sequential file of variable or fixed length and can be either permanent or temporary. The file type for a mailbox is MBX. Once a mailbox has been declared in the data dictionary, it can be used by any of the PowerHouse components.

Creating a Temporary or Permanent Mailbox

Permanent and temporary mailboxes can be created in the data dictionary (for example, on the PhD Record Screen) and can be created or deleted from the PhD File Maintenance Screen. When one of the PowerHouse components is called to open a mailbox that does not yet exist, that component can create a permanent or temporary mailbox. Note that a process that creates a permanent mailbox must have OpenVMS PRMMBX privilege.

The open name of a mailbox must be a logical name. PowerHouse creates a mailbox with the same MBAn, where n is a four-digit number. PowerHouse then equates the open name, that is, the logical name, with this name.

When a temporary mailbox is created, PowerHouse places the logical name in the Job Logical Name table.

When a permanent mailbox is created, the related logical name is placed in the System Logical Name table. Therefore, processes that create permanent mailboxes must also have OpenVMS SYSNAM privilege. To restrict mailbox access to a single group, use the following DEFINE statement to ensure that the logical name is placed in the group table. In this case, you must have OpenVMS GRPNAM privilege to create the mailbox:

```
$ DEFINE/TABLE=LNM$PROCESS_DIRECTORY -
       LNM$PERMANENT_MAILBOX LNM$GROUP
```

Temporary Mailbox Application

Mailboxes are typically used to store parameters that are passed from one process to another. For example, the designer can establish a QUICK screen that is used to prompt a user for data. Once prompted, the user enters the required data and QUICK writes it to the mailbox and calls QUIZ. QUIZ reads the mailbox when it is included in the ACCESS list, and produces a related report.

In the following example, the screen prompts for up to ten last names. These names are written to the mailbox EMPLOYEES_MBX. QUIZ is called and reports all employees from the employees file whose last names were entered in QUICK. The mailbox is declared in the data dictionary with the following attributes:

```
Record: EMPLOYEES_MBX
of File: EMPLOYEES_MBX
Organization: SEQUENTIAL
Type: MBX
Open: EMPLOYEES_MBX
Scope: Temporary
Record Format: Variable
Supersede: No
Record Size: 20 Bytes

Record Contents

Item   Type      Size Occ Offset
LASTNAMECHARACTER 200
```

The file is declared as a temporary mailbox with a file type of MBX, variable length record format, and a record-structure that includes only one item, LASTNAME.

The screen design created to write to the mailbox EMPLOYEES_MBX and then call QUIZ is

```
>   SCREEN EMPLOYEE_REP ACTIVITIES ENTRY AUTOUPDATE
>   FILE EMPLOYEES_MBX PRIMARY &
>      OCCURS 10 &
>      NOWAIT ON SEND &
>      SIGNAL ON CLOSE
>   FILE EMPLOYEES REFERENCE
```

```
>   TITLE "EMPLOYEE REPORT BY LAST NAME" AT 3,40 CENTERED
>   TITLE &
>      "Enter Last Name of Employees to be reported:" &
       AT 6,1
>   SKIP TO 8
>   ALIGN (,,10)
>   CLUSTER OCCURS WITH EMPLOYEES_MBX FOR 1,30
>   FIELD LASTNAME OF EMPLOYEES_MBX LOOKUP ON EMPLOYEES
>
>   PROCEDURE POSTUPDATE
>      BEGIN
>         RUN COMMAND "QUIZ AUTO=EMPLOYEE_REP.QZC"
>         CLEAR ALL
>      END
>   BULD
```

The mailbox EMPLOYEES_MBX is declared as the primary file with an occurrence of 10. Two options added to the FILE statement that you should note are NOWAIT ON SEND and SIGNAL ON CLOSE. NOWAIT ON SEND tells QUICK not to wait for another process to read the record that it has just sent to the mailbox. SIGNAL ON CLOSE tells QUICK to send an end-of-file marker to the mailbox when the mailbox is closed.

When the user returns to the Action field after entering all last names on the screen, the update is performed automatically, because the AUTOUPDATE option was added to the SCREEN statement. The POSTUPDATE procedure, which follows the UPDATE procedure, calls QUIZ.

The QUIZ report that was established looks like this:

```
>   ACCESS EMPLOYEES_MBX LINK TO EMPLOYEES
>   REPORT EMPLOYEE LAST NAME FIRSTNAME CITY
>   BUILD EMLOYEE_REP
```

This screen can be used by multiple users simultaneously since each user has a separate temporary mailbox.

Permanent Mailbox Application

Permanent mailboxes are typically used in an application where many processes pass data to a single process. The application we have created assumes that there are many data entry clerks entering invoices. A printed copy of every invoice must be generated. To accomplish this, write each invoice number to a permanent mailbox after the file and any other related files have been updated. A QUIZ process that runs continuously in batch can then read the mailbox, link to the related files, and produce a printed copy of the invoice.

The attributes for the permanent mailbox, INVOICE_MBX, should be declared in the data dictionary as follows:

```
Record: INVOICE_MBX
of file: INVOICE_MBX
Organization: SEQUENTIAL
Type: MBX
Open: INVOICE_MBX
Scope: Permanent
Record Format: Fixed
Supersede: No
Record Size: 2 Bytes
Record Contents:
ItemTypeSizeOccOffset
INVOICE-NOINTEGER20
```

INVOICE_MBX is declared as a permanent mailbox, with a file type of MBX, a fixed length record format and a record-structure that includes only one item, INVOICE-NO.

An example of part of a typical invoice data entry screen follows:

```
>   SCREEN ORDER
>   FILE INVOICES PRIAMRY
>   FILE CUSTOMERS SECONDARY
>   FILE INVOICE-LINES SECONDARY OCCURS 10
>      ITEM EXTENSION SUM INTO TOTAL-EXTENSION
>   FILE PRODUCTS REFERENCE OCCURS WITH TEH INVOICE-DETAIL
```

```
>   FILE INVOICE_MBX DESIGNER WAIT ON SEND
.
.
.
>   PROCEDURE UPDATE
>      BEGIN
>          PUT INVOICES
>          PUT CUSTOMERS
>          FOR INVOICE-DETAIL
>              BEGIN
>                  PUT INVOICE DETAIL
>                 END
>
>          LET INVOICE-NO OF INVOICE_MBX=INVOICE-NO &
>            OF INVOICES
>          PUT INVOICE_MBX
>              LET INVOICE-NO=100 &
>              ;required to force change in record status
>              LET INVOICE-NO=0
>              PUT INVOICE_MBX
>      END
>   BUILD
```

In this example, several statements are added to the screen to allow QUICK to access the mailbox. First, a FILE statement is added to declare the mailbox as a DESIGNER file. The WAIT ON SEND option causes QUICK to wait for another process to read the record after it has written that record to the mailbox. The UPDATE procedure is then used to transfer the invoice number to the mailbox record, and then to write the record to the mailbox. QUICK writes two records to the mailbox: one record has a real invoice number, the other record has an invoice number of 0.

The ACCESS statement in the QUIZ report is

```
>   ACCESS INVOICE_MBX TO INVOICES OPTIONAL &
>      LINK TO CUSTOMERS OPTIONAL &
>      LINK TO SHIP_TO OPTIONAL &
>      LINK TO INVOICE-DETAIL OPTIONAL &
>      LINK TO PRODUCTS OPTIONAL
>   SET FILE INVOICE_MBX WAIT ON RECIEVE
.
.
.
>   SORTED ON INVOICE-NO RESET PAGE
```

The mailbox is specified in the ACCESS statement and is linked to the INVOICES file. The WAIT ON RECIEVE option is added to the SET FILE statement. This option indicates that QUIZ will wait for records to be written to the mailbox, if the mailbox is empty. The SORTED statement allows each record to be processed immediately. If the SORT statement is used instead, records are read and placed in a temporary file until an end-of-file marker is read from the mailbox. Only then are the records sorted and the report sent to the printer.

QUIZ reads and processes the real record after it reads the dummy record (invoice number 0). The next time a record is written to the mailbox, the dummy records is processed; as a result, the linkage in the ACCESS statement fails and no data is read. All numeric fields that are reported are created with the BWZ option; character fields are blank. Therefore, a blank page is generated by the dummy record. To avoid blank pages, simply add the SET NOBLANKS statement to the QUIZ program.

Mailboxes can also be used in QUIZ to perform batch updates. To achieve the same results in QTP as in QUIZ, add a condition to the OUTPUT or SET SUBFILE statement. For example,

```
>   OUTPUT filename UPDATE IF RECORD INVOICES EXISTS
```

To perform this application, the QUIZ report must be submitted in batch every morning, for example, using the following DCL statements:

```
$   @SETPOWERHOUSE
$   SETDICT DEMO
$   QUIZ AUTO=INVOICE.QZC
```

After this is submitted, the QUIZ report runs continuously until an end-of-file marker is sent to the mailbox. Therefore, every evening, a second QUIZ screen should be called to close the mailbox:

```
>   SCREEN CLOSE
>   FILE INVOICE_MBX SIGNAL ON CLOSE
>   FIELD INOICE-NO
>   BUILD
```

Enter F;^ on this screen to close the mailbox and terminate the QUIZ program after you exit the screen. This process can be automated by placing F;^ in a QKIN file and running the following command file:

```
$   DEFINE QKIN USERCOMMANDS.TXT
$   QUICK AUTO=CLOSE.QKC
```

In this way, you can use timed DCL command procedures to fully automate the daily creation and termination of the QUIZ batch program.

Using Mailboxes to Pass Source Statements

You can use mailboxes to pass source statements from one component to another. For example, the following QUICK screen was designed to include procedural code that writes QUIZ source code to a mailbox:

```
>   LET MBX_REC = "ACCESS FILENAME"
>   PUT MBX_FILE
>   LET MBX_REC = "REPORT ALL"
>   PUT MBX_FILE
>   LET MBX_REC = "GO"
>   PUT MBX_FILE
>   LET MBX_REC = "EXIT"
>   PUT MBX_FILE
>   RUN COMMAND "QUIZ AUTO=MBX_FILE"
```

Permanent mailboxes can also be used by a QUIZ batch job running continuously with AUTO=PERM_MBX_FILE. In this case, you would design a QUICK screen that sends requests to the QUIZ batch job using the following statements:

```
>   LET PER_MBX_REC = "EXE REPORT1"
>   PUT PER_MBX_FILE
```

Here, you do not send the EXIT statement to the mailbox until you terminate the QUIZ batch job. The advantage of using a permanent mailbox for this application is that many users can run QUIZ reports in batch, but only one batch job is required to run all the reports.

Mailboxes and System Crashes

Application managers should exercise caution when using mailboxes with sensitive data. A system crash will cause all data in the mailbox to be lost.

sitehook (OpenVMS)

The sitehook feature provides a callable interface to user-supplied entry and exit routines. These two routines must be called PH_SITEHOOK_INIT and PH_SITEHOOK_EXIT and must be linked as a shared image. Since a shared image is used, universal symbols are also required.

For VAX, transfer vectors that can be used to accomplish this are contained in this file:

PH_DEMO_LOCATION:SITEHOOK_VECTORS.MAR

For Alpha, the option file for linking vectors can be found in

PH_DEMO_LOCATION:SITEHOOK.OPT

The steps involved in producing and accessing this shared image are:

1. Create an object file of the users routines by compiling the user's source file(s)
2. For VAX, produce the shared image by linking the user's object file with the provided transfer vector object file. For example,

   ```
   $LINK/SHARE=<image> PH_DEMO_LOCATION:SITEHOOK_VECTORS, <users objects>
   ```

   For Alpha, produce the shared image by linking the user's object file with the provided option file. For example,

   ```
   $LINK/SHARE=<image> PH_DEMO_LOCATION:SITEHOOK/OPT, <users objects>
   ```

If none of the images are installed with privileges, then you must define a logical name for PH_SITEHOOK. For example,

```
$DEFINE PH_SITEHOOK <image>
```

where <image> is the file specification for the shared image name.

If any of the PowerHouse images are installed with privileges, then the shared image must be installed and either the image must reside in SYS$SHARE or a System Executive mode logical name (use the complete physical file specification) must be defined for the image. This is an OpenVMS requirement. If the image is PH_LOCATION:SITEHOOK, then define PH_SITEHOOK as follows:

```
$SET PROCESS/PRIVILEGE=SYSNAM
$DEFINE/SYSTEM/EXECUTIVE_MODE PH_SITEHOOK
- DISK$COGNOS:[COGNOS.POWERHSE.830c.PH_COMMON]SITEHOOK.EXE
```

The /EXECUTIVE_MODE is required. The shared image does not need to be installed with privileges. However, if any PowerHouse component has been installed with system privileges, then the PH_SITEHOOK executable must be installed with OPEN, HEADER and SHARE.

Once this is set up, the user supplied entry routine is called first whenever any of the components are invoked and the user defined exit routine is called whenever a component is exited.

To ensure that these routines are always called, install all images with privileges; use an innocuous privilege such as TMPMBX. Once installed with privileges, users can't circumvent the routines by creating their own version and defining a process logical name to run them.

Examples are provided in:

Machine	Location
VAX	PH_DEMO_LOCATION:SITEHOOK_EXAMPLE.MAR.
Alpha or VAX	PH_DEMO_LOCATION:SITEHOOK_EXAMPLE.C.

The command file PH_DEMO_LOCATION:SITEHOOK_LINK(_ALPHA).COM links this example together and defines the necessary logical name.

Large File Support (UNIX, Windows)

Direct and sequential files, non-indexed subfiles, and portable subfiles can exceed the two gigabyte limit in total number of bytes. A file built with a limit below two gigabytes must be recreated to be able to grow beyond two gigabytes. The file system must be able handle large files and must be configured to allow them.

PowerHouse 4GL can still only process up to 2,147,483,647 records.

DISAM Data Storage (Windows)

DISAM does not support unsigned integer types in index segments. As well, many of the PowerHouse date datatypes are based on unsigned integers. Since PowerHouse allows you to declare any datatype as an index segment, the solution is to map the unsigned integer and date datatypes to a character datatype. There is an additional complication on Windows in that the native storage format for integer data is little endian while the character datatype storage format is effectively big endian.

For consistency of data storage and retrieval, PowerHouse will convert integer data to big endian when it is saved to a DISAM file. It will be converted to little endian format when it is read into the PowerHouse buffers so that internally it is consistent with data read from other files.

About Little Endian and Big Endian

Little and big endian describe the sequence in which digits are stored in integer datatypes. In little endian format, the most significant digit is stored on the right. In big endian format, the most significant digit is stored on the left.

For example, if a two-byte integer contains the number 16,730, on a little endian machine, the hex value is 5A41, whereas, on a big endian machine the hex value is 415A. Note that it is the byte sequence that is different not the bits within the bytes. Since the machine code knows which sequence the bytes are in, the value is correct.

For non-integer data such as character or float, there is no difference to how the bytes are stored on the two types of machine.

If the two-byte integer is mapped to a character data type, it is always assumed to be effectively big endian for sequencing. If a two-byte integer, stored as little endian, is mapped to a character data type and sorted, the sort sequence will be incorrect. In order to get the correct sequence, the integer must be converted to big endian before use.

Datatype Mappings

When used as index segments, the mappings from PowerHouse datatypes to DISAM datatypes are the same as identified in the "Datatype Mapping Tables" section of *PDL and Utilities Reference*. Also note:

- PowerHouse will convert the data to big endian for storage when mapping to CHARTYPE occurs.
- CHARTYPE won't handle negative numbers correctly.
- FREEFORM keys and segments do not work due to the nature of the FREEFORM datatype.
- PACKED and ZONED only work if the signs are all the same.

Reading and Writing Data

Using these mappings and the dictionary structure, data is converted as it is read and written. The data is stored in the file in big endian format and resides in the PowerHouse buffers in little endian format.

Data can be moved to and from non-DISAM and DISAM files because the conversion is done as the DISAM files are read from and written to. Because of the way the data is stored, PowerHouse cannot read third-party DISAM files that are in "native" or little endian format.

Retrieving Data

Because the data is stored in big endian format, ranged retrieval works because integer index segments are converted prior to passing the data to DISAM. The conversion includes any datatypes that are stored in PowerHouse buffers as signed or unsigned integer, even if they are mapped to character.

For example, consider this ACCESS statement in QDESIGN.

```
> ACCESS VIA int-1, fulldate-1 &
>   USING exp-for-int-1, exp-for-fulldate-1
```

If INT-1 is an integer, the result of the expression exp-for-int-1 is converted from little endian to big endian before it is passed to DISAM for retrieval. If FULLDATE-1 is a century-included date, it is stored as a 4-byte unsigned integer but is mapped to a character. Because it is stored as an integer, the value must be converted from little endian to big endian.

Substructures are not converted on retrieval. Only what is specified or named in the retrieval syntax is converted. This includes default retrieval as well as developer-specified retrieval. In most cases, for proper retrieval specify the lowest level segments in your retrieval. DISAM supports multi-segment indexes so compound indexes that make use of substructures as opposed to multiple segments should be changed to use multiple segments. Multiple substructures and redefinitions should not be used in indexes.

Redefinitions and Substructures

When converting data on retrieval or storage, data is converted at the lowest substructure level of the last redefinition. If there's only one substructure, that is the one converted.

As mentioned previously, when data is passed for retrieval, conversion is based on the segments specified in the retrieval. Any substructures of the named segment are not converted.

Caution must be taken if using redefinitions or substructures involving integers. Specific areas to consider are:

- Some file systems do not support multi-segment indexes, such as KSAM and IMAGE on MPE/iX. It is not uncommon to use a substructured index as a workaround. When migrating these file systems to DISAM, substructured index segments should be redesigned to use multi-segment indexes. This also requires that explicit retrieval be modified. Retrieval must be done using the lowest level segments.
- Assigning data from one item to another should always be done at the lowest level to ensure that all needed conversion is done.
- While multiple redefinitions and substructures are rare, any being used must be reviewed to ensure that the correct data will be used. For example, conversion may not be correct if integers are overlayed by integers because PowerHouse only converts the lowest level of the last redefinition. Keep this in mind if different redefinitions have different levels of detail.

Chapter 2: Program Parameters

Overview

This chapter describes the program parameters you can use to control various attributes of PowerHouse.

About Program Parameters

Program parameters control attributes such as determining which dictionary PowerHouse runs. The program parameters for each PowerHouse component are temporary, lasting only as long as the session of the component.

Unlike most PowerHouse keywords, program parameters cannot be abbreviated except where noted.

The syntax for entering a PowerHouse program parameter is:

component [program parameter]...

For example,

```
qdesign auto=filespec list
```

Summary of Program Parameters

The following table is a summary of each program parameter.

Program Parameter	Description
auto	Establishes the file, screen, or report that is processed when the component is initiated.
autodetach|noautodetach	Automatically detaches database connections.
blockmode (**MPE/iX**)	Runs a blockmode terminal in either Panel or Block mode.
broadcast (**OpenVMS**)	Determines how QUICK handles broadcast, or non-PowerHouse messages.
bulkfetch	Specifies the number of rows a bulkfetch returns.
cc	Sets the conditional compile flags you can use in the statements of each PowerHouse component to tell it whether to process or skip blocks of code.
charmode	Specifies whether or not QUICK recognizes BLOCK TRANSFER control structures in Character mode.
checksum710 (**OpenVMS**)	Allows PowerHouse 8.30 and later versions to use the 7.10 form (unsigned) in the CHECKSUM calculations.
close_detach	Indicates that CLOSE verbs encountered in QUICK will cause a physical database detach.

Program Parameter	Description
columnowner	Determines how the item names in a cursor will be generated by PowerHouse Services.
commitpoints	Enables the default commit timing for COMMIT ON UPDATE used for versions prior to 7.2x.
compress_buffers	Causes the initialization pool in QDESIGN to be compressed before it is stored.
confirmer	Indicates that the user must confirm messages before processing can continue.
consolekeys|noconsole keys (**Windows**)	Instructs QUICK to display function keys.
createall	Creates all non-relational files declared in the dictionary, excluding only those files that are declared to be NOCREATE.
createbase (**MPE/iX**)	Creates the named IMAGE database, even if it is specified to be NOCREATE in the dictionary.
createfile	Creates the named file, even if it is specified to be NOCREATE in the dictionary.
cursorowner	Causes QDESIGN to use the owner name in the default query and in generated FIELD statements for the query.
dbaudit	Enables database monitoring for PowerHouse components.
dbdetach|nodbdetach	Releases or does not release the database connection when you return to the screen prompt.
dbwait|nodbwait	Specifies whether or not the program waits until a concurrency conflict is resolved.
dcl|nodcl (**OpenVMS**)	Stipulates whether or not operating system commands can be entered or executed when PowerHouse encounters an operating system prompt.
debug (QDESIGN)	To use the QUICK Debugger, you must use the debug program parameter in QDESIGN and compile the screens to be debugged.
debug (QUICK)	Controls the level of debugging capabilities.
deleteall	Deletes all non-relational files and IMAGE databases declared in the dictionary, excluding only those files that are declared to be NOCREATE, and those databases whose datasets are all declared to be NOCREATE.
deletebase (**MPE/iX**)	Deletes the named IMAGE database.
deletefile	Deletes the named file, even if it is specified to be NOCREATE in the dictionary.
designer_noretain	Causes a commit to end the transaction.
detail|nodetail	Specifies whether or not to copy the contents of a file into the pdlsave file.
dictionary|dict	Establishes the data dictionary that the PowerHouse component opens when it starts.

Program Parameter	Description
dicttypeldt (**OpenVMS**)	Specifies what dictionary type will be used.
direct_file_base_zero (**OpenVMS**)	Allows the use of zero-based record numbers for cross-platform compatibility.
disable_nulls	Controls whether null support is allowed at the item level. It overrides the dictionary setting.
dont_store_module	Prevents SQL modules from being compiled and stored in the database and compiled screen/run/report. This program parameter is used at parse time.
downshift|upshift|noshift	Specifies that the values of identifiers be shifted to lowercase, uppercase, or left as entered.
entryrecall	Specifies that data from previous screens is or is not available for recall in Entry mode.
errlist	Redirects standard error/list to the specified file.
fastread (**OpenVMS**)	Performs a block read to sequentially accessed read-only files.
fdl|nofdl (**OpenVMS**)	Specifies whether to create or delete File Definition Language (FDL) files.
initnulls|noinitnulls	Specifies whether to initialize columns to NULL in rows not retrieved.
intsize6|nointsize6 (**OpenVMS**)	Specifies whether to create INITEGR SIZE 6 datatypes.
jcwbase (**MPE/iX**)	Specifies the base value for JCW settings.
lineread (**MPE/iX**)	Specifies that QUICK uses multi-character reads.
list|nolist	Establishes whether or not the PowerHouse component displays the source statement file.
lockword (**MPE/iX**)	Enables the use of internal routines to prevent duplicate lockword prompting during screen loading and running.
moduleext (**MPE/iX**)	Causes the module names that are stored with compiled sections in an ALLBASE/SQL database to be modified.
moduleloc (**MPE/iX**)	Compiles a PowerHouse program, creating an installable module which can be copied to a second environment in ALLBASE/SQL.
nls (**MPE/iX, UNIX**)	Ensures that no line of text for the printer is split between two output blocks.
noblobs	Specifies whether blobs columns are processed.
nobreakset (**MPE/iX**)	Improves the performance of KSAM and KSAMXL reads. If specified, the system break around each read is not disabled.
nonportable	Sets the severity of warning messages when nonportable syntax is encountered.
nontermcompat (**Windows**)	Suppresses messages caused by design errors in the layout portion of the compiled screen file.

Program Parameter	Description
noowner	Prevents the owner name from being attached to the table name in generated code so that different users can use the same compiled screen to access their own tables (having the same name).
noprefix_openname	Suppresses the addition of "ORACLE@" to the open name for an ORACLE database.
nosetwarnstatus (**OpenVMS**)	Suppresses the warning status when a warning condition is detected in PowerHouse.
nouicbrackets (**OpenVMS**)	Causes the UIC function to not return brackets around the result values. It also causes internal security checking to match the existence of brackets based on the setting of the program parameter.
nxl	Suppresses the printing on an extra blank line at the end of reports.
obsolete	Sets the severity of the messages when obsolete syntax is encountered.
omnidex|noomnidex (**MPE/iX**)	Determines whether or not OMNIDEX indexes are seen by the PowerHouse component being run.
osaccess|noosaccess	Specifies whether or not access to the operating system from within PowerHouse components is allowed.
owner	Specifies the owner for tables in an ALLBASE/SQL, DB2, SYBASE, or ORACLE database when none is explicitly indicated. Also specifies the default owner of modules created by PowerHouse in ALLBASE/SQL.
parmfile (**OpenVMS, UNIX, Windows**)	Tells PowerHouse that the PARM values should come from the specified file instead of prompting the user.
parmprompt	Strips or does not strip trailing blanks from PARM values.
patch	Issues a number of informational messages about patches that have been applied to the PowerHouse component.
pollspeed (**MPE/iX**)	Sets the amount of time (n) that QUICK takes when polling the terminal to determine a terminal type before prompting the user.
pre_chooseall	Affects QUIZ reports and QTP runs in batch mode that use the CHOOSE statement with a PARM option and have no PARM values specified.
procloc	Causes PowerHouse to search for process files in a location other than the current directory (**OpenVMS, UNIX, Windows**) or group/account (**MPE/iX**).
prompt	Specifies the prompt for the PowerHouse component.
qktrace	Generates a log file of QUICK screen processing that gives a summary of activities.
quotedproccall	Allows quoted stored procedure calls to be passed directly to the database.
read (**MPE/iX**)	Determines how QUICK uses single-character reads.
resetbindvar|noresetbindvar	Determines if SQL bind variables are reset for each SQL statement.
resource	Specifies the PowerHouse resource file.

Program Parameter	Description
restore	Provides upward compatibility for screens compiled with versions of PowerHouse prior to 6.09 (**MPE/iX**), 6.00 (**OpenVMS**), and 6.03 (**UNIX**). The restore program parameter changes the default behavior of screen refreshing.
retainmark|noretainmark	Determines whether the field mark position in an array is retained.
reuse_screen_buffers|noreuse_screen_buffers	Causes QUICK to reuse or not reuse previously allocated buffers.
search	Causes QUIZ to search the ACCESS statement in a first-to-last or last-to-first sequence.
secured	Restricts the files and items listed by the SHOW FILES statement to those for which you have at least read access.
setjobshow|nosetjobshow	Restricts the files and items listed by the SHOW FILES statement to those for which you have at least read access.
statistics|nostatistics	Specifies whether or not to display statistics.
subdictionary|subdict	Specifies whether or not subdictionary support is enabled; when the relational subdictionaries are opened by PowerHouse; and when they are searched for unqualified record-structures.
subformat	Specifies the format of subfiles being created.
term	States the terminal type used and the maximum number of lines the terminal memory contains.
termpoll|notermpoll (**MPE/iX, OpenVMS**)	Specifies whether QUICK attempts to determine the terminal type before prompting the operator.
timezone|notimezone (**MPE/iX**)	Determines the mechanism that PowerHouse uses to obtain the values of SYSDATE and SYSTIME.
tpi|notpi (**MPE/iX, HP-UX, Windows**)	Determines whether TPI or OMNIDEX indexes are seen by the PowerHouse component being run.
trusted|notrusted (**OpenVMS**)	Activates or deactivates C2-level security for the execution of RUN commands and DCL commands within components.
update	Controls the way PUT verbs are generated in the UPDATE procedure.
version	Provides the build number of the PowerHouse version.
vmsdate (**OpenVMS**)	Changes how PowerHouse creates and processes VMSDATE items.

The following table lists the program parameters that are applicable to each PowerHouse component.

Program Parameter	PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
auto	✔	✔	✔	✔	✔	✔	✔	✔
autodetach|noautodetach						✔		
blockmode (**MPE/iX**)						✔		
broadcast (**OpenVMS**)						✔		

Program Parameter	PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
bulkfetch					✔	✔	✔	
cc	✔	✔	✔	✔	✔	✔	✔	✔
charmode						✔		
checksum710 (**OpenVMS**)		✔	✔	✔	✔	✔	✔	✔
close_detach						✔		
columnowner			✔		✔		✔	
commitpoints			✔					
compress_buffers			✔					
confirmer						✔		
consolekeys|noconsolekeys (**Windows**)						✔		
createall								✔
createbase (**MPE/iX**)								✔
createfile								✔
cursorowner			✔					
dbaudit			✔	✔	✔	✔	✔	
dbdetach|nodbdetach						✔		
dbwait|nodbwait			✔	✔	✔	✔	✔	
dcl|nodcl (**OpenVMS**)	✔	✔	✔	✔	✔	✔	✔	✔
debug (QDESIGN)			✔					
debug (QUICK)						✔		
deleteall								✔
deletebase (**MPE/iX**)								✔
deletefile								✔
designer_noretain			✔					
detail|nodetail	✔							
dictionary|dict	✔	✔	✔	✔	✔	✔	✔	✔
dicttype|dt (**OpenVMS**)			✔	✔	✔	✔	✔	✔
direct_file_base_zero (**OpenVMS**)			✔	✔	✔	✔	✔	
disable_nulls			✔					

Program Parameter	PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
dont_store_module			✔		✔		✔	
downshift|upshift|noshift	✔		✔	✔	✔	✔	✔	✔
entryrecall						✔		
errlist	✔	✔	✔	✔	✔	✔	✔	✔
fastread (**OpenVMS**)					✔	✔	✔	
fdl|nofdl (**OpenVMS**)								✔
initnulls|noinitnulls					✔			
intsize6|nointsize6 (**OpenVMS**)	✔	✔						
jcwbase (**MPE/iX**)							✔	
lineread (**MPE/iX**)						✔		
list|nolist	✔	✔	✔	✔	✔		✔	✔
lockword (**MPE/iX**)			✔			✔		
moduleext (**MPE/iX**)					✔	✔	✔	
moduleloc (**MPE/iX**)			✔		✔	✔	✔	
nls (**MPE/iX, UNIX**)							✔	
noblobs			✔		✔		✔	
nobreakset (**MPE/iX**)	✔		✔	✔	✔	✔	✔	✔
nonportable	✔	✔	✔	✔	✔		✔	✔
nontermcompat (**Windows**)			✔					
noowner	✔	✔	✔	✔	✔	✔	✔	✔
noprefix_openname	✔	✔	✔	✔	✔	✔	✔	
nosetwarnstatus (**OpenVMS**)	✔	✔	✔	✔	✔	✔	✔	✔
nouicbrackets (**OpenVMS**)	✔	✔	✔	✔	✔	✔	✔	
nxl							✔	
obsolete		✔	✔	✔	✔		✔	✔
omnidex|noomnidex (**MPE/iX**)			✔	✔	✔	✔	✔	✔
osaccess|noosaccess	✔	✔	✔	✔	✔	✔	✔	✔
owner			✔	✔	✔	✔	✔	
parmfile (**OpenVMS, UNIX, Windows**)					✔		✔	

Program Parameter	PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
parmprompt					✔		✔	
patch	✔	✔	✔	✔	✔	✔	✔	✔
pollspeed (**MPE/iX**)						✔		
pre_chooseall					✔		✔	
procloc	✔	✔	✔	✔	✔	✔	✔	✔
prompt	✔	✔	✔	✔	✔	✔	✔	✔
qktrace					✔			
quotedproccall			✔		✔		✔	
read (**MPE/iX**)						✔		
resetbindvar|noresetbindvar			✔		✔	✔	✔	
resource	✔	✔	✔	✔	✔	✔	✔	✔
restore						✔		
retainmark|noretainmark						✔		
reuse_screen_buffers| noreuse_screen_buffers						✔		
search							✔	
secured				✔				
setjobshow|nosetjobshow					✔		✔	
statistics|nostatistics					✔		✔	✔
subdictionary|subdict			✔	✔	✔	✔	✔	
subformat					✔		✔	
term			✔			✔		
termpoll|notermpoll (**MPE/iX, OpenVMS**)						✔		
timezone|notimezone (**MPE/iX**)	✔		✔	✔	✔	✔	✔	✔
tpi|notpi (**MPE/iX, HP-UX, Windows**)			✔		✔	✔	✔	
trusted|notrusted (**OpenVMS**)	✔	✔	✔	✔	✔	✔	✔	✔
update			✔					
version	✔	✔	✔	✔	✔	✔	✔	✔
vmsdate (**OpenVMS**)	✔	✔	✔	✔	✔	✔	✔	✔

auto

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Establishes the file, screen, or report that is processed when the component is initiated.

Syntax

auto=filespec

filespec

The specification for a file as it is identified to the operating system. It takes the general form:

MPE/iX:	**[*]filename[/lockword][.group[.account]]**
OpenVMS:	**[node::][device:][[directory]] filename[.extension][;<version>]** Square brackets are required around a directory name.
UNIX:	**/[directory/]...filename.extension**
Windows:	**[drive:\][directory\]...filename.extension**

Discussion

In QUICK and Debugger, **auto** establishes the screen or QKGO parameter file to be processed as soon as QUICK is initiated.

For the other PowerHouse components, **auto** establishes the source statement or compiled report file processed when the component is initiated.

If you use **auto** but the PowerHouse component can't find the file or screen, an error message is issued and the component terminates.

If **auto** is not specified, the PowerHouse component looks for a designated file. If the designated file is not found the component doesn't issue an error message. Instead you will be prompted to enter statements.

The designated files are as follows:

Component	MPE/iX	OpenVMS	UNIX, Windows
PDL	PDLUSE	PDLUSE.PDL	pdluse.pdl
PHDPDL		PHDPDLUSE.PDL	
QDESIGN	QKUSE	QKUSE.QKS	qkuse.qks
QSHOW	QSHOUSE	QSHOUSE.QSS	qshouse.qss
QTP	QTPUSE	QTPUSE.QTS	qtpuse.qts
QUICK		QKGO.QKG	qkgo.qkg
QUIZ	QUIZUSE	QUIZUSE.QZS	quizuse.qzs
QUTIL		QUTLUSE.QUS	qutluse.qus

For more information about locating files, see (p. 41).

autodetach|noautodetach

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Automatically detaches database connections.

Syntax

autodetach|noautodetach

Default: **autodetach**

Limit: Applies only to Sybase.

Discussion

The **autodetach** program parameter applies only to relational databases that support a single transaction per database attach. It specifies that QUICK automatically detaches the database connections if the transactions are committed or rolled back, and are not locally active when the user exits the screen. This minimizes the number of attaches for those single transaction databases. Currently, Sybase is the only supported relational database that fits this category.

PowerHouse versions prior to 8.4E did not detach automatically causing a growth in the number of attaches over time. The **noautodetach** program parameter is provided to allow the pre-8.4E behavior to be specified.

Equivalent Resource File Statement

AUTODETACH ON|OFF

blockmode (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Runs a blockmode terminal in either Panel or Block mode.

Syntax

blockmode=compatible|panel

Default: **compatible**

Limit: Only valid for blockmode terminals.

compatible

Causes a blockmode terminal to run in standard HP Block mode.

panel

Causes the blockmode terminal to run in Panel mode.

Equivalent Resource File Statement

TERMINAL BLOCKMODE COMPATIBLE|PANEL

broadcast (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Determines how QUICK handles broadcast, or non-PowerHouse messages.

Syntax

broadcast=default|deferred

Default: **deferred**

default

QUICK bases its treatment of non-PowerHouse messages on the specification established by the DCL SET BROADCAST command.

deferred

Non-PowerHouse messages are trapped and displayed on the message line when QUICK performs the next I/O to the terminal.

Equivalent Resource File Statement

BROADCAST DEFAULT|DEFERRED

bulkfetch

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔	✔	✔	

Specifies the number of rows a bulkfetch returns.

Syntax

bulkfetch=n

n

The number of rows to return.

Default: 0

Limit: 32767

Discussion

The **bulkfetch** program parameter allows you to specify the number of rows a retrieval returns and, therefore, the amount of memory allocated to the retrieval buffer. When retrieving rows from relational databases, PowerHouse often fetches more than one row at a time to improve performance. Changing the bulkfetch value may help performance depending on the retrieval situation. The **bulkfetch** program parameter has no effect on non-relational retrieval.

The amount of memory allocated is approximately the size of the rows times the number of rows to be retrieved. There is a trade off between the memory allocated and the performance improvement. Allocating too much memory impacts performance adversely in a multi-user environment. The default internal value (also available by setting bulkfetch=0) is conservative and very low. Trial and error is the best way to determine the optimal improvement in specific environments. A value of 512 is a good starting point.

The **bulkfetch** program parameter only helps in one-to-many or many-to-many linkages. It does not help in one-to-one or many-to-one linkages, whether unique or not, since multiple rows must be returned for bulkfetch to have any effect.

Equivalent Resource File Statement

BULKFETCH n

cc

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Sets the conditional compile flags you can use in the statements of each PowerHouse component to tell it whether to process or skip blocks of code.

Syntax

cc=(name[,name]...)

name

A unique name identifying a conditional compile flag. The names are not special keywords in themselves, but they are referenced by conditional compile constructs.

Limit: A name can be a maximum of 127 characters.

Discussion

The **cc** program parameter sets conditional compile flags you can use in PowerHouse statements to tell PowerHouse whether to process or skip blocks of code. PowerHouse adds an underscore in the listings to indicate skipped code, as in

```
> SCREEN PROJECT
> @IF UNIX
> DEFINE END_DATE DATETIME = SYSDATETIME
> @ELSEIF OPENVMS
>_DEFINE END_DATE VMSDATE = VMSTIMESTAMP
>_@ELSE
>_DEFINE END_DATE DATE = SYSDATE
>_@ENDIF
>
```

This program parameter is only effective at compile time; that is, it cannot change compiled files.

In QUICK, the **cc** program parameter can only be used in conjunction with the **debug** program parameter.

For more information about how to use compile-time flags and a list of predefined flags, see (p. 282).

Equivalent Resource File Statement

CC name[,name]...

Example

In the following example, the names "TEST" and "YEAREND" determine different report specifications.

```
>@IF TEST
>  SET REPORT LIMIT 50
>@ELSEIF YEAREND
>  SET REPORT LIMIT 50000
>  SET REPORT DEVICE PRINTER
>@ELSE
>  SET REPORT LIMIT 10000
>@ENDIF
```

QUIZ can be run with a **cc** flag of "TEST", or "YEAREND". If no **cc** flag is specified, the report limit of 10000 will be used.

charmode

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Specifies whether or not QUICK recognizes BLOCK TRANSFER control structures in Character mode.

Syntax

charmode=field|panel

Default: **panel**

field

Indicates that BLOCK TRANSFER control structures in a screen are ignored.

panel

Indicates that BLOCK TRANSFER control structures in a screen are recognized.

Discussion

If the parameter is not specified and BLOCK TRANSFER control structures exist in screens, the default interface mode is panel.

Equivalent Resource File Statement

TERMINAL CHARACTERMODE FIELD|PANEL

checksum710 (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
	✔	✔	✔	✔	✔	✔	✔

This program parameter only applies to QDESIGN, QSHOW and QUTIL when it is used with PHD dictionaries that are checksum710 calculated.

Allows PowerHouse 8.30 and later versions to use the 7.10 form (unsigned) in the CHECKSUM calculations.

Syntax

checksum710 [=onloff]

ON means PowerHouse uses unsigned input for checksum calculations. OFF means PowerHouse uses signed input.

Specifying checksum710 without an option is the same as specifying checksum710 = on.

Default: ON (version 7.10); OFF (versions 8.xx).

Discussion

CHECKSUM710 Backwards Compatibility Switch

The CHECKSUM function in 7.10 produces a different result than in 8.xx. While the algorithm used is the same, the internal input into that calculation is Unsigned in 7.10 but Signed in 8.xx. This produces different results from the same calculation.

This is only an issue if you store the results of the CHECKSUM function in your data files, check that value on processing the data, and share or migrate these data files between 7.10 and 8.30 and above.

There are two ways to deal with this issue. If you are migrating your data and application, and do not need to share with 7.10, then you may want to consider just recalculating the CHECKSUM values in your files and move upwards. This is the recommended route in this case.

If you need to share data, we have endeavored to provide backward compatibility that allows you to do so. However, if you use it, then you must use the 7.10 form for all applications using the same data in PowerHouse 8.30 and above. If, in the future, you drop the 7.10 requirements, you may recalculate the checksums at that point and drop the compatibility mode.

To allow PowerHouse 8.30 and later versions to use the 7.10 form (unsigned) in the CHECKSUM calculations, you need to use the CHECKSUM710 logical, program parameter or resource file statement. If possible, we recommend that you use the logical so that all PowerHouse components automatically use this setting. Depending on your environment and needs, this can be set at a process, group, or system level. You could, alternatively, use a resource file defined at any of these levels, or a program parameter on each execution.

The logical syntax is $DEF CHECKSUM710 "ON"|"OFF" (here the options are not optional as a null string is not a valid logical definition).

The precedence rules are as follows: a program parameter overrides a resource file statement which overrides a logical name setting.

Using CHECKSUM in PHD Dictionaries

If you are using PHD dictionaries, you need to recalculate the checksums in your dictionaries. The same procedure can be used to return the checksums to 8.xx values at a later date, if desired.

CHECKSUM.COM is found in PHD_LOCATION:. It has two parameters. The first parameter is your dictionary name (it modifies the dictionary directly). The second parameter is either 7 or 8, to indicate the form used when recalculating the dictionary checksums. Seven uses CHECKSUM710=ON, and 8 uses CHECKSUM=OFF. For example:

```
@PHD_LOCATION:CHECKSUM <userdict> 7
```

This example would recalculate the checksums in a user dictionary from 8.xx to 7.10-like values so that you can use the CHECKSUM710 program parameter, resource file statement, or logical when using these dictionaries in PowerHouse.

Also, there is an additional parameter to PHDMAINT, PHDADMIN, and the POW and PHDCONV commands. The parameter is CHECKSUM710(=ON|OFF). For all these command procedures, if the logical CHECKSUM710 is set to the option you require, the parameter does not need to be used.

When switching between PHD dictionaries with one setting or the other, you must reset the dictionary before starting up the application or you will get a "Corrupted dictionary" error when the product tries to open the dictionary. Any time you use the wrong option when trying to access a dictionary, you get the "Corrupted dictionary" error.

Equivalent Resource File Statement

CHECKSUM710 [ON|OFF]

close_detach

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Indicates that CLOSE verbs encountered in QUICK will cause a physical database detach.

Syntax

close_detach

Discussion

When the **close_detach** program parameter is used, CLOSE verbs cause an immediate physical database detach. When the program parameter is not used, detaches are only done upon exit of the screen where the attach was done.

In addition, when the program parameter is used, Oracle open names are not prefixed with "ORACLE@". Without the prefix, users can specify a logical name which could be set to different values, and thus point to different databases.

Equivalent Resource File Statement

CLOSE DETACH

columnowner

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔		✔	

Determines how the item names in a cursor will be generated by PowerHouse Services.

Syntax

columnowner

Discussion

Correlation names are qualified metadata references to column names appearing in PowerHouse applications as ITEM or FIELD names.

PowerHouse 8.4x's underlying database access software attempts to conform more strictly to the SQL92 standard, which describes column correlation names as table_name.column_name. The owner name is no longer included before the table_name.

In previous versions of PowerHouse, correlation names sometimes included the owner name and sometimes did not, depending on the specification of the SQL statement. Below, is a table showing sample SQL SELECT statements and indicating the resultant column correlation names:

	SQL Statements	Correlation Name
a)	SELECT COLUMN FROM TABLE	TABLE.COLUMN
b)	SELECT COLUMN FROM OWNER.TABLE	OWNER.TABLE.COLUMN
c)	SELECT TABLE.COLUMN FROM TABLE	TABLE.COLUMN
d)	SELECT TABLE.COLUMN FROM OWNER.TABLE	TABLE.COLUMN
e)	SELECT OWNER.TABLE.COLUMN FROM TABLE	TABLE.COLUMN
f)	SELECT OWNER.TABLE.COLUMN FROM OWNER.TABLE	OWNER.TABLE.COLUMN

For applications being upgraded to 8.4x, the **columnowner** program parameter enables the successful parsing of column names permitted in earlier versions. If **columnowner** is specified, the owner name is obtained from other metadata sources for the column and prefixed to the correlation name.

This allows applications coded prior to PowerHouse 8.4x to compile and execute without changing all column names that appear in the old format.

For applications with cursors defined in the form of examples (b) and (f), the **columnowner** program parameter may be used.

If an application has multiple cursors defined in mixed forms, for example, one cursor similar to (b) and another similar to (d), it may be necessary to make manual changes to the PowerHouse syntax since the program parameter won't distinguish between the different formats and will always add the owner name.

For PowerHouse syntax being created with the 8.4x releases, all column references should omit the owner name.

Equivalent Resource File Statement

COLUMNOWNER

commitpoints

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Enables the default commit timing for COMMIT ON UPDATE used for versions prior to 7.2x.

Syntax

commitpoints=obsolete

Discussion

All screens that require the pre-7.2x commit timing must be compiled using this program parameter.

Equivalent Resource File Statement

COMMITPOINTS OBSOLETE

compress_buffers

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Causes the initialization pool in QDESIGN to be compressed before it is stored.

Syntax

compress_buffers

Discussion

Using this program parameter means that the physical size of screens will be decreased if the data is compressible. However, it does impose a certain overhead on the reading of screens since this data must be uncompressed before it can be used.

Equivalent Resource File Statement

COMPRESS BUFFERS ON|OFF

confirmer

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Indicates that the user must confirm messages before processing can continue.

Syntax

confirmer

Discussion

The user cancels or accepts messages by selecting Cancel or OK in the Confirmer Window.

Equivalent Resource File Statement

TERMINAL CONFIRMER

consolekeys|noconsolekeys (Windows)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Instructs QUICK to display function keys.

Syntax

consolekeys|noconsolekeys

Default: noconsolekeys

consolekeys

Instructs QUICK to display function keys at the bottom of the Command Console window.

noconsolekeys

Instructs QUICK to not display function keys.

Discussion

The **consolekeys** program parameter instructs QUICK to display eight function keys across the bottom of the Command Console window under the QUICK screen. These labels are not clickable using a mouse and only represent the function keys and labels.

Neither the **consolekeys** program parameter nor the CONSOLE KEYS resource file statement has any effect on displaying function keys in QKView. To display function keys in QKView, select the Function Keys entry in the View menu.

Equivalent Resource File Statement

CONSOLE KEYS ON|OFF

createall

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
							✔

Creates all non-relational files declared in the dictionary, excluding only those files that are declared to be NOCREATE.

Syntax

createall

Limit: Not available for relational files.

Discussion

QUTIL terminates after the creation completed.

createbase (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
							✔

Creates the named IMAGE database, even if it is specified to be NOCREATE in the dictionary.

Syntax

createbase=file

Discussion

QUTIL terminates after the creation completed.

createfile

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
							✔

Creates the named file, even if it is specified to be NOCREATE in the dictionary.

Syntax

createfile=file

file

A file declared in the data dictionary.

Limit: Not available for relational files.

Discussion

QUTIL terminates after the creation completes.

cursorowner

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Causes QDESIGN to use the owner name in the default query and in generated FIELD statements for the query.

Syntax

cursorowner

Discussion

If you want to access a table which does not belong to you, you must specify the owner name. You can do this in two ways:

- by directly coding the owner name in SQL queries and FIELD statements
- by using the **owner** and **cursorowner** program parameters to include the owner name in generated SQL queries and FIELD statements.

When QDESIGN generates a default SQL query from a CURSOR statement, it does not use the owner name defined in the dictionary or in the **owner** program parameter. The **cursorowner** program parameter causes QDESIGN to use the current owner name in:

- the generated default SQL query.
- generated FIELD statements for the fields in the CURSOR statement.

When **cursorowner** is specified and FIELD statements are coded, you can only use the following syntax:

```
FIELD CURSOR_COLUMN OF CURSOR_STRUCTURE
```

or

```
FIELD OWNERNAME.CURSOR_STRUCTURE.CURSOR_COLUMN OF CURSOR_STRUCTURE
```

If the ownername changes, any screens compiled with the **cursorowner** program parameter must be recompiled.

Equivalent Resource File Statement

DEFAULT CURSOR OWNER

Examples

If QDESIGN is run as follows:

```
QDESIGN AUTO=SCREENX OWNER=USER1
```

and the program SCREENX contains:

```
CURSOR EMPLOYEE IN TEMPEST73
```

The default SQL query generated from the CURSOR statement is:

```
SELECT * FROM EMPLOYEE
```

If QDESIGN is run with the **cursorowner** program parameter:

```
QDESIGN AUTO=SCREENX OWNER=USER1 CURSOROWNER
```

the generated SQL query will be:

```
SELECT * FROM USER1.EMPLOYEE
```

When **cursorowner** is specified, FIELD statements generated by QDESIGN include the owner name for each relational field, as in:

```
FIELD OWNERNAME.CURSOR_STRUCTURE.CURSOR_COLUMN OF CURSOR_STRUCTURE NULL VALUE
NOT ALLOWED
```

In the following example, user1 wants to access the table, SKILL, which is owned by user2. User1 runs QDESIGN as follows:

```
QDESIGN AUTO=SCREENX OWNER=USER2 CURSOROWNER
```

The program contains:

```
CURSOR SKILL IN TEMPEST73
```

The generated SQL query will be:

```
SELECT * FROM USER2.SKILL
```

dbaudit

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	

Enables database monitoring for PowerHouse components.

Syntax

dbaudit=brief|file|full|msgs

brief

Displays a short (one line) information line as database operations occur.

file

Writes detailed information to a file called dbaudit.txt located, by default, in the current working directory.

full

Displays detailed information as database operations occur.

msgs

Displays Binary Language Representation (BLR) messages in hexadecimal.

Discussion

dbaudit=brief

When the **dbaudit=brief** program parameter is specified, the following lines describe the output produced:

```
ATTACH db_handle TO db_type db_name
COMPILE REQUEST request_handle
START LOGICAL TRANSACTION trans_name details

PREPARE LOGICAL TRANSACTION trans_name
COMMIT LOGICAL TRANSACTION trans_name
ROLLBACK LOGICAL TRANSACTION trans_name
START TRANSACTION trans_handle IN dbhandle_list

START REQUEST request_handle IN TRANSACTION trans_handle
RELEASE REQUEST request_handle FROM TRANSACTION trans_handle
PREPARE TRANSACTION trans_handle
COMMIT TRANSACTION trans_handle
ROLLBACK TRANSACTION trans_handle
DETACH db_handle FROM db_name
```

db_handle

A unique numeric value that identifies a database.

db_type

Either "ALLBASE", "ORACLE", "RDB", "DB2", "ODBC", or "SYBASE."

db_name

The database name.

request_handle

A unique numeric value that identifies a request.

trans_name

The transaction name.

details

May contain a combination of the following:

- active, all active, locally active
- read only|read write
- wait|nowait
- read uncommitted|cursor stability|repeatable read|phantom protection|serializable
- reserving reserve_name_comma_list (A list of the relation names specified for this transaction).

dbhandle_list

A comma-separated list of dbhandle.

dbaudit=full

When **dbaudit=full** is specified, the detailed information displays the value of DBKEY for every attach that is done to one or more databases.

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

Equivalent Resource File Statement

DBAUDIT BRIEF|FILE|FULL|MESSAGES|MSGS|NONE

dbdetach|nodbdetach

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Releases or does not release the database connection when you return to the screen prompt.

Syntax

dbdetach|nodbdetach

Default: **nodbdetach**

Discussion

When you leave a QUICK screen and go back to the screen ID prompt after processing through a relational database connection, QUICK can either detach from the database or keep the connection. If you keep the connection (not detaching), memory is still allocated for the connection. This means that when you call another screen, QUICK allocates more memory for the new database connection(s). This causes memory growth in the product.

The default is **nodbdetach**. While this uses more memory, there may be small performance benefits. Note that this only affects screens called from the screen ID prompt, which typically is not used in production environments.

Equivalent Resource File Statement

DBDETACH ON|OFF

dbwait|nodbwait

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	

Specifies whether or not the program waits until a concurrency conflict is resolved.

Syntax

dbwait|nodbwait

Default (except for QUICK): **nodbwait**

Default for QUICK: **dbwait**

Limit: Applies only to ALLBASE/SQL, ORACLE.

Discussion

The **dbwait** program parameter specifies that if a concurrency conflict occurs during access to a relational database, the program normally waits until the conflict is resolved. An example of concurrency conflict is attempting to write a record that has been locked by another user. If the **nodbwait** parameter is specified and the database encounters a concurrency conflict, an error message results.

Equivalent Resource File Statement

DBWAIT ON|OFF

dcl|nodcl (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Stipulates whether or not operating system commands can be entered or executed when PowerHouse encounters an operating system prompt.

Syntax

dcl|nodcl

Default (except for QUICK): **dcl**

Default for QUICK: **nodcl**

Discussion

If **nodcl** is selected, the informational message

```
Access to the operating system has been disabled
```

is issued when you enter the system prompt. When you call a PowerHouse component directly from another, that component is invoked with either **dcl** or **nodcl** in effect, whichever parameter is set in the calling component.

For more information about allowing and preventing operating system access, see (p. 138).

Equivalent Resource File Statement

OSACCESS ON|OFF

debug (QDESIGN)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

To use the QUICK Debugger, you must use the **debug** program parameter in QDESIGN and compile the screens to be debugged.

Syntax

debug

Discussion

For more information, see Chapter 10, "Debugger Commands", in the *QDESIGN Reference* book.

Equivalent Resource File Statement

DEBUG WARN

debug (QUICK)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Controls the level of debugging capabilities.

Syntax

debug=error|nowarn|source|warn

error

Returns an error message for screens compiled for Debugger.

nowarn

Runs screens compiled for Debugger but does not allow access to Debugger.

source

Runs compiled screens and allows full access to Debugger.

warn

Displays a warning message when a screen is encountered that was compiled with Debugger enabled.

Discussion

To debug your QUICK screens you must have compiled them using the **debug** program parameter in QDESIGN.

For more information, see Chapter 10, "Debugger Commands", in the *QDESIGN Reference* book.

Equivalent Resource File Statement

DEBUG ERROR|NOWARNING|SOURCE|WARNING

deleteall

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
							✔

Deletes all non-relational files and IMAGE databases declared in the dictionary, excluding only those files that are declared to be NOCREATE, and those databases whose datasets are all declared to be NOCREATE.

Syntax

deleteall

Limit: Not available for relational files.

Discussion

QUTIL terminates after the deletion completes.

deletebase (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
							✔

Deletes the named IMAGE database.

Syntax

deletebase=filespec

Discussion

QUTIL terminates after the deletion completes.

deletefile

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
							✔

Deletes the named file, even if it is specified to be NOCREATE in the dictionary.

Syntax

deletefile=file
Limit: Not available for relational files.

file

A file declared in the data dictionary.

Discussion

QUTIL terminates after the deletion completes.

designer_noretain

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Causes a commit to end the transaction.

Syntax

designer_noretain

Discussion

By default, DESIGNER files use the commit retain functionality, which means the transaction is kept open after a commit. By using this program parameter, you can cause a commit to end the transaction.

Equivalent Resource File Statement

DESIGNER NORETAIN

detail|nodetail

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔							

Specifies whether or not to copy the contents of a file into the pdlsave file.

Syntax

detail|nodetail

Default: **detail**

Discussion

When you specify the **detail|nodetail** program parameter and enter a REVISE statement, the **detail** program parameter copies the contents of the revised file into the pdlsave file when you exit from the system editor, the **nodetail** program parameter does not.

When you specify the **detail|nodetail** program parameter and enter a USE statement, the **detail** program parameter writes the contents of the file, rather than just the USE statement itself, to the pdlsave file. The **nodetail** program parameter writes just the USE statement, rather than the contents of the file, to PDL's source statement save file, pdlsave.

The program parameter is overridden when you specify either the **detail** or **nodetail** option on the SET, USE, or REVISE statement during a PDL session.

dictionary|dict

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Establishes the data dictionary that the PowerHouse component opens when it starts.

Syntax

dictionary=filespec

dict=filespec

filespec

The specification for a file, as it is identified to the operating system. A file specification takes the general form:

MPE/iX:	**[*]filename[/lockword][.group[.account]]**
OpenVMS:	**[node::][device:][[directory]] filename[.extension][;<version>]** Square brackets are required around a directory name.
UNIX:	**/[directory/]...filename.extension**
Windows:	**[drive:\][directory\]...filename.extension**

OpenVMS: For PHD type dictionaries, extension and version are not valid options.

Discussion

The **dictionary** program parameter establishes the specified dictionary for the duration of a session of a PowerHouse component but does not override any dictionary previously set. The program parameter can be abbreviated to **dict**.

You can also use the SET DICTIONARY statement or SETDICT command (**OpenVMS, UNIX, Windows**) to set a specified dictionary.

OpenVMS:	The default dictionary type for all components except PDL is PHD. If no PHD dictionary is found, the components look for a dictionary with the .pdc extension.
UNIX, Windows:	The default extension when specifying a dictionary is .pdc.

Equivalent Resource File Statement

DICTIONARY filespec

dicttype|dt (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	✔

Specifies what dictionary type will be used.

Syntax

dicttype=pdc|phd
dt=pdc|phd

Default: **phd**

Discussion

Specifies the default dictionary type to be used during the session of the PowerHouse component. When the **dicttype** program parameter is used, it applies to all SET DICTIONARY statements where the TYPE option is not specified. A dictionary type specified on the SET DICTIONARY statement will override the **dicttype** setting. If a type is not specified, PowerHouse searches first for a PHD dictionary, then for a PDC dictionary.

If an extension is included on the **dictionary** program parameter and a conflicting **dicttype** is specified, you will get an error.

Equivalent Resource File Statement

DICTIONARY filespec TYPE PHD|PDL

direct_file_base_zero (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	

Allows the use of zero-based record numbers for cross-platform compatibility.

Syntax

direct_file_base_zero

Discussion

In PowerHouse versions 8.00 and 8.10C, direct file access was zero-based from an external view to make PowerHouse code more portable between platforms.

In 8.10.C1 and subsequent versions, direct file access is one-based from an external view. This matches the physical implementation. On OpenVMS the record numbers start at 1, not 0.

To maintain upgrade paths, and to provide cross-platform transparency, the direct_file_base_zero program parameter is available to allow for the use of zero-based record numbers if required. Externally, the PowerHouse user can use zero-based numbers. PowerHouse adds one to the external value to make it one-based for the internal file system

Note: From 8.10.C1 through to 8.20.D4, the one-based access applied to READs, but not to WRITEs. This was incorrect and has been changed as of 8.20D6 and 8.30 so that it applies to both READ and WRITE.

Equivalent Resource File Statement

RMS FILE BASE ZERO|ONE

disable_nulls

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Controls whether null support is allowed at the item level. It overrides the dictionary setting.

Syntax

disable_nulls

Equivalent Resource File Statement

DISABLE NULLS

dont_store_module

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔		✔	

Prevents SQL modules from being compiled and stored in the database and compiled screen/run/report. This program parameter is used at parse time.

Syntax

dont_store_module

Discussion

This program parameter can decrease memory problems associated with compiled sections since the SQL modules will be forced to compile at run-time. However, this may have an impact on performance.

Equivalent Resource File Statement

STORE MODULES ON|OFF

downshift|upshift|noshift

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔		✔	✔	✔	✔	✔	✔

Specifies that the values of identifiers be shifted to lowercase, uppercase, or left as entered.

Syntax

downshift|upshift|noshift

Default: **upshift**

Discussion

By default, PowerHouse upshifts all components of table names. PowerHouse permits access to case-sensitive names by means of this program parameter or the NOSHIFT, UPSHIFT, and DOWNSHIFT options of the SET statement. If **noshift** or SET NOSHIFT is specified, all PowerHouse identifiers are taken as they appear in the source text instead of being upshifted. For system-wide access to mixed, lowercase, or uppercase identifiers, you can specify the SHIFT option in the SYSTEM OPTION statement.

Equivalent Resource File Statement

SHIFT DOWN|NONE|UP

entryrecall

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Specifies that data from previous screens is or is not available for recall in Entry mode.

Syntax

entryrecall

Discussion

Specifying **entryrecall** means that QUICK users can recall the previous record's values in Entry mode. Values are only recalled and displayed if requested using the Recall command (the Up Arrow, Ctrl-B, or whatever key has been set).

Users can change the displayed value before it is processed by QUICK. The cursor is positioned immediately to the right of the recalled value as if the user had typed it into the field. The positioning is to the left if the REVERSE option of the FIELD statement is specified. Error recall, and the recall of data in change processing, is not affected by **entryrecall**.

Note: You can also duplicate the previous record's values using the Duplicate command (by default, the underscore), but you cannot change the duplicated value before it is processed by QUICK.

Equivalent Resource File Statement

ENTRY RECALL

errlist

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Redirects standard error/list to the specified file.

Syntax

errlist=filename

Discussion

Errlist is used to redirect standard error and standard list to the specified file.

fastread (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔	✔	✔	

Performs a block read to sequentially accessed read-only files.

Syntax

fastread

Discussion

By using this program parameter, normal record I/O is much faster. However, it has a number of restrictions:

- The records must be fixed length.
- The access must be sequential, not indexed.
- The data and indexed portions of the file cannot be compressed. By default, PowerHouse creates files with compression turned on, so compression must be turned off manually using the FDL editor and the file recreated in order to access a PowerHouse file with this program parameter. If these conditions are not met, you do not receive a warning but PowerHouse does not set up the file for fast reads.

Equivalent Resource File Statement

RMS FAST READ ON|OFF

fdl|nofdl (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
							✔

Specifies whether to create or delete File Definition Language (FDL) files.

Syntax

fdl|nofdl

Default: **nofdl**

Discussion

FDL files have the form <filename>.fdl. If **fdl** is specified,

- the CREATE statement creates the FDL file as well as the data file
- the DELETE statement deletes the FDL file as well as the data file

To override this program parameter, use SET FDL or SET NOFDL within the QUTIL session.

initnulls|noinitnulls

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔			

Specifies whether to initialize columns to NULL in rows not retrieved.

Syntax

initnulls|noinitnulls

Defaults: **noinitnulls**

Discussion

Columns in rows not retrieved should be initialized to NULL if null values are allowed. This is what happens in QUIZ. In QTP, columns are initialized to spaces, zeroes, and dictionary initial values. The **initnulls** program parameter can be used to tell QTP to properly initialize such columns to NULL. The default is **noinitnulls** to remain consistent with the operation of previous versions of QTP.

Equivalent Resource File Statement

INITIALIZE NULLS ON|OFF

intsize6|nointsize6 (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔						

Specifies whether to create INITEGR SIZE 6 datatypes.

Syntax

intsize6|nointsize6

Defaults: **intsize6** for PDL; **nointsize6** for PHDPDL.

Discussion

When using PDL in PowerHouse 8.xx, integers of physical SIZE 6 are created for numeric elements with 10-14 digits with INTEGER datatype and no SIZE specified. In 7.10 and PHDPDL, these elements will default to SIZE 8. In a mixed PowerHouse version environment, or when using datafiles created under one version or dictionary type to be used by another, this will cause an incompatibility between dictionaries and physical datafiles. The physical record lengths will not match.

There are two methods to correct this problem. You can either

- specify SIZE for such items, thereby fixing the physical size to match the files, or
- use the **intsize6|nointsize6** program parameters to control how the products work

For PDL, **nointsize6** will cause the item sizes to not create SIZE 6 integers, thus matching PHDPDL and 7.10 created files. For PHDPDL, **intsize6** will cause integer SIZE 6 items to be created, thus matching files created in PDL and 8.xx versions.

Equivalent Resource File Statement

INTEGER SIZE 6 ON|OFF

jcwbase (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
						✔	

Specifies the base value for JCW settings.

Syntax

jcwbase=fatal|warn

Discussion

The **jcwbase** program parameter specifies the base value for QUIZ JCW settings. For more information on JCW settings, see "QUIZ Error Status Settings (MPE/iX, UNIX, Windows)" (p. 22). If the base value is fatal, an error can cause a job to stop.

Equivalent Resource File Statement

JCWBASE FATAL|WARN

lineread (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Specifies that QUICK uses multi-character reads.

Syntax

lineread

Discussion

The **linread** program parameter causes QUICK to run the application without the use of single character processing. It is the old form of the **read=line** program parameter.

list|nolist

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔		✔	✔

Establishes whether or not the PowerHouse component displays the source statement file.

Syntax

list|nolist

Limit: These parameters apply to source statement files only.

Default: **list**

Discussion

The **list** program parameter states that the contents of source statement files are to be listed as they are read; **nolist** suppresses the listing of the source file statements as they are read.

The **list|nolist** program parameter establishes the default list option for the USE and REVISE statements without LIST or NOLIST options. Within the PowerHouse component, an entry of SET DEFAULT or SET LIST overrides **nolist**. The LIST option of the USE statement temporarily overrides **nolist**. Likewise, an entry of SET NOLIST resets the LIST control from that point on. The NOLIST option of the USE statement temporarily overrides **list**.

Equivalent Resource File Statement

LIST USE ON|OFF

lockword (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔			✔		

Enables the use of internal routines to prevent duplicate lockword prompting during screen loading and running.

Syntax

lockword

Discussion

By default, if a compiled screen has a a file level password, LOCKWORD, the user is prompted for this lockword every time the screen is run. If the screen has to be loaded and then run, the user is prompted for the lockword twice.

The **lockword** program parameter enables internal routines to prevent duplicate prompting. However, use of the program parameter may cause a performance decrease so it should be tested to determine the effect.

Equivalent Resource File Statement

LOCKWORD PROMPT ONCE

moduleext (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔	✔	✔	

Causes the module names that are stored with compiled sections in an ALLBASE/SQL database to be modified.

Syntax

moduleext=extension

Discussion

The module name is based on the name of the file in which the compiled QUICK screen, QUIZ report, or QTP run is stored.

If the fully qualified name of the compiled file is

```
file.group.account
```

then the default module name is

```
file_group
```

If the **moduleext** program parameter is specified, the module name is:

```
file_extension
```

Equivalent Resource File Statement

ALLBASE MODULE EXTENSION string

moduleloc (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔	✔	✔	

Compiles a PowerHouse program, creating an installable module which can be copied to a second environment in ALLBASE/SQL.

Syntax

moduleloc=filelocation

filelocation

The specification for a group.

Limit: The location specified by **moduleloc** cannot be the same as the location of the file named in the BUILD or RUN statements.

Equivalent Resource File Statement

LOCATION [MODULE filelocation]

nls (no line split) (MPE/iX, UNIX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
						✔	

Ensures that no line of text for the printer is split between two output blocks.

Syntax

nls

Discussion

On some terminals in the HPSLAVE output block method, when a line of output to the printer is split between two output blocks, the portion of the line in the first block is overwritten by the portion of the line in the second block. **nls** prevents this from happening.

Equivalent Resource File Statement

HPSLAVE SPLIT LINES ON|OFF

noblobs

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔		✔	

Specifies whether blobs columns are processed.

Syntax

noblobs

Discussion

By default, QUIZ, QTP, and QDESIGN refer to blob columns. However, blobs are restricted in use; blobs cannot be stored in subfiles, sorted on, or written into an intermediate file in QTP. All of these actions produce an error. By specifying the **noblobs** program parameter, blob columns are not processed (ignored) and these errors are avoided.

Equivalent Resource File Statement

NOBLOBS

nobreakset (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔		✔	✔	✔	✔	✔	✔

Improves the performance of KSAM and KSAMXL reads. If specified, the system break around each read is not disabled.

Syntax

nobreakset

Discussion

This program parameter should only be used if the products are being run from a NOBREAK UDC or command file.

nonportable

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔		✔	✔

Sets the severity of warning messages when nonportable syntax is encountered.

Syntax

nonportable=error|nowarn|warn

Default: **nowarn**

Limit: Applies to source statement files only.

error

Rejects nonportable syntax and issues an error message.

nowarn

Suppresses the warning messages.

warn

Issues a warning, though processing continues.

Discussion

Not all PowerHouse syntax applies to all computer systems.

Equivalent Resource File Statement

NONPORTABLE ERROR|NOWARNING|WARNING

nontermcompat (Windows)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Suppresses messages caused by design errors in the layout portion of the compiled screen file.

Syntax

nontermcompat

Discussion

QDESIGN has many default settings that affect how pages are generated. QDESIGN generates both PowerHouse Web pages and character terminal screens used by the QUICK component of PowerHouse 4GL. You may see a default terminal layout when you enter the BUILD statement. To disable the terminal layout display enter

```
> SET NOLIST LAYOUT
```

QDESIGN may issue warnings and errors if layout objects are specified outside of the terminal screen area. Even though PowerHouse Web ignores specific rows and columns, you can avoid these errors by specifying rows and columns within the character terminal's 24 by 80 area. On Windows, you can instruct QDESIGN to ignore these errors by using the **nontermcompat** program parameter.

Using the nontermcompat program parameter in Axiant

QUICK on Windows requires the layout information in the compiled screen file, but Axiant does not.

The **nontermcompat** program parameter and the Terminal Compatible property in Axiant provide a mechanism to maintain the same level of functionality of Axiant and still maintain application compatibility with QUICK on Windows.

The **nontermcompat** program parameter is an internal parameter in Axiant. A Terminal Compatible property has been added to the Axiant Build Profile. When Terminal Compatible is set to TRUE, the program parameter is not used and QDESIGN will compile the screen with full terminal layout error checking. This is the default.

If an Axiant application is a thin client connecting to a Windows server, then the default value of TRUE may result in terminal layout compilation errors. If this occurs then you must manually select FALSE to suppress the errors. This may be necessary when upgrading such an Axiant application to version 3.4.

If the Terminal Compatible is set to FALSE, then the program parameter is used and some terminal layout error messages will be suppressed. When an error message has been suppressed, the compiled screen file will be marked with a layout error. Should QUICK try to execute such a screen, it will issue a screen design error and the screen will not run. If the screen is compiled with the setting set to FALSE and no terminal layout errors are suppressed, then QUICK will be able to execute the screen.

The relationship between the Axiant Build Profile setting and the program parameter is as follows:

Terminal Compatible Setting	Action
True	The program parameter is not used.
False	The program parameter is used.

noowner

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Prevents the owner name from being attached to the table name in generated code so that different users can use the same compiled screen to access their own tables (having the same name).

Syntax

noowner

Equivalent Resource File Statement

NOOWNER

noprefix_ownername

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	

Suppresses the addition of "ORACLE@" to the open name for an ORACLE database.

Syntax

noprefix_ownername

Discussion

Valid open names must begin with "ORACLE@". If the open name supplied does not begin with this string, PowerHouse normally inserts it. This program parameter suppresses the insertion thus allowing logical names (**OpenVMS**) and environment variables (**UNIX, Windows**) to used as the open name. If **noprefix_ownername** is used, the user must supply the "ORACLE@" in the string assigned.

Equivalent Resource File Statement

PREFIX ORACLE OPEN NAME ON|OFF

nosetwarnstatus (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Suppresses the warning status when a warning condition is detected in PowerHouse.

Syntax

nosetwarnstatus

Discussion

In versions prior to 8.40, a warning condition in PowerHouse returned a $WARNING level indication to the operating system. When Module Management System (MMS) sees a result code that is not $SUCCESS, processing stops.

If this is not the desired behavior, specify the **nosetwarnstatus** program parameter to turn off the warning status. If the **nosetwarnstatus** program parameter is used, a status code of $SUCCESS is returned to operating system for PowerHouse warning conditions, instead of $WARNING.

The default behavior is that $WARNING status is used.

Equivalent Resource File Statement

NOSET WARN STATUS

nouicbrackets (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Causes the UIC function to not return brackets around the result values. It also causes internal security checking to match the existence of brackets based on the setting of the program parameter.

Syntax

nouicbrackets

Discussion

This program parameter is provided for compatibility with the UNIX and Windows **uic** function, and previous versions of PowerHouse 8.xx.

Equivalent Resource File Statement

UIC BRACKETS ON|OFF

nxl (no extra line)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
						✔	

Suppresses the printing on an extra blank line at the end of reports.

Syntax

nxl

Equivalent Resource File Statement

HPSLAVE EXTRA LINE ON|OFF

obsolete

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
	✔	✔	✔	✔		✔	✔

Sets the severity of the messages when obsolete syntax is encountered.

Syntax

obsolete=error|ignore|nowarn|warn

Default: **warn**

Limit: Applies to source statement files only.

error

Rejects obsolete syntax when encountered and issues an error message.

ignore

Ignores all obsolete keywords as part of the syntax (they are treated as entity names).

nowarn

Suppresses the warning messages.

warn

Issues a warning message, though processing continues.

Discussion

As PowerHouse matures, some syntax may be marked for obsolescence and may not be supported in future releases.

Equivalent Resource File Statement

OBSOLETE ERROR|IGNORE|NOWARNING|WARNING

omnidex|noomnidex (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	✔

Determines whether or not OMNIDEX indexes are seen by the PowerHouse component being run.

Syntax

omnidex|noomnidex

Default: **omnidex**

noomnidex

OMNIDEX indexes are not seen by the product being run.

omnidex

OMNIDEX indexes are seen by the product being run. This is provided to allow people to override a resource file setting of QUIZ or OFF.

Equivalent Resource File Statement

OMNIDEX ON|OFF|QUIZ

Discussion

The program parameter or resource file statement is only required at parse time.

In PowerHouse versions from 7.29 to 8.29, OMNIDEX indexes could be declared in the dictionary, but were only used by QUIZ (and reported by QSHOW). QTP, QDESIGN and QUICK had no knowledge of these indexes. DISC, the developers of OMNIDEX, provided tools to access OMNIDEX indexes from these versions of QUICK.

In version 8.39, OMNIDEX index support was added to QTP, QDESIGN, and QUICK. This changed some default operations. Because these indexes are now visible, they are used in the default linking rules. This can result in different default linkages. Linkages that were specifically coded remain unchanged. The addition of this support to QUICK prevents the DISC tools from functioning.

Version	Default use of OMNIDEX indexes
7.29 to 8.29	QUIZ
8.39 and above	QUIZ, QTP, QDESIGN, QUICK

With the appropriate program parameter or resource file statement, you can determine how you want OMNIDEX indexes to be used.

osaccess|noosaccess

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Specifies whether or not access to the operating system from within PowerHouse components is allowed.

Syntax

osaccess|noosaccess

Default (except for QUICK): **osaccess**

The **osaccess program** parameter allows access to the operating system from within PowerHouse components; **noosaccess** prevents access.

Where the parameter is set to **noosaccess**, any command preceded by a system prompt character is ignored.

Discussion

By default, when QUICK encounters the system prompt character, it ignores it and issues the message

```
"Operating system has been disabled".
```

When you call QUICK directly from QDESIGN, QUICK is invoked with either **osaccess** or **noosaccess** in effect, depending on which parameter is set in QDESIGN.

MPE/iX

The system prompt character is a colon (:). Entering a colon, followed by a system command causes execution of that command by the operating system.

OpenVMS

The system prompt character is a dollar sign ($). Entering a dollar sign followed by a system command causes execution of that command by the operating system.

Note: See also **dcl|nodcl** on (p. 100).

UNIX

The system prompt character is an exclamation mark (!). Entering !<shell_abbreviation> (for example, !csh) opens a new shell, while entering the exclamation mark followed by a system command causes execution of that command by the operating system.

Where the parameter is set to **noosaccess**, any command preceded by an exclamation mark is ignored. Although **noosaccess** denies users access to the operating system from within PowerHouse, it does not prevent users from using Ctrl-Z to make PowerHouse run in the background. Further, it is impossible to deny any access to the operating system when the REVISE statement is executed, as any UNIX editor can be defined by the environment variable, PHEDIT.

Windows

The system prompt character is an exclamation mark (!). Entering an exclamation mark followed by a system command causes execution of that command by the operating system.

Equivalent Resource File Statement

OSACCESS ON|OFF

owner

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	

Specifies the owner for tables in an ALLBASE/SQL, DB2, SYBASE, or ORACLE database when none is explicitly indicated. Also specifies the default owner of modules created by PowerHouse in ALLBASE/SQL.

Syntax

owner=ownername

ownername

May be the database owner or a string. It may also be a system variable or file equation (**MPE/iX**), logical (**OpenVMS**), or environment variable (**UNIX, Windows**) which must be enclosed in quotation marks.

Defaults: **MPE/iX**: USERNAME@ACCOUNTNAME. **OpenVMS, UNIX, Windows:** the current username.

Limit: You must have Database Administrator (DBA) authority to create modules owned by another **ownername**.

Discussion

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.

The program parameter only applies at parse time and cannot be used at execution time to provide access to specific data.

Equivalent Resource File Statement

OWNER owner

parmfile (OpenVMS, UNIX, Windows)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔		✔	

Tells PowerHouse that the PARM values should come from the specified file instead of prompting the user.

Syntax

parmfile=filespec

filespec

The specification for a file as it is identified to the operating system. A file specification takes the general form:

OpenVMS:	**[node::][device:][[directory]] filename[.extension][;<version>]** Square brackets are required around a directory name.
UNIX:	**/[directory/]...filename.extension**
Windows:	**[drive:\][directory\]...filename.extension**

Discussion

By default, trailing blanks are stripped from values in the specified file. If this is not the desired operation, use the **parmprompt** program parameter or the TRUNCATE PARM VALUES resource file statement to change the behavior.

parmprompt

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔		✔	

Strips or does not strip trailing blanks from PARM values.

Syntax

parmprompt=truncate|notruncate

Default: **notruncate** for values entered interactively; **truncate** for values read from a file, whether in a batch job or a file specified in the **parmfile** program parameter.

truncate

Specifies that trailing blanks are stripped from any PARM values.

notruncate

Specifies that trailing blanks are significant and are not stripped from PARM values.

Equivalent Resource File Statement

TRUNCATE PARM VALUES ON|OFF

patch

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Issues a number of informational messages about patches that have been applied to the PowerHouse component.

Syntax

patch

pollspeed (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Sets the amount of time (n) that QUICK takes when polling the terminal to determine a terminal type before prompting the user.

Syntax

pollspeed=n

Limit: The range of POLLSPEED is 1 to 20 seconds.

Default: 1 second

Discussion

For more information about determining terminal types, see (p. 162).

Equivalent Resource File Statement

TERMINAL POLLING SPEED n

pre_chooseall

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔		✔	

Affects QUIZ reports and QTP runs in batch mode that use the CHOOSE statement with a PARM option and have no PARM values specified.

Syntax

pre_chooseall

In batch mode, this parameter affects QUIZ reports and QTP runs that use the CHOOSE statement with a PARM option and have no PARM values specified.

If **pre_chooseall** is specified, no records are chosen by the QUIZ reports and QTP runs. Note that this used to be the default in older versions of PowerHouse.

Default: All values of the items being prompted for are chosen.

Discussion

This program parameter has no effect if QUIZ and QTP are being run interactively.

procloc

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Causes PowerHouse to search for process files in a location other than the current directory (**OpenVMS, UNIX, Windows**) or group/account (**MPE/iX**).

Syntax

procloc=filelocation

filelocation

OpenVMS, UNIX, Windows:	filespec without the filename or extension If a logical name is used, you must include the colon (:) at the end if you specify a full file location. (**OpenVMS**)
MPE/iX:	group.account

Discussion

The **procloc** program parameter is applied to filenames that are unqualified (**MPE/iX, OpenVMS**) or not fully qualified (**UNIX, Windows**) that are specified

- in the **auto** program parameter
- through the appropriate designated files
- in the EXECUTE, SUBSCREEN (and RUN SCREEN verb), and USE statements

The **procloc** program parameter is not applied when locating the dictionary or data files.

The BUILD and SAVE statements save files in the current working directory rather than in the directory specified by **procloc**.

Equivalent Resource File Statement

LOCATION [PROCESS filelocation]

prompt

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Specifies the prompt for the PowerHouse component.

Syntax

prompt=string
Default: >

Discussion

PowerHouse adds a trailing space after the prompt string.

Equivalent Resource File Statement

PROMPT string

qktrace

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Generates a log file of QUICK screen processing that gives a summary of activities.

Syntax

qktrace [=filespec]

filespec

Names the log file.

Default: qktrace.qkt. On platforms that don't support the file extension, the name defaults to QKTRACE.

Discussion

The **qktrace** program parameter generates a log file that gives a summary of screen processing activities such as:

- screen execution and termination
- QUICK verb processing
- screen procedure execution
- actions specified in the Action field or from a menu

You can specify the name of the trace file using the program parameter. An alternative is to start QUICK with the **qktrace** program parameter and specify the filename with a file equation, logical, or environment variable.

When you start QUICK with a qktrace file equation, logical or environment variable, you must specify the **qktrace** program parameter to generate the log file.

UNIX, OpenVMS: You can specify whether to have full or partial tracing. To specify partial tracing, set the logical QKTRACE_FULL to a value of 0.

If QUICK is unable to start tracing, an error message is displayed and QUICK stops processing.

Examples

Example:

```
quick auto=screen1 qktrace
DEFINE QKTRACE <filename>
```

The default name of the log file is qktrace.qkt. On platforms that don't support file extensions, the name defaults to QKTRACE. If you want, you can specify a filename for the log file when you start QUICK.

Example:

```
QUICK qktrace=myqktrace.log
```

Another alternative is to start QUICK with the **qktrace** program parameter but specify the filename with the logical.

Example:

```
DEFINE QKTRACE myqktrace.log
quick auto=screen1 qktrace
```

When you start QUICK with the QKTRACE logical, you must specify the **qktrace** program parameter to generate the log file.

You can specify whether to have full or partial tracing. To specify partial tracing set the logical QKTRACE_FULL to a value of 0.

Example:

```
DEFINE QKTRACE_FULL 0
```

If QUICK is unable to start the tracing, then an error message is displayed and QUICK stops processing.

quotedproccall

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔		✔	

Allows quoted stored procedure calls to be passed directly to the database.

Syntax

quotedproccall

Discussion

As part of the strict SQL 92 compatibility, quoted stored procedure calls, where the quoted procedure call syntax is passed directly to the database, cause parse errors. For example:

```
> DECLARE mycursor CURSOR FOR CALL "myproc('param')"
```

is not accepted. In order to allow the double quotes and pass what is between the double quotes directly to the database, specify the **quotedproccall** program parameter at compile time.

read (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Determines how QUICK uses single-character reads.

Syntax

read=char|line

char

Causes QUICK to run the application with the use of the single character processing.

line

Causes QUICK to run the application without the use of single character processing.

Equivalent Resource File Statement

TERMINAL READ CHARACTER|LINE

resetbindvar|noresetbindvar

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔	✔	✔	

Determines if SQL bind variables are reset for each SQL statement.

Syntax

resetbindvar|noresetbindvar

Default: **resetbindvar**

resetbindvar

Causes the SQL bind variables to be reset for each SQL statement.

noresetbindvar

Causes the SQL bind variables to be different for each SQL statement.

Discussion

A bind variable is a placeholder in SQL generated at compile time where a value will be substituted at execution time. For example, if a request value for a Find is needed in generated SQL, a bind variable acts as the placeholder in the WHERE clause. Each bind variable has a unique identifier made up of a number and the field name. In versions previous to 8.4xD1, the number was incremented from statement to statement even though the field was the same. This meant that generated SQL was different even though the SQL statements themselves were the same. Because the generated SQL was different, it could not be reused by the database.

The **resetbindvar** program parameter specifies that the bind variables are to be reset for each SQL statement. This allows the generated SQL to be identical for identical SQL syntax. The bind variables will be a letter and a number. The letter is S for Select operations, U for update operations, I for insert operations, and D for delete operations. The number is incremented from 1.

Equivalent Resource File Statement

RESET BIND VARIABLES ON|OFF

resource

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Specifies the PowerHouse resource file.

Syntax

resource=filespec

restore

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Provides upward compatibility for screens compiled with versions of PowerHouse prior to 6.09 (**MPE/iX**), 6.00 (**OpenVMS**), and 6.03 (**UNIX**). The **restore** program parameter changes the default behavior of screen refreshing.

Syntax

restore=lines

Default: QUICK removes a screen from the display area when you return to a previous screen.

Discussion

The **restore=lines** program parameter changes the default behavior of screen refreshing.

By default, QUICK removes a screen from the display area when you return to a previous screen. This is the expected behavior for pop-up windows. Only the screens currently active in the screen hierarchy are visible. For certain applications, such as heavy data entry applications, you may want to leave information from a lower-level screen in the display area when you return to a previous screen. If the screen background isn't removed, QUICK can avoid rewriting the screen background each time the lower-level screen is invoked. This applies when the screens don't overlap in the display area. With extended terminal memory, QUICK can use two display areas and simply switch between them by adjusting the terminal window.

When you use the **restore** program parameter, QUICK uses a line oriented refresh algorithm. When you return to a screen, only the application lines required by that screen are refreshed in terminal memory. Application lines used by lower-level screens required by the higher-level screen aren't altered. The background remains available when the screen is again invoked.

Equivalent Resource File Statement

RESTORE LINES ON|OFF

retainmark|noretainmark

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Determines whether the field mark position in an array is retained.

Syntax

retainmark|noretainmark

Default: noretainmark

retainmark

The fieldmark position is retained.

noretainmark

The fieldmark position is not retained.

Discussion

If fieldmarking is used in an array and a new screen load is retrieved, by default the first occurrence is marked even if the mark was originally on another occurrence. RETAINMARK instructs QUICK to retain the original mark occurrence.

Equivalent Resource File Statement

RETAINMARK ON|OFF

reuse_screen_buffers|noreuse_screen_buffers

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Causes QUICK to reuse or not reuse previously allocated buffers.

Syntax

reuse_screen_buffers|noreuse_screen_buffers

Default: **noreuse_screen_buffers**

reuse_screen_buffers

Causes QUICK to reuse previously allocated buffers.

noreuse_screen_buffers

Causes QUICK to not reuse previously allocated buffers.

Discussion

The **reuse_screen_buffers** program parameter causes QUICK to reuse previously allocated buffers when the user moves repeatedly back and forth from a screen to a subscreen. Since buffers don't have to be re-allocated, performance may improve.

Applications using RDB/VMS (i.e. native access) and non-relational files should see a performance improvement by reusing screen buffers. For databases accessed through SQL, reusing screen buffers may cause more memory usage since memory is not cleaned up immediately when a screen exits.

Axiant thin-client applications should not reuse screen buffers.

This feature conflicts with the AX_SCREEN_TUNING environment variable. They should not be used together.

Equivalent Resource File Statement

REUSE SCREEN BUFFERS ON|OFF

search

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
						✔	

Causes QUIZ to search the ACCESS statement in a first-to-last or last-to-first sequence.

Syntax

search=first|last

Default: **first**

Discussion

When QUIZ looks for an unqualified item name, which may be contained in more than one of the data structures of the ACCESS statement, the **search** program parameter tells QUIZ which search sequence to follow. **first** is a first-to-last search sequence, and **last** is a last-to-first search sequence.

A first-to-last search sequence means that the data structures are searched from left to right as they are declared in the ACCESS statement. The converse applies to last-to-first search sequence.

secured

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
			✔				

Restricts the files and items listed by the SHOW FILES statement to those for which you have at least read access.

Syntax

secured

Discussion

Restricts the files listed by SHOW FILES SUMMARY statement to those for which you have at least read access. Restricts the files and items listed by the SHOW FILES DETAIL statement to those for which you have at least read access.

setjobshow|nosetjobshow (Windows)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔		✔	

Restricts the files and items listed by the SHOW FILES statement to those for which you have at least read access.

Syntax

setjobshow|nosetjobshow

Default: **nosetjobshow**

Discussion

With the SET JOB statement, the job is submitted as a separately spawned process just before the product exits. By default, the spawned process window is hidden. To show the window, use the **setjobshow** program parameter.

Equivalent Resource File Statement

SETJOBSHOW ON|OFF

statistics|nostatistics

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔		✔	✔

Specifies whether or not to display statistics.

Syntax

statistics|nostatistics

Default: **statistics** (QUIZ, QTP); **nostatistics** (QUTIL)

Discussion

In QUIZ and QTP, the **statistics** program parameter displays or lists statistics at the end of a report or run; **nostatistics** does not. Stipulating SET STATISTICS, SET NOSTATISTICS, or SET DEFAULT(QUIZ and QTP) resets the statistics control.

In QUTIL, the **statistics** program parameter produces a report of details of the file being created, such as record size, indexes, and physical location.

Limit: The **statistics|nostatistics** program parameter for QUTIL is only valid for OpenVMS, UNIX, and Windows.

Equivalent Resource File Statement

STATISTICS ON|OFF

subdictionary|subdict

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	

Specifies whether or not subdictionary support is to be enabled; when the relational subdictionaries are to be opened by PowerHouse; and when they are to be searched for unqualified record-structures.

Syntax

subdictionary=(option[,option]...)

The **subdictionary** program parameter options are **delay, nodelay, enable, disable, search,** and **nosearch**.

delay|nodelay

The **delay** program parameter indicates that subdictionaries are not to be opened by PowerHouse until referenced by an IN database qualifier, or by the subdictionary search process that looks for unqualified record-structures that do no exist in the data dictionary. The **nodelay** program parameter indicates that all subdictionaries are to be opened when you enter PowerHouse.

Default: **delay**

enable|disable

Disables or enables relational subdictionary support. If the **disable** program parameter is set, all other **subdictionary** program parameter options are ignored.

Default: **enable**

search|nosearch

The **search** program parameter specifies that if an unqualified record-structure cannot be found in the data dictionary, then PowerHouse is to search for it in the relational subdictionaries. **nosearch** specifies that PowerHouse is not to search for an unqualified record-structure.

subdict=search is only used for QDESIGN FILE statements, QUIZ and QTP ACCESS statements, and QTP OUTPUT statements. For SQL syntax, the database must be specified using the IN database option, or the SET DATABASE statement.

Default: **nosearch**

Discussion

An unqualified record-structure is a record-structure name without the IN database qualifier. The **subdictionary** program parameter can be abbreviated to **subdict**.

Equivalent Resource File Statement

SUBDICTIONARY
    DELAY|DISABLE|ENABLE|NODELAY|NOSEARCH|SEARCH
    [DELAY|DISABLE|ENABLE|NODELAY|NOSEARCH|SEARCH]...

subformat

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔		✔	

Specifies the format of subfiles being created.

Syntax

subformat = value

The values are:

MPE/iX, UNIX:	0, 1, 3, 5, 6, 7, 8
OpenVMS:	7, 8
Windows:	8

Default: 8 (all platforms)

Discussion

This program parameter allows you to specify the subfile format to be used at run-time when the format differs from the default.

If a NULL or invalid value is specified for the **subformat** parameter, an error is issued.

If the FORMAT option of the SET SUBFILE statement is used, these FORMAT specifications override a **subformat** program parameter setting. When neither a FORMAT statement option nor the **subformat** program parameter is specified, the default subfile format of 8 is used.

MPE/iX

In PowerHouse 6.09, the maximum size of an item name increased from 20 to 31 characters. PowerHouse makes use of file labels when creating subfiles, limiting the amount of information that can be stored to a subfile. Due to the increase in item information being stored to the subfile, the maximum number of items supported decreased from approximately 440 to approximately 310.

Note: This changed in later versions as well. Now, the number of items that can be written to subfiles depends on the size of the item names.

Equivalent Resource File Statement

SUBFORMAT n

term

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔			✔		

States the terminal type used and the maximum number of lines the terminal memory contains.

Syntax

term=terminal-type[-terminal-parameter]...

terminal-type

Specifies the terminal type supported by the PowerHouse component.

terminal-parameter

The terminal parameter is one of:

Parameter	Description	Platform
n	number of lines of terminal memory	all
A,B, or C	alternate character set containing line drawing characters	HP-terminals only
ANY	doesn't trap nonprinting control codes replacing them with a question mark	all
KEY; NOKEY	turns on the PowerHouse function keys on the numeric keypad	OpenVMS
MTS	used when running multipoint terminal software	MPE/iX
TAE	used for terminals with type-ahead capabilities	MPE/iX
X25	required when using X.25 communications software	MPE/iX

Discussion

In QDESIGN, **term** is passed to QUICK when the GO statement is entered.

Terminal Type When Running QUICK

To communicate properly with the user during a QUICK session, QUICK must know the type of terminal being used. A full list of supported terminal types, and necessary strapping and switch-setting information is provided with the installation package.

QUICK determines the terminal type in one of the following ways:

1. Specification (**MPE/iX, UNIX**)

   If terminal type is specified in the dictionary (on the PORTS option of the SYSTEM OPTIONS statement), QUICK uses that specification.

2. Detection

   Based on terminal characteristics, QUICK can identify most terminals automatically.

   By default, QUICK polls a terminal for one second to try to identify it. If QUICK is unsuccessful, it then prompts the user to supply a terminal type. **MPE/iX:** However, this may not be enough time for terminals on busy systems to respond. The **pollspeed=n** parameter can be used to lengthen the time QUICK waits for a response when polling a terminal. This parameter is useful only if the terminal in question can be identified by polling.

3. User response

   If the terminal type cannot be identified in steps 1 and 2, QUICK prompts for a predefined terminal type and the QUICK user responds, as in

   ```
   Terminal type=HP2392
   ```

   If the user responds with a question mark (?), a full list of acceptable terminal types is displayed.

4. Override

   To override steps 1, 2, and 3 and allow the user to specify the terminal type directly via the prompt, specify the **notermpoll** program parameter with the command that initiates QUICK, as in

   ```
   :QUICK INFO="NOTERMPOLL"
   ```

5. **term** program parameter

The **term** program parameter can be used in QDESIGN to provide the terminal type to be used by QUICK if the GO statement is entered. The format and allowable entries are the same as those used in QUICK. All previous steps are bypassed if a terminal type is specified with the command that initiates QUICK, as in

```
:QUICK INFO="TERM=HP2392"
```

The suffix ANY instructs QUICK not to trap nonprinting control codes and replace them with a question mark for display. ANY can be used to direct the terminal to an alternative character set. Steps must be taken to ensure that the control-code sequences do not interfere with QUICK's terminal display.

MPE/iX

Some terminals have more than 24 lines of memory. To tell QUICK that more than 24 lines of memory are available, specify the number of lines of memory after the terminal type. The terminal type and lines of memory must be separated by a hyphen, as in

```
:QUICK INFO="TERM=HP2624-96"
```

This terminal type specification tells QUICK that the terminal is an HP2624 and has 96 lines of memory. (The number of memory lines can also be specified in the dictionary.) If the number of lines of memory is not specified, QUICK assumes 48 lines unless the terminal profile within QUICK contains a lower number.

Line drawing is an option on some terminals; however, QUICK does not assume that line drawing is installed. To tell QUICK that line drawing is installed, specify the alternate character set (A, B, or C) where the line-drawing characters reside. Again, the terminal type and the line-drawing character set are separated by a hyphen, as in

```
:QUICK INFO="TERM=HP2624-96-B"
```

Although some terminals have more than 24 lines of memory and support highlighting, QUICK may not be able to support these features. To be supported, memory must be continuous, not paged. Many terminals have memory that can be addressed in discrete pages of 24 lines, but require that the page be identified when the user addresses a specific line. QUICK supports lines greater than 24 only if the line number itself can be addressed.

On some terminals, the highlighting control sequences require space on the terminal display. QUICK does not support highlighting of this type because the display would change from one terminal type to another.

Other suffixes can be added to the terminal specification:

- MTS (when running multipoint terminal software)
- X25 or TRANS (when using communications software)
- TAE (for terminals with type-ahead capabilities)
- ANY

When sending data in Block mode over X.25 or TRANSPAC, stacking screens will cause an error. This also applies to sending data in Block mode over MTS.

OpenVMS

PowerHouse supports function keys for the DEC VT family of terminals. On all of these types of terminals, PowerHouse can use the 18 keys on the numeric keypad. On VT 200/VT300-series terminals, PowerHouse can also use the last six of the keys located across the top of the keyboard (the others are all reserved), and the keys labeled FIND, INSERT HERE, REMOVE, SELECT, PREV SCREEN, and NEXT SCREEN.

To take advantage of your terminal's function key capability, use the **term** program parameter when you invoke PowerHouse. The syntax is

```
POWERHOUSE [TERM=]termtype
```

The **termtype** is one of VT100, VT200, VT300 or VT400 with an optional suffix of -KEY or -NOKEY. For example, you might enter

```
$ POWERHOUSE TERM=VT400-KEY
```

PowerHouse treats certain keys differently depending on which suffix you use:

- If you use -NOKEY, or do not use a suffix at all, PowerHouse does not recognize most of the numeric keypad keys as function keys, but instead treats them as numeric keys. The exceptions are [GOLD], [PF2], [PF3], and [PF4], which PowerHouse always treats as function keys when you use the TERM option. You may want to use TERM=VT400-NOKEY if you have a VT200-series terminal and you want to use the numeric keypad as a numeric keypad but still be able to use the other function keys.
- If you use -KEY, PowerHouse treats all the numeric keypad keys as function keys.

You can also use the **term** program parameter to specify the maximum number of lines the terminal memory can contain (up to a maximum of 240). The number of lines of terminal memory specified should be evenly divisible by 24. The number of lines that QUICK actually uses depends on the stacking and windowing options that you use, which may or may not make use of available application lines. Regardless of the number of terminal memory lines specified, QUICK never uses more than the application lines specified in QKGO.

Windows

The only terminal type available is WINDOWS-24.

Equivalent Resource File Statement

TERMINAL terminal-type

termpoll|notermpoll (MPE/iX, OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Specifies whether QUICK attempts to determine the terminal type before prompting the operator.

Syntax

termpoll|notermpoll
Default: **termpoll**

Discussion

For more information about determining terminal types, see (p. 162).

Equivalent Resource File Statement

TERMPOLL ON|OFF

timezone|notimezone (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔		✔	✔	✔	✔	✔	✔

Determines the mechanism that PowerHouse uses to obtain the values of SYSDATE and SYSTIME.

Syntax

timezone|notimetimezone

Default: **notimezone**

timezone

Specifies that PowerHouse takes the value of the TZ system variable into account when it returns SYSDATE and SYSTIME values. With the use of **timezone**, the value of SYSTIME is only accurate to the nearest second.

notimezone

Specifies that PowerHouse does not take the value of the TZ system variable into account when it returns SYSDATE and SYSTIME values. With **notimezone,** the value of SYSTIME is accurate to the nearest tenth of a second.

Discussion

By default, PowerHouse returns values that are based on the MPE System Time. This time is the same for all users of the system and normally reflects the correct time where the computer is located. These results are not affected by the value of the TZ system variable.

If the **timezone** program parameter or the TIME ZONE ON resource file statement is specified, then PowerHouse returns values that take the value of the TZ system variable into account. Multiple users can see different values for SYSDATE and SYSTIME by setting TZ to different values within their sessions. The TZ variable would typically be set to reflect the location where the end user is working if this is in a different time zone than where the computer is located.

For information about the TZ variable and the values that can be assigned to it, see your Hewlett Packard documentation.

Equivalent Resource File Statement

TIME ZONE ON | OFF

tpi|notpi (MPE/iX, HP-UX, Windows)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔	✔	✔	

Determines whether or not TPI or OMNIDEX indexes are seen by the PowerHouse component being run.

Syntax

tpi|notpi

Default: tpi

notpi

TPI (Third Party Indexing) or OMNIDEX indexes are not seen by the product being run.

tpi

TPI or OMNIDEX indexes are seen by the product being run. This is provided to allow you to override a resource file setting of QUIZ or OFF for the resource file statement TPI ON|OFF|QUIZ.

Discussion

The new program parameters, **tpi** and **notpi**, provide the same functionality as the existing parameters **omnidex** and **noomnidex**.

The extra keywords have been added to reflect the fact that support for TPI functionality is independent of Omnidex.

Both the existing and new keywords are available on MPE/iX, HP-UX and Windows.

Equivalent Resource File Statement

TPI ON|OFF|QUIZ

trusted|notrusted (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Activates or deactivates C2-level security for the execution of RUN commands and DCL commands within components.

Syntax

trusted|notrusted

Default: **trusted**

trusted

Indicates that OpenVMS should trust that this is a "well-behaved" application and allow DCL access.

notrusted

Indicates that OpenVMS should not trust this application and not allow DCL access.

Discussion

The security level on captive accounts was increased under OpenVMS 6.1 to prevent CAPTIVE users from spawning. This security also prevents users from executing a RUN COMMAND statement from a PowerHouse application running on a system that has C2-level security. To work around this, the System Manager changed a SYSGEN parameter to disable the new security feature.

The DCL options SPAWN|TRUSTED|NOTRUSTED allow CAPTIVE accounts to spawn. By making use of the TRUSTED flag in PowerHouse, CAPTIVE accounts can spawn, regardless of the SYSGEN setting.

By default, all DCL commands done from PowerHouse are done with TRUSTED (the pre-6.1 default setting). For all components, the use of the **nodcl** program parameter will disable DCL. In QUICK, you are still able to program RUN COMMAND statements to execute DCL commands.

The **notrusted** program parameter disables RUN COMMANDs from QUICK for CAPTIVE users.

You can use the **notrusted** program parameter in each of the components to disable all forms of DCL access for captive users.

TRUSTED CAPTIVE users can spawn.

NOTRUSTED CAPTIVE users cannot spawn.

NON-CAPTIVE users are unaffected by the use of TRUSTED|NOTRUSTED. They are always TRUSTED by OpenVMS.

The **dcl** or **osaccess** and **trusted** program parameters may be combined with the following results in CAPTIVE accounts.

PowerHouse spawns whenever it needs to access DCL, such as when interactive DCL is requested by the user, when a QUICK RUN COMMAND is executed, or a GO is requested from QDESIGN.

Parameter		InterActive DCL	RUN COMMAND
DCL	TRUSTED	allowed	allowed
N	N	N	N
N	Y	N	Y

Parameter		InterActive DCL	RUN COMMAND
DCL	**TRUSTED**	**allowed**	**allowed**
Y	N	N	N
Y	Y	Y	Y

Equivalent Resource File Statement

TRUSTED ON|OFF

update

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Controls the way PUT verbs are generated in the UPDATE procedure.

Syntax

update= bottomup|topdown|fkc_put_order

Default: **bottomup**

bottomup

By default, the ordering of PUT verbs follows the rules described in the UPDATE procedure section of Chapter 7, "QDESIGN Procedures", in the *QDESIGN Reference* book.

OpenVMS: Causes PowerHouse to generate PUT verbs in the same order they are generated in version 8.x.

topdown (OpenVMS)

Causes PowerHouse to generate PUT verbs in the same order as they are generated in version 7.10.

fkc_put_order

Causes PowerHouse to generate the order of PUT verbs based on foreign key constraints that may be in effect according to the database definition.

Discussion

For more information about these order methods, see the UPDATE procedure in Chapter 7, "QDESIGN Procedures", of the *QDESIGN Reference* book.

Equivalent Resource File Statements

UPDATE ORDER BOTTOM UP

UPDATE ORDER TOP DOWN

UPDATE ORDER FOREIGN KEY CONSTRAINT

version

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Provides the build number of the PowerHouse version.

Syntax

version

vmsdate (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	

Changes how PowerHouse creates and processes VMSDATE items.

Syntax

vmsdate = current|obsolete

current

The default behavior of PowerHouse 7.10 and previous versions is CURRENT.

Applications upgraded from PowerHouse 7.10 to 8.30 and higher should use CURRENT behavior to make their VMSDATE data transparent between versions.

Default: For PowerHouse 8.30 and higher, the default is CURRENT.

obsolete

The default behavior of PowerHouse versions 8.00, 8.10 and 8.20 (prior to 8.20D2) is OBSOLETE. No CURRENT behavior is available.

In PowerHouse versions 8.20D2 and above, both CURRENT or OBSOLETE behavior is available by using the appropriate option of the **vmsdate** program parameter or resource file statement.

To use OBSOLETE behavior in PowerHouse versions 8.20D2 or above, use the **vmsdate=obsolete** program parameter on all RUN commands or use the VMSDATE OBSOLETE resource statement in a Resource File.

Discussion

VMSDATE items have different internal formats, depending on what version of PowerHouse created them. PowerHouse version 7.10 depends on the incoming or outgoing system date being in the form of a fractional day. PowerHouse versions 8.xx versions use clock time (HHMMSSTTT).

VMSDATE items were not being converted properly in some versions. As a result, these versions cannot correctly read and report VMSDATE items created in other versions of PowerHouse; nor can they correctly read or report VMSDATE items created outside of PowerHouse. Similarly, PowerHouse versions 7.10 and 8.30 and higher, as well as external applications, cannot read PowerHouse 8.00, 8.10, or 8.20 VMSDATE items correctly. A new program parameter, **vmsdate=obsolete|current**, was added to PowerHouse 8.20D2 and later to resolve this problem.

PowerHouse 8.00, 8.10, and 8.20 process VMSDATEs using the **vmsdate=obsolete** behavior. PowerHouse 8.30 and higher defaults to **vmsdate=current** behavior. PowerHouse 7.10 defaults to external OpenVMS behavior, which is the same as **vmsdate=current**.

VMSDATE Conversions

For information about VMSDATE conversions and compatibility information, see Appendix B, in the *PowerHouse 4GL Version 8.30 for OpenVMS Upgrade Guide*.

Equivalent Resource File Statement

VMSDATE = CURRENT|OBSOLETE

Chapter 3: Resource File Statements

Overview

This chapter describes the resource file statements you can use to specify program parameters and other system characteristics that you want your users to use with PowerHouse applications.

About Resource File Statements

You can create PowerHouse resource files that serve as a single location in which to specify program parameters and other system characteristics you want your users to use with PowerHouse applications.

MPE/iX:	The resource file is specified by having a PHRS file in the current group and account, or by a file equation pointing to the appropriate file. For example, `FILE PHRS=SYSRES.PUB.SYS`
OpenVMS:	The resource file is specified with the PHRS logical or in the ph.rc file in your current location.
UNIX:	The resource file is specified with the PHRS environment variable or in the .phrc file in the HOME directory.
Windows:	The resource file is specified with the PHRS environment variable or in the phrc file in the HOME directory.

Resource files contain a list of one or more statements, which are documented on the following pages. These statements can be included in any order, but the last statement must be EXIT or QUIT.

The following is an example of a PowerHouse resource file:

```
DBAUDIT MSGS
OSACCESS OFF
PROMPT '-'
QUIT
```

Summary of Resource File Statements

The following table is a summary of each resource file statement.

Statement	Description
ALLBASE MODULE EXTENSION (**MPE/iX**)	Causes the module names that are stored with compiled sections in an ALLBASE/SQL database to be modified.
AUTODETACH	Automatically detaches database connections.
BROADCAST (**OpenVMS**)	Determines how QUICK handles broadcast, or non-PowerHouse messages.

Statement	Description
BULKFETCH	Specifies the number of rows a bulkfetch returns.
CC	Sets the conditional compile flags you can use in the statements of each PowerHouse component to tell it whether to process or skip blocks of code.
CHECKSUM710 (**OpenVMS**)	Allows PowerHouse 8.30 and later versions to use the 7.10 form (unsigned) in the CHECKSUM calculations.
CLOSE DETACH	Indicates that CLOSE verbs encountered in QUICK will cause a physical database detach.
COLUMNOWNER	Determines how the item names in a cursor are generated by PowerHouse Services.
COMMITPOINTS OBSOLETE	Enables the default commit timing for COMMIT ON UPDATE used for versions prior to 7.2x.
COMPRESS BUFFERS	Causes the initialization pool in QDESIGN to be compressed before it is stored.
CONSOLE KEYS (**Windows**)	Instructs QUICK to display function keys.
DATABASE	Establishes the default relational database.
DBAUDIT	Enables database monitoring for PowerHouse components.
DBDETACH	QUICK detaches from database connections when returning to the screen prompt.
DBWAIT	Specifies whether the program waits until a concurrency conflict is resolved.
DEBUG	Controls the level of QUICK's debugging capabilities.
DEFAULT CURSOR OWNER	Causes QDESIGN to use the owner name in the default query and in generated FIELD statements for the query.
DESIGNER NORETAIN	Causes a commit to end the transaction.
DICTIONARY	Establishes the data dictionary as well as the dictionary type (**OpenVMS**) that the PowerHouse component opens when it starts.
DIRECTORY (**UNIX, Windows**)	Moves you to the specified location when a PowerHouse component processes the resource file.
DISABLE NULLS	Controls whether null support is allowed at the item level. It overrides the dictionary setting.
ENTRY RECALL	Specifies that data from previous screens is available for recall in Entry mode.
EXIT	Ends the PowerHouse resource file.
HPSLAVE EXTRA LINE	Suppresses the printing on an extra blank line at the end of reports.

Statement	Description
HPSLAVE SPLIT LINES (**MPE/iX, UNIX**)	Ensures that no line of text for the printer is split between two output blocks.
INITIALIZE NULLS	Specifies whether to initialize columns to NULL in rows not retrieved.
INTEGER SIZE 6 (**OpenVMS**)	Controls how the products work regarding physical record lengths.
JCWBASE (**MPE/iX**)	Specifies the base value for JCW settings.
LIST	Establishes whether or not the PowerHouse component displays the source statement file.
LOCATION MODULE (**MPE/iX**)	Specifies the location and compiles a PowerHouse program into an intallable module for ALLBASE/SQL.
LOCATION PROCESS	Causes PowerHouse to search for process files in a location other than the current location.
LOCKWORD (**MPE/iX**)	Enables the use of internal routines to prevent duplicate lockword prompting during screen loading and running.
NOBLOBS	Specifies whether blobs columns are processed.
NONPORTABLE	Sets the message severity for nonportable syntax.
NOOWNER	Prevents the owner name from being attached to the table name in generated code so that different users can use the same compiled screen to access their own tables (having the same name).
NOSET WARN STATUS	Suppresses the warning status when a warning condition is detected in PowerHouse.
OBSOLETE	Sets the severity of the messages when obsolete syntax is encountered.
OMNIDEX (**MPE/iX**)	Determines whether or not OMNIDEX indexes are seen by the PowerHouse component being run.
OSACCESS	Determines whether operating system command entry and execution is allowed.
OWNER	Specifies the owner for tables in an ALLBASE/SQL, DB2, SYBASE or ORACLE database when none is explicitly indicated. Also specifies the default owner of modules created by PowerHouse in ALLBASE/SQL.
PREFIX ORACLE OPEN NAME	Suppresses the addition of "ORACLE@" to the open name for an ORACLE database.
PROMPT	Specifies the prompt string for the PowerHouse component.
QUIT	Ends the PowerHouse resource file.
RESET BIND VARIABLES	Determines if SQL bind variables are reset for each SQL statement.
RESTORE LINES	Changes the default behavior of screen refreshing.
RETAIN MARK	Determines whether the field mark position in an array is retained.

Statement	Description
REUSE SCREEN BUFFERS	Causes QUICK to reuse or not reuse previously allocated buffers.
RMS FAST READ (**OpenVMS**)	Performs a block read to sequentially accessed read-only files.
RMS FILE BASE (**OpenVMS**)	Allows the use of zero-based record numbers for cross-platform compatibility.
SET	Determines whether the PowerHouse resource file is listed before PowerHouse components are initiated.
SETJOBSHOW (**Windows**)	Determines whether the window for a spawned SET JOB process is shown or hidden.
SHIFT	Determines how PowerHouse identifiers, such as item names, are shifted.
STATISTICS	Determines whether statistics are displayed at the end of QUIZ reports and QTP runs. Determines if QUTIL displays additional information about the file being created.
STORE MODULES	Prevents SQL modules from being compiled and stored in the database. This statement is used at parse time.
SUBDICTIONARY	Specifies whether subdictionary support is enabled; when the relational subdictionaries are opened by PowerHouse; and when they are searched for unqualified record-structures.
SUBFORMAT	Specifies the format of subfiles being created.
TERMINAL	Establishes the terminal type.
TERMINAL BLOCKMODE (**MPE/iX**)	Determines whether QUICK recognizes BLOCK TRANSFER control structures in Block mode.
TERMINAL CHARACTERMODE	Specifies whether QUICK recognizes BLOCK TRANSFER control structures in Character mode.
TERMINAL CONFIRMER	Changes confirmation messages to pop-up windows with OK/CANCEL buttons.
TERMINAL POLLING SPEED (**MPE/iX**)	Sets the amount of time (n) that QUICK takes when polling the terminal to determine a terminal type before prompting the user.
TERMINAL READ (**MPE/iX**)	Determines how QUICK uses single-character reads.
TERMPOLL (**MPE/iX, OpenVMS**)	Determines whether to poll the terminal to establish the terminal type.
TIC RESOURCE FILE (**UNIX, Windows**)	References the QUICK Terminal Interface Configuration resource file.
TIME ZONE (**MPE/iX**)	Determines the mechanism that PowerHouse uses to obtain the values of SYSDATE and SYSTIME
TPI (**MPE/iX, HP-UX, Windows**)	Determines whether TPI or OMNIDEX indexes are seen by the PowerHouse component being run.

Statement	Description
TRUNCATE PARM VALUES	Strips or does not strip trailing blanks from PARM values.
TRUSTED (**OpenVMS**)	Activates or deactivates C2-level security for the execution of RUN commands and DCL commands within components.
UIC BRACKETS (**OpenVMS**)	Causes the UIC function to not return brackets around the result values.
UPDATE ORDER	Controls the way PUT verbs are generated in the UPDATE procedure.
USE	Calls another PowerHouse resource file and processes its statements.
VMSDATE (**OpenVMS**)	Determines how VMSDATE datatypes are processed. There are two methods: the method used for 7.10 and after 8.20D2 and the method used prior to 8.20D2 in 8.xx PowerHouse.

The following table lists the resource file statements that are applicable to each PowerHouse component.

Statement	PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
ALLBASE MODULE EXTENSION (**MPE/iX**)				✔	✔	✔		
AUTODETACH						✔		
BROADCAST (**OpenVMS**)						✔		
BULKFETCH					✔	✔	✔	
CC	✔	✔	✔	✔	✔	✔	✔	✔
CHECKSUM710 (**OpenVMS**)		✔	✔	✔	✔	✔	✔	✔
CLOSE DETACH						✔		
COLUMNOWNER			✔		✔		✔	
COMMITPOINTS OBSOLETE			✔					
COMPRESS BUFFERS			✔					
CONSOLE KEYS (**Windows**)						✔		
DATABASE			✔	✔	✔	✔	✔	
DBAUDIT			✔	✔	✔	✔	✔	
DBDETACH						✔		
DBWAIT			✔	✔	✔	✔	✔	
DEBUG						✔		
DEFAULT CURSOR OWNER			✔					
DESIGNER NORETAIN			✔					

Statement	PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
DICTIONARY	✔	✔	✔	✔	✔	✔	✔	✔
DIRECTORY (**UNIX, Windows**)	✔	✔	✔	✔	✔	✔	✔	✔
DISABLE NULLS			✔					
ENTRY RECALL						✔		
HPSLAVE EXTRA LINE							✔	
HPSLAVE SPLIT LINES (**MPE/iX, UNIX**)							✔	
INITIALIZE NULLS					✔			
INTEGER SIZE 6 (**OpenVMS**)	✔	✔						
JCWBASE (**MPE/iX**)							✔	
LIST	✔	✔	✔	✔	✔		✔	✔
LOCATION MODULE (**MPE/iX**)			✔		✔	✔	✔	
LOCATION PROCESS	✔	✔	✔	✔	✔	✔	✔	✔
LOCKWORD (**MPE/iX**)			✔			✔		
NOBLOBS			✔		✔		✔	
NONPORTABLE	✔	✔	✔	✔	✔		✔	✔
NOOWNER	✔	✔	✔	✔	✔	✔	✔	✔
NOSET WARN STATUS	✔	✔	✔	✔	✔	✔	✔	✔
OBSOLETE		✔	✔	✔	✔	✔	✔	✔
OMNIDEX (**MPE/iX**)			✔	✔	✔	✔	✔	✔
OSACCESS	✔	✔	✔	✔	✔	✔	✔	✔
OWNER			✔	✔	✔	✔	✔	
PREFIX ORACLE OPEN NAME	✔	✔	✔	✔	✔	✔	✔	
PROMPT	✔	✔	✔	✔	✔	✔	✔	✔
RESET BIND VARIABLES			✔		✔	✔	✔	
RESTORE LINES						✔		
RETAIN MARK						✔		
REUSE SCREEN BUFFERS						✔		
RMS FAST READ (**OpenVMS**)					✔	✔	✔	

Statement	PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
RMS FILE BASE (**OpenVMS**)			✔	✔	✔	✔	✔	
SETJOBSHOW (**Windows**)					✔		✔	
SHIFT	✔		✔	✔	✔	✔	✔	✔
STATISTICS					✔		✔	✔
STORE MODULES			✔		✔		✔	
SUBDICTIONARY			✔	✔	✔	✔	✔	
SUBFORMAT					✔		✔	
TERMINAL			✔			✔		
TERMINAL BLOCKMODE (**MPE/iX**)						✔		
TERMINAL CHARACTERMODE						✔		
TERMINAL CONFIRMER						✔		
TERMINAL POLLING SPEED (**MPE/iX**)						✔		
TERMINAL READ (**MPE/iX**)						✔		
TERMPOLL (**MPE/iX, OpenVMS**)						✔		
TIC RESOURCE FILE (**UNIX, Windows**)						✔		
TIME ZONE (**MPE/iX**)	✔		✔	✔	✔	✔	✔	✔
TPI (**MPE/iX, HP-UX, Windows**)			✔		✔	✔	✔	
TRUNCATE PARM VALUES					✔		✔	
TRUSTED (**OpenVMS**)	✔	✔	✔	✔	✔	✔	✔	✔
UIC BRACKETS (**OpenVMS**)	✔	✔	✔	✔	✔	✔	✔	
UPDATE ORDER			✔					
VMSDATE (**OpenVMS**)	✔	✔	✔	✔	✔	✔	✔	✔

ALLBASE MODULE EXTENSION (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔	✔	✔	

Causes the module names that are stored with compiled sections in an ALLBASE/SQL database to be modified.

Syntax

ALLBASE MODULE EXTENSION string

Discussion

The module name is based on the name of the file in which the compiled QUICK screen, QUIZ report, or QTP run is stored.

If the fully qualified name of the compiled file is

```
file.group.account
```

then the default module name is

```
file_group
```

Equivalent Program Parameter

moduleext=extension

AUTODETACH

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Automatically detaches database connections.

Syntax

AUTODETACH ON|OFF

Default: ON

Limit: Applies only to Sybase.

ON

Automatically detaches database connections.

OFF

Retains database connections.

Discussion

The AUTODETACH Resource file statement applies only to relational databases that support a single transaction per database attach. The ON option specifies that QUICK automatically detaches the database connections if the transactions are committed or rolled back, and are not locally active when the user exits the screen. This minimizes the number of attaches for those single transaction databases. Currently, Sybase is the only database that fits this category.

PowerHouse versions prior to 8.4E did not detach automatically causing a growth in the number of attaches over time. The OFF option is provided to allow the pre-8.4E behavior to be specified.

Equivalent Program Parameter

autodetach|noautodetach

BROADCAST (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Determines how QUICK handles broadcast, or non-PowerHouse messages.

Syntax

BROADCAST DEFAULT | DEFERRED

Default: DEFERRED

DEFAULT

QUICK bases its treatment of non-PowerHouse messages on the specification established by the DCL SET BROADCAST command.

DEFERRED

Non-PowerHouse messages are trapped and displayed on the message line when QUICK performs the next I/O to the terminal.

Equivalent Program Parameter

broadcast=default|deferred

BULKFETCH n

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔	✔	✔	

Specifies the number of rows a bulkfetch returns.

Syntax

BULKFETCH n

n

The number of rows to return.

Default: 0

Limit: 32767

Limit: Applies only to relational databases.

Discussion

The BULKFETCH resource file statement allows you to specify the number of rows a retrieval returns and, therefore, the amount of memory allocated to the retrieval buffer. When retrieving rows from relational databases, PowerHouse often fetches more than one row at a time to improve performance. Changing the bulkfetch value may help performance depending on the retrieval situation. The BULKFETCH resource file statement has no effect on non-relational retrieval.

The amount of memory allocated is approximately the size of the rows times the number of rows to be retrieved. There is a trade off between the memory allocated and the performance improvement. Allocating too much memory impacts performance adversely in a multi-user environment. The default internal value (also available by setting bulkfetch=0) is conservative and very low. Trial and error is the best way to determine the optimal improvement in specific environments. A value of 512 is a good starting point.

The BULKFETCH Resource file statement only helps in one-to-many or many-to-many linkages. It will not help in one-to-one or many-to-one linkages, whether unique or not, since multiple rows must be returned for bulkfetch to have any effect.

Equivalent Program Parameter

bulkfetch=n

CC

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Sets the conditional compile flags you can use in the statements of each PowerHouse component to tell it whether to process or skip blocks of code.

Syntax

CC name[, name]...

name

A unique name identifying a conditional compile flag. The names are not special keywords in themselves, but they are referenced by conditional compile constructs.

Limit: Maximum of 127 characters.

Discussion

Sets conditional compile flags you can use in PowerHouse statements to tell PowerHouse whether to process or skip blocks of code. PowerHouse adds an underscore in the listings to indicate skipped code, as in

```
> SCREEN PROJECT
> @IF UNIX
> DEFINE END_DATE DATETIME = SYSDATETIME
> @ELSEIF OPENVMS
>_DEFINE END_DATE VMSDATE = VMSTIMESTAMP
>_@ELSE
>_DEFINE END_DATE DATE = SYSDATE
>_@ENDIF
>
```

This resource file statement is only effective at compile time; that is, it cannot change compiled files.

In QUICK, **cc** can only be used in conjunction with **debug**.

For more information about how to use compile-time flags and a list of predefined flags, see (p. 282).

Equivalent Program Parameter

cc=(name[,name]...)

CHECKSUM710

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
	✔	✔	✔	✔	✔	✔	✔

Allows PowerHouse 8.30 and later versions to use the 7.10 form (unsigned) in the CHECKSUM calculations.

Syntax

CHECKSUM710 [ON|OFF]

Specifying CHECKSUM710 without an option is the same as specifying it with an ON option.

Default: version 7.10: on. versions 8.xx, OFF.

Discussion

CHECKSUM710 Backwards Compatibility Switch

The CHECKSUM function in 7.10 produces a different result than in 8.xx. While the algorithm used is the same, the internal input into that calculation is Unsigned in 7.10 but Signed in 8.xx. This produces different results from the same calculation.

This is only an issue if you store the results of the CHECKSUM function in your data files, check that value on processing the data, and share and migrate these data files between 7.10 and 8.30 and above.

There are two ways to deal with this issue. If you are migrating your data and application, and do not need to share with 7.10, then you may want to consider just recalculating the CHECKSUM values in your files and move upwards. This is the recommended route in this case.

If you need to share data, we have endeavored to provide backward compatibility that allows you to do so. However, if you use it then you must use the 7.10 form for all applications using the same data in PowerHouse 8.30 and above. If, in the future, you drop the 7.10 requirements, you may recalculate the checksums at that point and drop the compatibility mode.

To allow PowerHouse 8.30 and later versions to use the 7.10 form (unsigned) in the CHECKSUM calculations, you need to use the CHECKSUM710 logical, program parameter or resource file statement. If possible, we recommend that you use the logical so that all PowerHouse components automatically use this setting. Depending on your environment and needs, this can be set at a process, group, or system level. You could, alternatively, use a resource file defined at any of these levels, or a program parameter on each execution.

The logical syntax is $DEF CHECKSUM710 "ON"|"OFF" (here the options are not optional as a null string is not a valid logical definition).

The precedence rules are as follows: a program parameter overrides a resource file statement which overrides a logical name setting.

Using CHECKSUM in PHD Dictionaries

If you are using PHD dictionaries, you need to recalculate the checksums in your dictionaries. The same procedure can be used to return the checksums to 8.xx values at a later date, if desired.

CHECKSUM.COM is found in PHD_LOCATION:. It has two parameters. The first parameter is your dictionary name (it modifies the dictionary directly). The second parameter is either 7 or 8, to indicate the form used when recalculating the dictionary checksums. Seven uses CHECKSUM710=ON, and 8 uses CHECKSUM710=OFF. For example:

```
@PHD_LOCATION:CHECKSUM <userdict> 7
```

This example would recalculate the checksums in a user dictionary from 8.xx to 7.10-like values so that you could then use the CHECKSUM710 program parameter, resource file statement, or logical when using these dictionaries in PowerHouse.

Also, there is an additional parameter to PHDMAINT, PHDADMIN, and the POW and PHDCONV commands. The parameter is CHECKSUM710(=ON|OFF). For all these command procedures, if the logical CHECKSUM710 is set to the option you require, the parameter does not need to be used.

When switching between PHD dictionaries with one setting or the other, you must reset the dictionary before starting up the application or you will get a "Corrupted dictionary" error when the product tries to open the dictionary. Any time you use the wrong option when trying to access a dictionary, you get the "Corrupted dictionary" error.

Equivalent Program Parameter

checksum710[=on|off]

CLOSE DETACH

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Indicates that CLOSE verbs encountered in QUICK will cause a physical database detach.

Syntax

CLOSE DETACH

Discussion

When the CLOSE DETACH resource file statement is used, CLOSE verbs cause an immediate physical database detach. When the statement is not used, detaches are only done upon exit of the screen where the attach was done.

In addition, when the resource file statement is used, Oracle open names are not prefixed with "ORACLE@". Without the prefix, users can specify a logical name which could be set to different values, and thus point to different databases.

Equivalent Program Parameter

close_detach

COLUMNOWNER

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔		✔	

Determines how the item names in a cursor will be generated by PowerHouse Services.

Syntax

COLUMNOWNER

Discussion

Correlation names are qualified metadata references to column names appearing in PowerHouse applications as ITEM or FIELD names.

PowerHouse 8.4x's underlying database access software attempts to conform more strictly to the SQL92 standard, which describes column correlation names as table_name.column_name. The owner name is no longer included before the table_name.

In previous versions of PowerHouse, correlation names sometimes included the owner name and sometimes did not, depending on the specification of the SQL statement. Below, is a table showing sample SQL SELECT statements and indicating the resultant column correlation names:

	SQL Statements	Correlation Name
a)	SELECT COLUMN FROM TABLE	TABLE.COLUMN
b)	SELECT COLUMN FROM OWNER.TABLE	OWNER.TABLE.COLUMN
c)	SELECT TABLE.COLUMN FROM TABLE	TABLE.COLUMN
d)	SELECT TABLE.COLUMN FROM OWNER.TABLE	TABLE.COLUMN
e)	SELECT OWNER.TABLE.COLUMN FROM TABLE	TABLE.COLUMN
f)	SELECT OWNER.TABLE.COLUMN FROM OWNER.TABLE	OWNER.TABLE.COLUMN

For applications being upgraded to 8.4x, the COLUMNOWNER resource file statement enables the successful parsing of column names permitted in earlier versions. If COLUMNOWNER is specified, the owner name is obtained from other metadata sources for the column and prefixed on to the correlation name.

This allows applications coded prior to PowerHouse 8.4x to compile and execute without changing all column names that appear in the old format.

For applications with cursors defined in the form of examples (b) and (f) the COLUMNOWNER resource file statement may be used.

If an application has multiple cursors defined in mixed forms, for example, one cursor similar to (b) and another similar to (d), it may be necessary to make manual changes to the PowerHouse syntax since the resource file statement won't distinguish between the different formats and will always add the owner name.

For PowerHouse syntax being created with the 8.4x releases, all column references should omit the owner name.

Equivalent Program Parameter

columnowner

COMMITPOINTS OBSOLETE

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Enables the default commit timing for COMMIT ON UPDATE used for versions prior to 7.2x.

Syntax

COMMITPOINTS OBSOLETE

Discussion

All screens that require the pre-7.2x commit timing must be compiled using this statement or program parameter.

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

Equivalent Program Parameter

commitpoints=obsolete

COMPRESS BUFFERS

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Causes the initialization pool in QDESIGN to be compressed before it is stored.

Syntax

compress_buffers

Discussion

Using this statement means that the physical size of screens will be decreased if the data is compressible. However, it does impose a certain overhead on the reading of screens since this data must be uncompressed before it can be used.

Equivalent Program Parameter

compress_buffers

CONSOLE KEYS (Windows)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Instructs QUICK to display function keys.

Syntax

CONSOLE KEYS ON|OFF

Default: OFF

ON

Instructs QUICK to display function keys at the bottom of the Command Console window.

OFF

Instructs QUICK to not display function keys.

Discussion

The CONSOLE KEYS Resource file statement instructs QUICK to display eight function keys across the bottom of the Command Console window under the QUICK screen. These labels are not clickable using a mouse and only represent the function keys and labels.

Neither the **consolekeys** program parameter nor the CONSOLE KEYS Resource file statement has any effect on displaying function keys in QKView. To display function keys in QKView, select the Function Keys entry in the View menu.

Equivalent Program Parameter

consolekeys|noconsolekeys

DATABASE

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	

Establishes the default relational database.

Syntax

DATABASE database [OPEN filespec|open-name-string]
[PASSWORD string] [USERID string]

database

The name of the database as it is identified to the PowerHouse dictionary through the PDL DATABASE statement.

OPEN filespec|open-name-string

Specifies the physical 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 name of the database as it is known to the operating system. For ALLBASE/SQL, the filespec must point to the root DBEnvironment.

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

Limits: For ALLBASE/SQL and Oracle Rdb 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 looks for default environment variables or logicals that are specific to running that database's environment.

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

PASSWORD string

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.

Limit: For DB2, ODBC, ORACLE, and SYBASE databases only. This option is required if a password is not included in the OPEN open-name-string option or in an associated PDL DATABASE statement.

USERID string

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.

Limit: For DB2, ODBC, ORACLE, and SYBASE databases only. This option is required if a userid is not included in the OPEN open-name-string option or in an associated PDL DATABASE statement.

Discussion

PowerHouse combines the OPEN, USERID, and PASSWORD options into a valid open name for its data management component. The separator between OPEN and USERID is an at-sign (@), and the separator between USERID and PASSWORD is a slash (/).

If you specify passwords on DATABASE statements when other users are allowed to read or write to the resource file, then PowerHouse issues the following warning message:

```
The Resource File ^ contains passwords, but is not properly protected.
```

DBAUDIT

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	

Enables database monitoring for PowerHouse components.

Syntax

DBAUDIT BRIEF|FILE|FULL|MESSAGES|MSGS|NONE

BRIEF

Displays a short (one line) information line as database operations occur.

FILE

Writes detailed information to a file called dbaudit.txt located, by default, in the current working directory.

FULL

Displays detailed information as database operations occur.

MESSAGES|MSGS

Displays Binary Language Representation (BLR) messages in hexadecimal.

NONE

Does not produce any output.

Discussion

When the DBAUDIT BRIEF resource file statement is specified, the following lines describe the output produced:

```
ATTACH db_handle TO db_type db_name
COMPILE REQUEST request_handle
START LOGICAL TRANSACTION trans_name details

PREPARE LOGICAL TRANSACTION trans_name
COMMIT LOGICAL TRANSACTION trans_name
ROLLBACK LOGICAL TRANSACTION trans_name
START TRANSACTION trans_handle IN dbhandle_list

START REQUEST request_handle IN TRANSACTION trans_handle
RELEASE REQUEST request_handle FROM TRANSACTION trans_handle

PREPARE TRANSACTION trans_handle
COMMIT TRANSACTION trans_handle
ROLLBACK TRANSACTION trans_handle
DETACH db_handle FROM db_name
```

db_handle

A unique numeric value that identifies a database.

db_type

Either "ALLBASE", "DB2", "ODBC", "ORACLE", "RDB" or "SYBASE".

db_name

The database name.

request_handle

A unique numeric value that identifies a request.

trans_name

The transaction name.

details

May contain a combination of the following:

- active, all active, locally active
- read only|read write
- wait|nowait
- read committed|cursor stability|reproducible read|phantom protection|serializable
- reserving reserve_name_comma_list (A list of the relation names specified for this transaction).

dbhandle_list

A comma-separated list of dbhandle.

Equivalent Program Parameter

dbaudit=brief|file|full|msgs

DBDETACH

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Releases or does not release the database connection when you return to the screen prompt.

Syntax

DBDETACH ON|OFF

Default: DBDETACH OFF

ON

Releases the database connection when you return to the screen prompt.

OFF

Keeps the database attached when you return to the screen prompt.

Discussion

When you leave a QUICK screen and go back to the screen ID prompt after processing through a relational database connection, QUICK can either detach from the database or keep the connection. If you keep the connection (not detaching), memory is still allocated for the connection. This means that when you call another screen, QUICK allocates more memory for the new database connection(s). This causes memory growth in the product.

The default is DBDETACH OFF. While this uses more memory, there may be small performance benefits. Note that this only affects screens called from the screen ID prompt which is not typically used in production environments.

Equivalent Program Parameter

dbdetach|nodbdetach

DBWAIT

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	

Specifies whether the program waits until a concurrency conflict is resolved.

Syntax

DBWAIT ON|OFF

Limit: Applies only to ALLBASE/SQL, ORACLE.

Default (except for QUICK): DBWAIT OFF

Default for QUICK: DBWAIT ON

ON

The program waits until a concurrency conflict is resolved.

OFF

The program does not wait until a concurrency conflict is resolved.

Discussion

The DBWAIT ON statement specifies that if a concurrency conflict occurs during access to a relational database, the program normally waits until the conflict is resolved. An example of concurrency conflict is attempting to write a record that has been locked by another user. If the DBWAIT OFF statement is specified and the database encounters a concurrency conflict, an error message results.

Equivalent Program Parameter

dbwait|nodbwait

DEBUG

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Controls the level of QUICK's debugging capabilities.

Syntax

ERROR

Returns an error message for screens compiled for Debugger.

NOWARNING

Runs screens compiled for Debugger but does not allow access to Debugger.

SOURCE

Runs compiled screens and allows full access to Debugger.

WARNING

Displays a warning message when a screen is encountered that was compiled with Debugger enabled.

Discussion

To debug your QUICK screens you must have compiled them using the **debug** program parameter in QDESIGN.

Equivalent Program Parameter

debug=source|warn|nowarn|error

DEFAULT CURSOR OWNER

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Determines whether or not QDESIGN uses the owner name in the default query and in the generated FIELD statements for the query.

Syntax

DEFAULT CURSOR OWNER ON|OFF

ON

Causes QDESIGN to use the owner name.

OFF

The owner name is not used in the default query and in the generated FIELD statements for the query.

Default: OFF

Discussion

If you want to access a table which does not belong to you, you must specify the owner name. You can do this in two ways:

- by directly coding the owner name in SQL queries and FIELD statements
- by using the OWNER and DEFAULT CURSOR OWNER resource file statements to include the owner name in generated SQL queries and FIELD statements.

When QDESIGN generates a default SQL query from a CURSOR statement, it does not use the owner name defined in the dictionary or in the OWNER statement. The DEFAULT CURSOR OWNER causes QDESIGN to use the current owner name in:

- the generated default SQL query.
- generated FIELD statements for the fields in the CURSOR statement.

When DEFAULT CURSOR OWNER is specified and FIELD statements are coded, you can only use the following syntax:

```
FIELD CURSOR_COLUMN OF CURSOR_STRUCTURE
```

or

```
FIELD OWNERNAME.CURSOR_STRUCTURE.CURSOR_COLUMN OF CURSOR_STRUCTURE
```

If the ownername changes, any screens compiled with the DEFAULT CURSOR OWNER statement must be recompiled.

Equivalent Program Parameter

cursorowner

DESIGNER NORETAIN

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Causes a commit to end the transaction.

Syntax

DESIGNER NORETAIN

Discussion

By default, DESIGNER files use the commit retain functionality, which means the transaction is kept open after a commit. By using this statement, you can cause a commit to end the transaction.

Equivalent Program Parameter

designer_noretain

DICTIONARY

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Establishes the data dictionary that the PowerHouse component opens when it starts.

Syntax

DICTIONARY filespec TYPE PHD|PDL

filespec

The specification for a file, as it is identified to the operating system. A file specification takes the general form

MPE/iX:	**[*]filename[/lockword][.group[.account]]**
OpenVMS:	**[node::][device:][directory] filename[.extension][;<version>]** Square brackets are required around a directory name.
UNIX:	**/[directory/]...filename.extension**
Windows:	**[drive:\][directory\]...filename.extension**

OpenVMS: For PHD type dictionaries, extension and version are not valid options.

TYPE (OpenVMS)

Specifies the default dictionary type that the PowerHouse component uses during the session.

PDL

Specifies a PDC dictionary. PDL dictionaries have a .pdc extension.

PHD

Specifies a PHD dictionary. PHD dictionaries have a .phd extension.

Discussion

The DICTIONARY statement temporarily establishes the dictionary as specified for the duration of a session of a PowerHouse component but does not override any dictionary previously set up.

OpenVMS:	The default dictionary type for all components except PDL is PHD. If no PHD dictionary is found, the components look for a dictionary with the .pdc extension.
UNIX, Windows:	The default extension when specifying a dictionary is .pdc.

OpenVMS

When TYPE is specified, it applies to all SET DICTIONARY statements where the TYPE option is not specified. The dictionary type specified on the SET DICTIONARY statement overrides the TYPE option of the resource file DICTIONARY statement. If a type is not specified in the DICTIONARY statement, PowerHouse searches first for a PHD dictionary, then for a PDC dictionary.

If an extension is specified in the resource file DICTIONARY statement and a conflicting dictionary TYPE is specified, you will get an error.

Equivalent Program Parameter

dictionary=filespec or dict=filespec

dicttype=pdl|phd or dt=pdl|phd

DIRECTORY (UNIX, Windows)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Moves you to the specified location when a PowerHouse component processes the resource file.

Syntax

DIRECTORY location

location

Specifies the location of the resource file. Location has the general form:

/[directory/]

DISABLE NULLS

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Controls whether null support is allowed at the item level. It overrides the dictionary setting.

Syntax

DISABLE NULLS

Equivalent Program Parameter

disable_nulls

ENTRY RECALL

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Specifies that data from previous screens is available for recall in Entry mode.

Syntax

ENTRY RECALL ON|OFF

Default: OFF

Discussion

Specifying ENTRY RECALL means that QUICK users can recall the previous record's values in Entry mode. Values are only recalled and displayed if requested using the Recall command (the Up Arrow, Ctrl-B, or whatever key has been set).

Users can change the displayed value before it is processed by QUICK. The cursor is positioned immediately to the right of the recalled value as if the user had typed it into the field. The positioning is to the left if the REVERSE option of the FIELD statement is specified. Error recall, and the recall of data in change processing, is not affected by ENTRY RECALL.

Note: You can also duplicate the previous record's values using the Duplicate command (by default, the underscore), but you cannot change the duplicated value before it is processed by QUICK.

Equivalent Program Parameter

entryrecall

EXIT

Ends the PowerHouse resource file.

Syntax

EXIT

Discussion

The EXIT statement ends the PowerHouse resource file. Any statements after the EXIT statement are ignored.

The EXIT and QUIT statements are interchangeable.

HPSLAVE EXTRA LINE

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
						✔	

Suppresses the printing on an extra blank line at the end of reports.

Syntax

HPSLAVE EXTRA LINE ON|OFF
Default: ON

ON

Suppresses the printing on an extra blank line at the end of reports.

OFF

Does not suppress the printing on an extra blank line at the end of reports.

Equivalent Program Parameter

nxl

HPSLAVE SPLIT LINES (MPE/iX, UNIX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
						✔	

Ensures that no line of text for the printer is split between two output blocks.

Syntax

HPSLAVE SPLIT LINES ON|OFF

Default: ON

ON

Ensures that no line of text for the printer is split between two output blocks.

OFF

Does not ensure that no line of text for the printer is split between two output blocks.

Discussion

On some terminals in the HPSLAVE output block method, when a line of output to the printer is split between two output blocks, the portion of the line in the first block is overwritten by the portion of the line in the second block. The OFF setting prevents this from happening.

Equivalent Program Parameter

nls

INITIALIZE NULLS

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔			

Specifies whether to initialize columns to NULL in rows not retrieved.

Syntax

INITIALIZE NULLS ON|OFF

Defaults: **OFF**

Discussion

Columns in rows not retrieved should be initialized to NULL if null values are allowed. This is what happens in QUIZ. In QTP, columns are initialized to spaces, zeroes, and dictionary initial values. The INITIALIZE NULLS ON resource file statement can be used to tell QTP to properly initialize such columns to NULL. The default is INITIALIZE NULLS OFF to remain consistent with the operation of previous versions of QTP.

Equivalent Program Parameter

initnulls|noinitnulls

INTEGER SIZE 6 (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔						

Controls how the products work regarding physical record lengths.

Syntax

INTEGER SIZE 6 ON|OFF

Default: ON (PDL); OFF (PHDPDL).

Discussion

When using PDL in PowerHouse 8.xx, integers of physical SIZE 6 are created for numeric elements with 10-14 digits with INTEGER datatype and no SIZE specified. In 7.10 and PHDPDL, these elements will default to SIZE 8. In a mixed PowerHouse version environment, or when using datafiles created under one version or dictionary type to be used by another, this will cause an incompatibility between dictionaries and physical datafiles. The physical record lengths will not match.

There are two methods to correct this problem. You can either

- specify SIZE for such items, thereby fixing the physical size to match the files, or
- use the INTEGER SIZE 6 resource file statement to control how the products work

For PDL, INTEGER SIZE 6 OFF will cause the item sizes to not create SIZE 6 integers, thus matching PHDPDL and 7.10 created files. For PHDPDL, INTEGER SIZE 6 ON will cause integer SIZE 6 items to be created, thus matching files created in PDL and 8.xx versions.

Equivalent Program Parameter

intsize6|nointsize6

JCWBASE (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
						✔	

Specifies the base value for JCW settings.

Syntax

JCWBASE FATAL|WARN

Discussion

The JCWBASE resource file statement specifies the base value for QUIZ JCW settings. For more information on JCW settings, see "QUIZ Error Status Settings (MPE/iX, UNIX, Windows)" (p. 22). If the base value is fatal, an error can cause a job to stop.

Equivalent Program Parameter

jcwbase=fatal|warn

LIST

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔		✔	✔

Establishes whether or not the PowerHouse component displays the source statement file.

Limit: These parameters apply to source statement files only.

Syntax

```
LIST GENERATE|LAYOUT|PROCEDURES|SQL|TRANSACTION|
    USE [GENERATE|LAYOUT|PROCEDURES|SQL|
    TRANSACTION|USE]... ON|OFF
```

GENERATE

Controls the listing of the results of the GENERATE statement.

LAYOUT

Controls the listing of the sample screen layout.

PROCEDURES

Controls the listing of the generated procedural code.

SQL

Controls the listing of SQL statements. It shows the SQL requests sent from PowerHouse to the database, including the effects of any substitutions.

TRANSACTION

Displays the transaction model used by the screen. SET LIST TRANSACTION also displays all transactions defined in a screen and gives all file/transaction associations.

USE

Controls the listing of source statements contained in USE files.

ON

Lists the specified options.

OFF

Does not list the specified options.

Discussion

The LIST resource file statement states that the contents of source statement files are to be listed as they are read.

The LIST resource file statement establishes the default list option for the USE and REVISE statements without LIST or NOLIST options.

Equivalent Program Parameter

list|nolist

LOCATION MODULE (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔	✔	✔	

Specifies the location and compiles a PowerHouse program into an installable module for ALLBASE/SQL.

Syntax

LOCATION MODULE filelocation

filelocation

The specification for a group.

Equivalent Program Parameters

moduleloc=filelocation

LOCATION PROCESS

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Causes PowerHouse to search for process files in a location other than the current directory (**OpenVMS, UNIX, Windows**) or group/account (**MPE/iX**).

Syntax

LOCATION PROCESS filelocation

filelocation

OpenVMS, UNIX, Windows:	filespec without the filename or extension If a logical name is used, you must include the colon (:) at the end if you specify a full file location. (**OpenVMS**)
MPE/iX:	group.account

Discussion

The LOCATION PROCESS resource file statement is applied to filenames that are unqualified (**MPE/iX, OpenVMS**) or not fully qualified (**UNIX, Windows**) that are specified

- in the **auto** program parameter
- through the appropriate designated files
- in the EXECUTE, SUBSCREEN (and RUN SCREEN verb), and USE statements

The LOCATION PROCESS resource file statement is not applied when locating the dictionary or data files.

The BUILD and SAVE statements save files in the current working directory rather than in the directory specified by LOCATION PROCESS statement.

Equivalent Program Parameters

procloc=filelocation

LOCKWORD (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔			✔		

Enables the use of internal routines to prevent duplicate lockword prompting during screen loading and running.

Syntax

LOCKWORD PROMPT ONCE

Discussion

By default, if a compiled screen has a a file level password, LOCKWORD, the user is prompted for this lockword every time the screen is run. If the screen has to be loaded and then run, the user is prompted for the lockword twice.

The LOCKWORD resource file statement enables internal routines to prevent duplicate prompting. However, use of the statement may cause a performance decrease so it should be tested to determine the effect.

Equivalent Program Parameter

lockword

NOBLOBS

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔		✔	

Specifies whether blobs columns are processed.

Syntax

NOBLOBS

Discussion

By default, QUIZ, QTP, and QDESIGN refer to blob columns. However, blobs are restricted in use; blobs cannot be stored in subfiles, sorted on, or written into an intermediate file in QTP. All of these actions produce an error. By specifying the NOBLOBS statement, blob columns are not processed (ignored) and these errors are avoided.

Equivalent Program Parameter

noblobs

NONPORTABLE

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔		✔	✔

Sets the message severity for nonportable syntax.

Limit: Applies to source statement files only.

Syntax

NONPORTABLE ERROR|NOWARNING|WARNING

Default: NOWARNING

ERROR

Rejects nonportable syntax and issues an error message.

NOWARNING

Suppresses the warning messages.

WARNING

Issues a warning, though processing continues.

Discussion

Not all PowerHouse syntax applies to all computer systems. The NONPORTABLE statement specifies what happens when such syntax is encountered.

Equivalent Program Parameter

nonportable=error|nowarn|warn

NOOWNER

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Prevents the owner name from being attached to the table name in generated code so that different users can use the same compiled screen to access their own tables (having the same name).

Syntax

NOOWNER

Equivalent Program Parameter

noowner

NOSET WARN STATUS (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Suppresses the warning status when a warning condition is detected in PowerHouse.

Syntax

NOSET WARN STATUS

Discussion

In versions prior to 8.40, a warning condition in PowerHouse would return a $WARNING level indication to the operating system. When Module Management System (MMS) sees a result code that is not $SUCCESS, processing stops.

If this is not the desired behavior, specify the **nosetwarnstatus** program parameter to turn off the warning status. If the **nosetwarnstatus** program parameter is used, a status code of $SUCCESS is returned to operating system for PowerHouse warning conditions, instead of $WARNING.

The default behavior is that $WARNING status is used.

Equivalent Program Parameter

nosetwarnstatus

OBSOLETE

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
	✔	✔	✔	✔	✔	✔	✔

Sets the severity of the messages when obsolete syntax is encountered.

Limit: Applies to source statement files only.

Syntax

OBSOLETE ERROR|IGNORE|NOWARNING|WARNING

Default: WARNING

ERROR

Rejects obsolete syntax when encountered and issues an error message.

IGNORE

Ignores all obsolete keywords as part of the syntax (they are treated as entity names).

NOWARNING

Suppresses the warning messages.

WARNING

Issues a warning, though processing continues.

Discussion

As PowerHouse matures, some syntax may be marked for obsolescence and may not be supported in future releases. The OBSOLETE statement specifies what happens when such syntax is encountered.

Equivalent Program Parameter

obsolete=error|ignore|nowarn|warn

OMNIDEX (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	✔

Determines whether OMNIDEX indexes are seen by the PowerHouse component being run.

Syntax

OMNIDEX ON|OFF|QUIZ

ON

OMNIDEX indexes are visible to all products.

OFF

OMNIDEX indexes are not seen by any products.

QUIZ

OMNIDEX indexes are only seen by QUIZ. This gives the same results as PowerHouse versions from 7.29 to 8.29.

Default: ON

Discussion

The program parameter or resource file statement is only required at parse time.

In PowerHouse versions from 7.29 to 8.29, OMNIDEX indexes could be declared in the dictionary, but were only used by QUIZ (and reported by QSHOW). QTP, QDESIGN and QUICK had no knowledge of these indexes. DISC, the developers of OMNIDEX, provided tools to access OMNIDEX indexes from these versions of QUICK.

In version 8.39, OMNIDEX index support was added to QTP, QDESIGN, and QUICK. This changed some default operations. Because these indexes are now visible, they are used in the default linking rules. This can result in different default linkages. Linkages that were specifically coded remain unchanged. The addition of this support to QUICK prevents the DISC tools from functioning.

Version	Default use of OMNIDEX indexes
7.29 to 8.29	QUIZ
8.39 and above	QUIZ, QTP, QDESIGN, QUICK

With the appropriate program parameter or resource file statement, you can determine how you want OMNIDEX indexes to be used.

Equivalent Program Parameter

omnidex|noomnidex

OSACCESS

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Determines whether operating system command entry and execution is allowed.

Syntax

OSACCESS ON|OFF

Default: OSACCESS ON (except for QUICK)

ON

Allows access to the operating system from within PowerHouse components.

OFF

Denies access to the operating system from within PowerHouse components.

Discussion

Allows or denies access to the operating system from within PowerHouse components.

By default, when QUICK encounters the system prompt character, it ignores it and issues the message

```
"Operating system access has been disabled."
```

When you call QUICK directly from QDESIGN, QUICK is invoked with either OSACCESS=ON or OSACCESS=OFF in effect, depending on which statement is set in QDESIGN.

MPE/iX

The system prompt character is a colon (:). A colon followed by a system command causes execution of that command by the operating system.

When the OSACCESS OFF statement is used, any command preceded by a colon is ignored.

OpenVMS

The system prompt character is a dollar sign ($). Entering a dollar sign followed by a system command causes execution of that command by the operating system.

UNIX

The system prompt character is an exclamation mark (!). Entering !<shell_abbreviation> (for example, !csh) opens a new shell, while entering the exclamation mark followed by a system command causes execution of that command by the operating system.

When the OSACCESS OFF statement is used, any command preceded by an exclamation mark is ignored. Although OSACCESS OFF denies users access to the operating system from within PowerHouse, it does not prevent users from suspending the PowerHouse process (usually by pressing [Ctrl-Z]), unless this has been disabled with stty(1) commands.

Windows

The system prompt character is an exclamation mark (!). Entering an exclamation mark followed by a system command causes execution of that command by the operating system.

Equivalent Program Parameter

osaccess|noosaccess

OpenVMS: dcl|nodcl

OWNER

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	

Specifies the owner for tables in an ALLBASE/SQL, DB2, SYBASE or ORACLE database when none is explicitly indicated. Also specifies the default owner of modules created by PowerHouse in ALLBASE/SQL.

Syntax

OWNER owner

owner

Owner may be the database owner or a string.

Owner may be an system variable or file equation (**MPE/iX**), logical (**OpenVMS**) or environment variable (**UNIX, Windows**) which must be enclosed in quotation marks.

Discussion

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.

The resource file statement only applies at parse time and cannot be used at execution time to provide access to specific data.

Equivalent Program Parameter

owner=ownername

PREFIX ORACLE OPEN NAME

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	

Suppresses the addition of "ORACLE@" to the open name for an ORACLE database.

Syntax

PREFIX ORACLE OPEN NAME ON|OFF

Default: ON

ON

Suppresses the addition of "ORACLE@" to the open name.

OFF

Does not suppress the addition of "ORACLE@" to the open name.

Discussion

Valid open names must begin with "ORACLE@". If the open name supplied does not begin with this string, PowerHouse normally inserts it. This statement suppresses the insertion thus allowing logical names (**OpenVMS**) and environment variables (**UNIX, Windows**) to used as the open name. If PREFIX ORACLE OPEN NAME OFF is used, the user must supply the "ORACLE@" in the string assigned.

Equivalent Program Parameter

noprefix_openname

PROMPT

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Specifies the prompt string for the PowerHouse component.

Syntax

PROMPT string
Default: >

Discussion

PowerHouse adds a trailing space after the prompt string.

Equivalent Program Parameter

prompt=string

QUIT

Ends the PowerHouse resource file.

Syntax

QUIT

Discussion

The QUIT statement ends the PowerHouse resource file. Any statements after the QUIT statement are ignored.

The QUIT and EXIT statements are interchangeable.

RESET BIND VARIABLES

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔	✔	✔	

Determines if SQL bind variables are reset for each SQL statement.

Syntax

RESET BIND VARIABLES ON|OFF

Default: RESET BIND VARIABLES ON

Discussion

A bind variable is a placeholder in SQL generated at compile time where a value is substituted at execution time. For example, if a request value for a Find is needed in generated SQL, a bind variable acts as the placeholder in the WHERE clause. Each bind variable has a unique identifier made up of a number and the field name. In versions previous to 8.4xD1, the number was incremented from statement to statement even though the field was the same. This meant that generated SQL was different even though the SQL statements themselves were the same. Because the generated SQL was different, it could not be reused by the database.

The **RESET BIND VARIABLES ON** resource file statement specifies that the bind variables are to be reset for each SQL statement. This allows the generated SQL to be identical for identical SQL syntax. The bind variables will be a letter and a number. The letter is S for Select operations, U for update operations, I for insert operations, and D for delete operations. The number is incremented from 1.

Equivalent Program Parameter

resetbindvar|noresetbindvar

RESTORE LINES

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Provides upward compatibility for screens compiled with versions of PowerHouse prior to 6.09 (**MPE/iX**), 6.00 (**OpenVMS**), and 6.03 (**UNIX**). The RESTORE LINES resource file statement changes the default behavior of screen refreshing.

Syntax

RESTORE LINES ON|OFF

Default: OFF (QUICK removes a screen from the display area when you return to a previous screen.)

Discussion

The RESTORE LINES resource file statement changes the default behavior of screen refreshing.

By default, QUICK removes a screen from the display area when you return to a previous screen. This is the expected behavior for pop-up windows. Only the screens currently active in the screen hierarchy are visible. For certain applications, such as heavy data entry applications, you may want to leave information from a lower-level screen in the display area when you return to a previous screen. If the screen background isn't removed, QUICK can avoid rewriting the screen background each time the lower-level screen is invoked. This applies when the screens don't overlap in the display area. With extended terminal memory, QUICK can use two display areas and simply switch between them by adjusting the terminal window.

When you use the RESTORE LINES resource file statement, QUICK uses a line oriented refresh algorithm. When you return to a screen, only the application lines required by that screen are refreshed in terminal memory. Application lines used by lower-level screens required by the higher-level screen aren't altered. The background remains available when the screen is again invoked.

Equivalent Program Parameter

restore=lines

RETAIN MARK

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Determines whether the field mark position in an array is retained.

Syntax

RETAIN MARK ON | OFF

Default: OFF

Discussion

If fieldmarking is used in an array and a new screen load is retrieved, by default the first occurrence is marked even if the mark was originally on another occurrence. RETAINMARK instructs QUICK to retain the original mark occurrence.

Equivalent Program Parameter

retainmark|noretainmark

REUSE SCREEN BUFFERS

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Causes QUICK to reuse or not reuse previously allocated buffers.

Syntax

REUSE SCREEN BUFFERS ON|OFF

Default: OFF

ON

Causes QUICK to reuse previously allocated buffers.

OFF

Causes QUICK to not reuse previously allocated buffers.

Discussion

The REUSE SCREEN BUFFERS resource file statement causes QUICK to reuse previously allocated buffers when the user moves repeatedly back and forth from a screen to a subscreen. Since buffers don't have to be re-allocated, performance may improve.

Applications using RDB/VMS (i.e. native access) and non-relational files should see a performance improvement by reusing screen buffers. For databases accessed through SQL, reusing screen buffers may cause more memory usage since memory is not cleaned up immediately when a screen exits.

Axiant thin-client applications should not reuse screen buffers.

This feature conflicts with the AX_SCREEN_TUNING environment variable. They should not be used together.

Equivalent Program Parameter

reuse_screen_buffers|noreuse_screen_buffers

RMS FAST READ (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔	✔	✔	

Performs a block read to sequentially accessed read-only files.

Syntax

RMS FAST READ ON|OFF

Default: OFF

Discussion

By using this resource file statement, normal record I/O is much faster. However, it has a number of restrictions:

- The records must be fixed length.
- The access must be sequential, not indexed.
- The data and indexed portions of the file cannot be compressed.
- Files must have compression turned off. By default, PowerHouse creates files with compression turned on, so this must be manually changed using the FDL editor and the file recreated in order to access a PowerHouse file with this statement. If these conditions are not met, you do not receive a warning but PowerHouse does not set up the file for fast reads.

Equivalent Program Parameter

fastread

RMS FILE BASE (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	

Allows the use of zero-based record numbers for cross-platform compatibility.

Syntax

RMS FILE BASE ZERO|ONE

Default: ONE

Discussion

In PowerHouse versions 8.00 and 8.10C, direct file access was zero-based from an external view in an attempt to make PowerHouse code more portable between platforms. However, this decision was changed in 8.10.C1 to make the external view of the record number used to access the file match the physical implementation on each platform. This meant that on OpenVMS the record numbers start at 1, not 0. In an effort to maintain upgrade paths, and for those who really want cross-platform transparency in this area, a resource file statement was added to allow for the use of zero-based record numbers. From 8.10.C1 through to 8.20.D4, the one-based access applied to READs, but not to WRITEs. This was incorrect and has been changed as of 8.20D6 and 8.30 so that it applies to both READ and WRITE.

Equivalent Program Parameter

direct_file_base_zero

SET

Determines whether the PowerHouse resource file is listed before PowerHouse components are initiated.

Syntax

SET LIST|NOLIST

LIST

Lists the PowerHouse resource file.

NOLIST

Does not list the PowerHouse resource file.

Discussion

SET LIST lists the PowerHouse resource file before PowerHouse components are initiated. This is handy for debugging resource files because an error produces the appropriate expected list.

SETJOBSHOW (Windows)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔		✔	

Restricts the files and items listed by the SHOW FILES statement to those for which you have at least read access.

Syntax

SETJOBSHOW ON|OFF

Default: OFF

ON

Shows the SET JOB spawned process window.

OFF

Hides the SET JOB spawned process window.

Discussion

With the SET JOB statement, the job is submitted as a separate spawned process just before the product exits. By default, the spawned process window is hidden. To show the window, use the **SETJOBSHOW ON** resource file statement.

Equivalent Program Parameter

setjobshow|nosetjobshow

SHIFT

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔		✔	✔	✔	✔	✔	✔

Determines how PowerHouse identifiers (such as item names) are shifted.

Syntax

SHIFT DOWN|NONE|UP

Default: UP

DOWN

Shifts the names of entered identifiers to lowercase.

NONE

Leaves the case of identifiers as entered.

UP

Shifts the names of entered identifiers to uppercase.

Discussion

This option facilitates the use of dictionaries and databases with case-sensitive entity names.

By default, PowerHouse upshifts all components of table names. PowerHouse permits access to case-sensitive names by means of this statement or the NOSHIFT, UPSHIFT, and DOWNSHIFT options of the SET statement. If SHIFT NONE or SET NOSHIFT is specified, all PowerHouse identifiers are taken as they appear in the source text instead of being upshifted. For system-wide access to mixed, lowercase, or uppercase identifiers, you can specify the SHIFT option in the SYSTEM OPTION statement.

Equivalent Program Parameter

downshift|upshift|noshift

STATISTICS

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔		✔	✔

Specifies whether or not to display statistics.

Syntax

STATISTICS ON|OFF

Default: ON (QUIZ, QTP); OFF (QUTIL).

ON

Specifies that statistics or detailed file information are displayed.

OFF

Specifies that statistics or detailed file information are not displayed.

Discussion

In QUIZ and QTP, the STATISTICS ON resource file statement displays statistics at the end of a report or run; STATISTICS OFF does not. Stipulating SET STATISTICS, SET NOSTATISTICS, or SET DEFAULT (QUIZ and QTP) resets the statistics control.

In QUTIL, the STATISTICS ON resource file statement produces a report of details of the file being created, such as record size, indexes, and physical location.

Limit: The STATISTICS resource file statement for QUTIL is only valid for OpenVMS, UNIX, and Windows.

Equivalent Program Parameter

statistics|nostatistics

STORE MODULES

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔		✔	

Prevents SQL modules from being compiled and stored in the database. This resource file statement is used at parse time.

Syntax

STORE MODULES ON|OFF

Default: ON

Discussion

This resource file can decrease memory problems associated with compiled sections since the SQL modules are compiled at run-time. However, this may have an impact on performance.

Equivalent Program Parameter

dont_store_module

SUBDICTIONARY

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔	✔	✔	✔	✔	

Specifies whether subdictionary support is enabled; when the relational subdictionaries are opened by PowerHouse; and when they are searched for unqualified record-structures.

Syntax

SUBDICTIONARY
DELAY|DISABLE|ENABLE|NODELAY|NOSEARCH|SEARCH
[DELAY|DISABLE|ENABLE|NODELAY|NOSEARCH|SEARCH]...

DELAY

Indicates that subdictionaries are not to be opened by PowerHouse until referenced by an IN database qualifier, or by the subdictionary search process that looks for unqualified record-structures that do no exist in the data dictionary.

DISABLE

Disables relational subdictionary support. All other SUBDICTIONARY statement options are ignored.

ENABLE

Enables relational subdictionary support.

NODELAY

Indicates that all subdictionaries are to be opened when you enter PowerHouse.

NOSEARCH

Specifies that PowerHouse is not to search for unqualified record-structures in the subdictionaries when they cannot be found in the data dictionary.

SEARCH

Specifies that PowerHouse is to search for unqualified record-structures in the subdictionaries when they cannot be found in the data dictionary.

Discussion

An unqualified record-structure is a record-structure name without the IN database qualifier.

Equivalent Program Parameter

subdictionary=(option[,option]...)

SUBFORMAT n

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔		✔	

Specifies the format of subfiles being created.

Syntax

SUBFORMAT n

The values are:

MPE/iX, UNIX:	0, 1, 3, 5, 6, 7, 8
OpenVMS:	7, 8
Windows:	8

Default: 8 (all platforms)

Discussion

This statement allows you to specify the subfile format to be used at run-time when the format differs from the default.

If a NULL or invalid value is specified for the SUBFORMAT statement, an error is issued.

If the FORMAT option of the SET SUBFILE statement is used, these FORMAT specifications override the SUBFORMAT statement. When neither a FORMAT statement option nor the SUBFORMAT statement is specified, the default subfile format of 8 is used.

MPE/iX

In PowerHouse 6.09, the maximum size of an item name increased from 20 to 31 characters. PowerHouse makes use of file labels when creating subfiles, limiting the amount of information that can be stored to a subfile. Due to the increase in item information being stored to the subfile, the maximum number of items supported decreased from approximately 440 to approximately 310.

Note: This changed in later versions as well. Now, the number of items that can be written to subfiles depends on the size of the item names.

Equivalent Program Parameter

subformat=n

TERMINAL

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔			✔		

Establishes the terminal type.

Syntax

TERMINAL terminal-type

terminal-type

Specifies the terminal type supported by the PowerHouse component.

Discussion

In QDESIGN, the terminal type is passed to QUICK when the GO statement is entered.

Terminal Type When Running QUICK

To communicate properly with the user during a QUICK session, QUICK must know the type of terminal being used. A full list of supported terminal types, and necessary strapping and switch-setting information is provided with the installation package.

QUICK determines the terminal type in one of the following ways:

1. Specification (**MPE/iX, UNIX**)

   If terminal type is specified in the dictionary (on the PORTS option of the SYSTEM OPTIONS statement), QUICK uses that specification.
2. Detection

   Based on terminal characteristics, QUICK can identify most terminals automatically.

   By default, QUICK polls a terminal for one second to try to identify it. If QUICK is unsuccessful, it then prompts the user to supply a terminal type. **MPE/iX:** However, this may not be enough time for terminals on busy systems to respond. The TERMINAL POLLING SPEED statement can be used to lengthen the time QUICK waits for a response when polling a terminal. This parameter is useful only if the terminal in question can be identified by polling.
3. User response

   If the terminal type cannot be identified in steps 1 and 2, QUICK prompts for a predefined terminal type and the QUICK user responds, as in

   ```
   Terminal type=HP2392
   ```

   If the user responds with a question mark (?), a full list of acceptable terminal types is displayed.
4. Override

   To override steps 1, 2, and 3 and allow the user to specify the terminal type directly via the prompt, specify the statement TERMPOLL OFF.
5. TERMINAL resource file statement

The TERMINAL resource file statement can be used in QDESIGN to provide the terminal type to be used by QUICK if the GO statement is entered. The format and allowable entries are the same as those used in QUICK. All previous steps are bypassed if a terminal type is specified.

The suffix ANY instructs QUICK not to trap nonprinting control codes and replace them with a question mark for display. ANY can be used to direct the terminal to an alternative character set. Steps must be taken to ensure that the control-code sequences do not interfere with QUICK's terminal display.

MPE/iX

Some terminals have more than 24 lines of memory. To tell QUICK that more than 24 lines of memory are available, specify the number of lines of memory after the terminal type. The terminal type and lines of memory must be separated by a hyphen, as in

```
TERMINAL HP2624-96
```

This terminal type specification tells QUICK that the terminal is an HP2624 and has 96 lines of memory. (The number of memory lines can also be specified in the dictionary.) If the number of lines of memory is not specified, QUICK assumes 48 lines unless the terminal profile within QUICK contains a lower number.

Line drawing is an option on some terminals; however, QUICK does not assume that line drawing is installed. To tell QUICK that line drawing is installed, specify the alternate character set (A, B, or C) where the line-drawing characters reside. Again, the terminal type and the line-drawing character set are separated by a hyphen, as in

```
TERMINAL HP2624-96-B
```

Although some terminals have more than 24 lines of memory and support highlighting, QUICK may not be able to support these features. To be supported, memory must be continuous, not paged. Many terminals have memory that can be addressed in discrete pages of 24 lines, but require that the page be identified when the user addresses a specific line. QUICK supports lines greater than 24 only if the line number itself can be addressed.

On some terminals, the highlighting control sequences require space on the terminal display. QUICK does not support highlighting of this type because the display would change from one terminal type to another.

Other suffixes can be added to the terminal specification:

- MTS (when running multipoint terminal software)
- X25 or TRANS (when using communications software)
- TAE (for terminals with type-ahead capabilities)
- ANY

When sending data in Block mode over X.25 or TRANSPAC, stacking screens will cause an error. This also applies to sending data in Block mode over MTS.

OpenVMS

PowerHouse supports function keys for the DEC VT family of terminals. On all of these types of terminals, PowerHouse can use the 18 keys on the numeric keypad. On VT 200/VT300-series terminals, PowerHouse can also use the last six of the keys located across the top of the keyboard (the others are all reserved), and the keys labeled FIND, INSERT HERE, REMOVE, SELECT, PREV SCREEN, and NEXT SCREEN.

To take advantage of your terminal's function key capability, use the TERMINAL statement in the resource file when you invoke PowerHouse. The syntax is

```
TERMINAL termtype
```

The TERMINAL statement is one of VT100, VT200, VT300 or VT400 with an optional suffix of -KEY or -NOKEY. For example, you might enter

```
TERMINAL VT400-KEY
```

PowerHouse treats certain keys differently depending on which suffix you use:

- If you use -NOKEY, or do not use a suffix at all, PowerHouse does not recognize most of the numeric keypad keys as function keys, but instead treats them as numeric keys. The exceptions are [GOLD], [PF2], [PF3], and [PF4], which PowerHouse always treats as function keys when you use the TERM option. You may want to use TERMINAL VT400-NOKEY if you have a VT200-series terminal and you want to use the numeric keypad as a numeric keypad but still be able to use the other function keys.
- If you use -KEY, PowerHouse treats all the numeric keypad keys as function keys.

You can also use the TERMINAL statement to specify the maximum number of lines the terminal memory can contain (up to a maximum of 240). The number of lines of terminal memory specified should be evenly divisible by 24. The number of lines that QUICK actually uses depends on the stacking and windowing options that you use, which may or may not make use of available application lines. Regardless of the number of terminal memory lines specified, QUICK never uses more than the application lines specified in QKGO.

Windows

The only terminal type available is WINDOWS-24.

Equivalent Program Parameter

term=terminal-type[-terminal-parameter]...

TERMINAL BLOCKMODE (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Determines whether or not QUICK recognizes BLOCK TRANSFER control structures in Block mode.

Syntax

TERMINAL BLOCKMODE COMPATIBLE|PANEL

Default: COMPATIBLE

COMPATIBLE

Runs in standard HP Block mode.

PANEL

Runs blockmode terminals in Panel mode.

Equivalent Program Parameter

blockmode=compatible|panel

TERMINAL CHARACTERMODE

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Specifies whether or not QUICK recognizes BLOCK TRANSFER control structures in Character mode.

Syntax

TERMINAL CHARACTERMODE FIELD|PANEL

FIELD

Indicates that BLOCK TRANSFER control structures in a screen are ignored.

PANEL

Indicates that BLOCK TRANSFER control structures in a screen are recognized.

Discussion

If neither the TERMINAL CHARACTERMODE FIELD statement nor the **charmode** program parameter are specified, and BLOCK TRANSFER control structures are included in screen design, the default interface is Panel mode.

Equivalent Program Parameter

charmode=field|panel

TERMINAL CONFIRMER

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Changes confirmation messages to pop-up windows with OK/CANCEL buttons.

Syntax

TERMINAL CONFIRMER

Equivalent Program Parameter

confirmer

TERMINAL POLLING SPEED (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Sets the amount of time (n) that QUICK takes when polling the terminal to determine a terminal type before prompting the user.

Syntax

TERMINAL POLLING SPEED n

Limit: The range of POLLSPEED is 1 to 20 seconds.

Default: 1 second

Equivalent Program Parameter

pollspeed=n

TERMINAL READ (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Determines how QUICK uses single-character reads.

Syntax

TERMINAL READ CHARACTER|LINE

CHARACTER

Causes QUICK to run the application with the use of the single character processing.

LINE

Causes QUICK to run the application without the use of single character processing.

Equivalent Program Parameter

read=char|line

TERMPOLL (MPE/iX, OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

Determines whether to poll the terminal to establish the terminal type.

Syntax

TERMPOLL ON|OFF
Default: ON

ON

Specifies that the terminal is polled.

OFF

Specifies that the terminal is not polled.

Equivalent Program Parameter

termpoll|notermpoll

TIC RESOURCE FILE (UNIX, Windows)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
					✔		

References the QUICK Terminal Interface Configuration (TIC) resource file.

Syntax

TIC RESOURCE FILE filespec

filespec

The specification for a file, as it is identified to the operating system. It is the name of file or environment variable, and is case-sensitive.

A file specification takes the general form

UNIX: /[directory/]...filename.extension
WINDOWS: [drive:\][directory\]...filename.extension

Discussion

The TIC RESOURCE FILE statement references the flat TIC resource file. By default, the location of this file is:

$PH_QKGO_LOCATION/resource

TIME ZONE (MPE/iX)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔		✔	✔	✔	✔	✔	✔

Determines the mechanism that PowerHouse uses to obtain the values of SYSDATE and SYSTIME.

Syntax

TIME ZONE ON | OFF

Default: OFF

ON

Specifies that PowerHouse takes the value of the TZ system variable into account when it returns SYSDATE and SYSTIME values. With the use of **TIME ZONE ON**, the value of SYSTIME is only accurate to the nearest second.

OFF

Specifies that PowerHouse does not take the value of the TZ system variable into account when it returns SYSDATE and SYSTIME values. With **TIME ZONE OFF**, the value of SYSTIME is accurate to the nearest tenth of a second.

Discussion

By default, PowerHouse returns values that are based on the MPE System Time. This time is the same for all users of the system and normally reflects the correct time where the computer is located. These results are not affected by the value of the TZ system variable.

If the **timezone** program parameter or the TIME ZONE ON resource file statement is specified, then PowerHouse returns values that take the value of the TZ system variable into account. Multiple users can see different values for SYSDATE and SYSTIME by setting TZ to different values within their sessions. The TZ variable would typically be set to reflect the location where the end user is working if this is in a different time zone than where the computer is located.

For information about the TZ variable and the values that can be assigned to it, see your Hewlett Packard documentation.

Equivalent Program Parameter

timezone|notimetimezone

TPI (MPE/iX, HP-UX, Windows)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔		✔	✔	✔	

Determines whether or not TPI or OMNIDEX indexes are seen by the PowerHouse component being run.

Syntax

TPI ON|OFF|QUIZ

Default: ON

Discussion

The new resource file statement, TPI ON|OFF|QUIZ, provides the same functionality as the existing resource file statement OMNIDEX ON|OFF|QUIZ.

The extra keywords have been added to reflect the fact that support for TPI functionality is independent of Omnidex.

Both the existing and new keywords are available on MPE/iX, HP-UX and Windows.

Equivalent Program Parameter

tpi|notpi

TRUNCATE PARM VALUES

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
				✔		✔	

Strips or does not strip trailing blanks from PARM values.

Syntax

TRUNCATE PARM VALUES ON|OFF

Default: OFF for values entered interactively; ON for values read from a file, whether in a batch job or a file specified in the **parmfile** program parameter.

ON

Specifies that trailing blanks are stripped from any parm values.

OFF

Specifies that trailing blanks are significant and are not stripped from parm values.

Equivalent Program Parameter

parmprompt=truncate|notruncate

TRUSTED (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Activates or deactivates C2-level security for the execution of RUN commands and DCL commands within components.

Syntax

TRUSTED ON|OFF

Default: ON

ON

Indicates that OpenVMS should trust that this is a "well-behaved" application and allow DCL access.

OFF

Indicates that OpenVMS should not trust this application and not allow DCL access.

Discussion

The security level on captive accounts was increased under OpenVMS 6.1 to prevent CAPTIVE users from spawning. This security also prevents users from executing a RUN COMMAND statement from a PowerHouse application running on a system that has C2-level security. To work around this, the System Manager changed a SYSGEN parameter to disable the new security feature.

The DCL options SPAWN|TRUSTED|NOTRUSTED allow CAPTIVE accounts to spawn. By making use of the TRUSTED flag in PowerHouse, CAPTIVE accounts can spawn, regardless of the SYSGEN setting.

By default, all DCL commands done from PowerHouse are done with TRUSTED (the pre-6.1 default setting). For all components, the use of the **nodcl** program parameter will disable DCL. In QUICK, you are still able to program RUN COMMAND statements to execute DCL commands.

The TRUSTED OFF resource file statement disables RUN COMMANDs from QUICK for CAPTIVE users. You can use the TRUSTED OFF resource file statement in each of the components to disable all forms of DCL access for captive users.

TRUSTED CAPTIVE users can spawn. NOTRUSTED CAPTIVE users cannot spawn.

NON-CAPTIVE users are unaffected by the use of this resource file statement. They are always TRUSTED by OpenVMS.

The **dcl** or **osaccess** program parameter and TRUSTED ON resource file statement may be combined with the following results in CAPTIVE accounts.

PowerHouse spawns whenever it needs to access DCL, such as when interactive DCL is requested by the user, when a QUICK RUN COMMAND is executed, or a GO is requested from QDESIGN.

OSACCESS or DCL	TRUSTED ON	Interactive DCL Allowed	RUN COMMAND Allowed
N	N	N	N
N	Y	N	Y
Y	N	N	N

OSACCESS or DCL	TRUSTED ON	Interactive DCL Allowed	RUN COMMAND Allowed
Y	Y	Y	Y

Equivalent Program Parameter

trusted|notrusted

UIC BRACKETS (OpenVMS)

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	

Causes the UIC function to not return brackets around the result values. It also causes internal security checking to match the existence of brackets based on the setting of the statement.

Syntax

UIC BRACKETS ON|OFF

Default: ON

ON

The UIC function returns brackets around the result values.

OFF

The UIC function does not return brackets around the result values.

Discussion

This resource file statement is provided for compatibility with the UNIX uic function and previous versions of PowerHouse 8.xx.

Equivalent Program Parameter

nouicbrackets

UPDATE ORDER

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
		✔					

Controls the way PUT verbs are generated in the UPDATE procedure.

Syntax

UPDATE ORDER BOTTOM UP

or

UPDATE ORDER TOP DOWN (OpenVMS)

or

UPDATE ORDER FOREIGN KEY CONSTRAINT

Default: UPDATE ORDER BOTTOM UP

BOTTOM UP

By default, the ordering of PUT verbs follows the rules described in the UPDATE procedure section of Chapter 7, "QDESIGN Procedures", in the *QDESIGN Reference* book.

OpenVMS: Causes PowerHouse to generate PUT verbs in the same order they are generated in version 8.x.

TOP DOWN (OpenVMS)

Causes PowerHouse to generate PUT verbs in the same order as they are generated in version 7.10.

FOREIGN KEY CONSTRAINT

Causes PowerHouse to generate the order of PUT verbs based on foreign key constraints that may be in effect according to the database definition.

Discussion

For more information about these order methods, see the UPDATE procedure in Chapter 7, "QDESIGN Procedures", of the *QDESIGN Reference* book.

Equivalent Program Parameter

update=bottomup|topdown|fkc_put_order

USE

Calls another PowerHouse resource file and processes its statements.

Syntax

USE filespec [LIST|NOLIST]

filespec

The specification for a file, as it is identified to the operating system. A file specification takes the general form:

MPE/iX:	**[*]filename[/lockword][.group[.account]]**
OpenVMS:	**[device:][directory] filename[.extension][;<version>]** Square brackets are required around a directory name.
UNIX:	**/[directory/]...filename.extension**
Windows:	**[drive:\][directory\]...filename.extension**

LIST

Lists the called PowerHouse resource file.

NOLIST

Does not list the called PowerHouse resource file.

Discussion

The USE statement allows you to process the statements of another PowerHouse resource file in addition to those in the current PowerHouse resource file.

VMSDATE

PDL	PHDPDL	QDESIGN	QSHOW	QTP	QUICK	QUIZ	QUTIL
✔	✔	✔	✔	✔	✔	✔	✔

Changes how PowerHouse creates and processes VMSDATE items.

Syntax

VMSDATE = CURRENT|OBSOLETE

CURRENT

The default behavior of PowerHouse 7.10 and previous versions is CURRENT.

Applications upgraded from PowerHouse 7.10 to 8.30 and higher should use CURRENT behavior to make their VMSDATE data transparent between versions.

Default: For PowerHouse 8.30 and higher, the default is CURRENT.

OBSOLETE

The default behavior of PowerHouse versions 8.00, 8.10, and 8.20 (prior to 8.20D2) is OBSOLETE. No CURRENT behavior is available.

In PowerHouse versions 8.20D2 and above, both CURRENT or OBSOLETE behavior is available by using the appropriate option of the **vmsdate** program parameter or resource file statement.

To use OBSOLETE behavior in PowerHouse versions 8.20D2 or above, use the **vmsdate=obsolete** program parameter on all RUN commands or use the VMSDATE OBSOLETE resource statement in a Resource File.

Discussion

VMSDATE items have different internal formats, depending on what version of PowerHouse created them. PowerHouse versions 7.10 depends on the incoming or outgoing system date being in the form of a fractional day. PowerHouse versions 8.xx use clock time (HHMMSSTTT).

VMSDATE items were not being converted properly in some versions. As a result, these versions cannot correctly read and report VMSDATE items created in other versions of PowerHouse; nor can they correctly read or report VMSDATE items created outside of PowerHouse. Similarly, PowerHouse versions 7.10 and 8.30 and higher, as well as external applications, cannot read PowerHouse 8.00, 8.10, or 8.20 VMSDATE items correctly. VMSDATE=OBSOLETE|CURRENT, was added to PowerHouse 8.20D2 and later to resolve this problem.

PowerHouse 8.00, 8.10, and 8.20 process VMSDATEs using the VMSDATE=OBSOLETE behavior. PowerHouse 8.30 and higher defaults to VMSDATE=CURRENT behavior. PowerHouse 7.10 defaults to external OpenVMS behavior, which is the same as VMSDATE=CURRENT.

For detailed information about processing VMSDATEs and conversion issues, see the *PowerHouse 4GL Version 8.30 for OpenVMS Upgrade Guide*.

Equivalent Program Parameter

vmsdate=current|obsolete

Chapter 4: Messages in PowerHouse

Overview

PowerHouse provides the following types of messages:

- PowerHouse 4GL messages
- Service Layer messages
- Designer Messages for your application (QUICK, QUIZ, QTP)

For both the PowerHouse 4GL and the service layer you can create your own alternative message file to replace some or all of the standard messages, or use the ready-made files. These ready-made files in English, French, and German for PowerHouse 4GL and the service layer are included with PowerHouse.

PowerHouse 4GL Messages

Each PowerHouse 4GL component has internal messages built into it. As well, each component has a designated message file which you can use to override these internal messages. These alternative message files allow you to specify messages in a language other than English, as well as tailor messages to meet your needs.

If you specify a designated file for a given component, PowerHouse uses the messages contained in the file. If a message does not exist in the file, PowerHouse will use the internal message.

The following table lists the designated files for each PowerHouse 4GL component:

Designated Message File[1]	OpenVMS Logical or UNIX/Windows Environment Variable	Component
pdlmsg	PDLMSG	PDL
phdpdlmsg (**OpenVMS**)	PHDPDLMSG	PHDPDL
qkdmsg	QKDMSG	QDESIGN
qkmsg	QKMSG	QUICK
qshomsg	QSHOMSG	QSHOW
qtpmsg	QTPMSG	QTP
quizmsg	QUIZMSG	QUIZ
qutlmsg	QUTLMSG	QUTIL

[1]The file extension for **OpenVMS, UNIX** and **Windows** message files is .txt.

Message Explanation Files

Message explanation files contain detailed explanations in English of the standard messages issued by the PowerHouse component.

The following table lists the message explanation files for each PowerHouse 4GL component:

Message Explanation File[1]	Component
pdlinfo	PDL
phdpdlinfo (**OpenVMS**)	PHDPDL
qkdinfo	QDESIGN
qkinfo	QUICK
qshoinfo	QSHOW
qtpinfo	QTP
quizinfo	QUIZ
qutlinfo	QUTIL

[1]The file extension for **OpenVMS, UNIX** and **Windows** message files is .txt.

Using Alternative Message Files (MPE/iX, OpenVMS, UNIX)

In order for PowerHouse to find your alternative message files, you must either define the corresponding file equation (**MPE/iX**), logical (**OpenVMS**), or environment variable (**UNIX**) or rename the alternative message file to the name PowerHouse expects.

For example, to use the German alternative message file for QUIZ and QTP, enter:

MPE/iX:	`FILE QUIZMSG=QUIZMSGD.PH729C.COGNOS` `FILE QTPMSG=QTPMSGD.PH729C.COGNOS`

OpenVMS:	`DEFINE QKDMSG PH_DOC_LOCATION:QKDMSGD.TXT`
	`DEFINE PHMSG PH_DOC_LOCATION:PHMSGD.TXT`
	or
	`SETPOWERHOUSE LANG=D`

UNIX:	`setenv PH_HELP_LOCATION $PH_USR/msg/deutsch`
	`setenv QUIZMSG $PH_HELP_LOCATION/quizmsg.txt`
	`setenv QTPMSG $PH_HELP_LOCATION/qtpmsg.txt`

When using alternative message files, all the file equations (**MPE/iX**), logicals (**OpenVMS**), or environment variables (**UNIX**) for the components you use must be set.

Using Alternative Message Files (Windows)

PowerHouse 4GL for Windows provides two sets of default message files in separate folder locations within the installation location. One set of message files is used when the display is in a GUI window while the other set is used when the display is in a Command Prompt window. Messages that use the extended character set do not display correctly unless they are based on the appropriate code page for either a GUI window or a Command Prompt window.

The folder for the message files for GUI windows is axmsg. The folder for the message files for the Command Prompt windows is phmsg.

For example, messages for QUIZ reports and QTP Runs should use the phsmg folder location because they are run in a Command prompt window. If you use QKView or Axiant thin-client applications, you should use the QUICK message files in the axmsg folder.

The files that supply the default internal messages are phmsgf.dll, phmsgd.dll, phmsge.dll. These files cannot be edited. Phmsgf.dll supplies messages in French. Phmsgd.dll supplies messages in German. Phmsge.dll supplies messages in English. By default, messages are supplied in English. Specifying an alternative message file overrides the default messages, unless the message line is blank.

The files from the different folders appear differently if they are edited in a GUI editor such as Notepad. The files from the axmsg folder appear correctly but the files from the phmsg folder appear incorrectly. To edit the files from the phmsg folder, use the edit command from a Command Prompt window.

To use the default French or German alternative message file without making any changes, use the following procedure:

1. In Windows Explorer, in the PowerHouse 4GL installation location, locate and open the appropriate message folder, axmsg or phmsg.
2. Create a copy of the alternative message file you want to supply your messages, for example, phmsgd.dll.
3. Rename the copy to phmsgm.dll. Phmsgm.dll is the designated file that PowerHouse 4GL uses to supply messages.
4. Move the renamed file to the root installation folder location. This will overwrite the existing phmsgm.dll that, by default, contains the default English messages.

To use the text-based alternative message files, you must define the corresponding environment variable or rename the alternative message file to the name PowerHouse expects.

For example, to use the German alternative message file for QUIZ and QTP, enter:

Windows:

```
set PH_HELP_LOCATION="C:\Program Files\Cognos\PowerHouse 4GL
8.41E\phmsg"
set QUIZMSG=%PH_HELP_LOCATION%\quizmsg.txt
set QTPMSG=%PH_HELP_LOCATION%\qtpmsg.txt
```

When using alternative message files, all environment variables for the components you use must be set.

Format of a PowerHouse 4GL Message File

A PowerHouse 4GL message file is an order-dependent text file. PowerHouse 4GL uses the record number to retrieve a message. A message may contain one or more message substitution characters, by default, a caret (^), which PowerHouse uses to insert additional text.

When modifying a message or translating a message to another language, you may want to alter the order of substitution. To do this, see (p. 271).

The following is an example of a PowerHouse 4GL message file:

```
•
•
•
No valid files were specified in the GENERATE statement.
Some of the files specified in the GENERATE statement were invalid.
The file ^ has no associated records.
Use SET GENERATE DEVICE DISC NAME <name> to create a new output file.
A language must be specified to execute the SET GENERATE DEVICE statement.
Cannot perform GENERATE without an output device specified.
•
•
•
```

Message Template Files

PowerHouse 4GL standard messages are in English. Message template files are provided in English, French, and German. Message template files provide all the messages in a text file. The message template files can be used as is for an alternative message file, as a basis for translation into other languages, or as a basis for other customization.

On **MPE/iX**, the message template files are installed in the PowerHouse install location.

On **OpenVMS**, the message template files are installed in a MESSAGE directory under the main install location. The message location is available in the logical PH_MESSAGE_LOCATION.

On **UNIX**, the message template files are installed in individual language directories named english, francais, and deutsch in a directory named msg in the main install directory. Because the message template files for different languages are not in the same directory, they are named the same for each language. The PH_HELP_LOCATION environment variable is set to one of the message locations based on the language.

On **Windows**, the message template files are installed into the PowerHouse install location in two different directories, axmsg and phmsg. The two sets of message template files use different code pages to account for display differences in the extended character set, depending on where the display occurs. The message template files in phmsg should be used when a PowerHouse component is run from the command prompt. The message template files in axmsg should be used when using QKView or Axiant in thin-client mode. Note that when a screen running in QKView runs QUIZ or QTP, the report or run executes in a command prompt and therefore QUIZ and QTP should use the message template files from the phmsg directory.

Creating or Modifying an Alternative Message File

To create an alternative message file, follow these steps:

1. Make a copy of a ready-made alternative message file, such as the message template file, or create your own file of the same type. For example, to copy a file, enter:

MPE/iX:	`COPY PDLMSGE.PH839C.COGNOS,MYPDLMSG`
OpenVMS:	`COPY PH_MESSAGE_LOCATION:PDLMSG.TXT MYMSGFILE.TXT`
UNIX:	`cp pdlmsg.txt mymsgfile.txt`
Windows:	`copy pdlmsg.txt mymsgfile.txt`

2. Using the editor, make the modifications to each message that you want to change. To change only some of the default PowerHouse 4GL messages, leave the messages you don't want to change blank. PowerHouse substitutes the default messages for those messages you haven't defined.
3. Save the changes you have made.
4. **MPE/iX:** Set a file equation to reference this alternative message file. For example

   ```
   FILE PDLMSGE=PDLMSG
   ```

   OpenVMS: Convert the message file to fixed length format using the following command:

   ```
   $convert/fdl=ph_message_location:msg.fdl/PAD=%D32 mymsgfile.txt
   PH_MESSAGE_LOCATION:mymsgfile.txt
   ```

   Modify the logical to reference this alternative message file. For example

   ```
   DEFINE QKDMSG PH_MESSAGE_LOCATION:MYMSGFILE.TXT
   ```

   UNIX: Modify the environment variable to reference this alternative message file. For example

   ```
   setenv QUIZMSG "/user/home/mymsgfile.txt"
   ```

   Windows: Modify the environment variable to reference this alternative message file. For example

   ```
   set QUIZMSG="C:\Program Files\Cognos\PowerHouse 4GL
   8.41E\phmsg\mymsgfile.txt"
   ```

File system and other messages are sometimes trapped and reproduced by PowerHouse. These messages cannot be changed using the alternative message files.

Rules for Modifying Default Messages

When you modify an alternative message file, ensure that

- the modified message is of the same file type as the original alternative file
- the relative record numbers of the messages within the file are maintained

- the modified message contains the same number of substitution characters as the original message, if you modify a message that contains substitution characters. The message substitution character, by default, a caret (^), is an indicator that variable text (for example, a file name) is to be inserted into the message at that point. The message substitution character can be redefined in the data dictionary.
- the total length of each message, including inserted text, is less than 78 characters (or 132 if your terminal is set to accommodate that many characters). Messages with inserted text that exceed the allowable number of characters wrap to the next line.
- Mode and Action field labels do not exceed the size shown in QKMSG (**MPE/iX**) or qkmsg.txt (**OpenVMS, UNIX, Windows**)

Service Layer Messages

The service layer provides a set of basic functions such as relational data access, networking, repository, and communications that are available to all PowerHouse and PowerHouse Web components.

There are two types of messages used by the service layer:

- default (or compiled) messages
- alternative messages

The service layer messages are available in English, French, and German.

The messages in the service layer are distinct from other compiled messages in PowerHouse and PowerHouse Web. They must be decompiled in order to get a source file that can be modified, and are then recompiled using the procedure described on (p. 268).

How the Service Layer Locates Message Files

MPE/iX

The message files that contain the English, French, and German versions of the service layer messages are named SRVCMSGE, SRVCMSGF, and SRVCMSGD respectively and are located in the install location. The designated file for service layer messages is SRVCMSGS.

By default, there is a file equation for SRVCMSGS pointing to SRVCMSGE.

In order for the service layer to use an alternative message file, a file equation for SRVCMSGS must be used to point to the language file of choice.

OpenVMS

The message files that contain the English, French, and German versions of the service layer messages are named SRVCMSGE.MSG, SRVCMSGF.MSG, and SRVCMSGD.MSG respectively. The files are located in PH_MESSAGE_LOCATION for PowerHouse and PHWEB_MESSAGE_LOCATION for PowerHouse Web.

The default messages are in PH_MESSAGE_LOCATION:SRVCMSGE.MSG or PHWEB_MESSAGE_LOCATION:SRVCMSGE.MSG and are referenced by the logical name, SRVCMSGS.

In order for the service layer to use an alternative message file, the SRVCMSGS logical must be set to reference it.

UNIX

The message files that contain the English, French, and German versions of the service layer messages are named srvcmsgs.msg and are located in the **english**, **francais**, and **deutsch** subdirectories in the **msg** directory in the install location.

The default messages are in PH_USR/msg/english and are referenced by the SRVCMSGS environment variable.

In order for the service layer to use an alternative message file, the SRVCMSGS environment variable must be set to reference it.

Windows

The message files that contain the English, French, and German versions of the service layer messages are named srvcmsg_en.msg, srvcmsg_fr.msg, and srvcmsg_de.msg respectively. The files are located in the install location.

By default, the SRVCMSGS entry in the powerhouse.ini and phweb.ini files point to srvcmsg_en.msg.

In order for the service layer to use an alternative message file, the SRVCMSGS entry in the powerhouse.ini or phweb.ini file, or the SRVCMSGS environment variable must be set to reference it. The ini file entry takes precedence over the environment variable.

Format of a Default Message File

A message file contains blocks of messages for each component. Each block consists of the following required elements in the specified order:

- a COMPONENT statement
- a NAME statement
- a PREFIX statement which is for internal use only

The following optional statements for each block are:

- SEVERITY statements which are for internal use only
- CATEGORY statements which are for internal use only
- PARAMETER statements
- message lines
- comment lines

COMPONENT Statement

There can be only one per component block and it contains just the name in the format:

COMPONENT <name>

NAME Statement

There can be only one per component block and it contains the short name for the component, for example, EXPE for the component EXPENG. The format is

NAME <name>

SEVERITY Statement

The SEVERITY statement is used to define a severity category for use with subsequent CATEGORY statements. It also has the side effect of setting the current message category to the newly defined severity. The SEVERITY statement consists of the SEVERITY keyword followed by

- the symbolic name to be assigned to the severity (that is, error, fatal, warning)
- a numeric severity level (in the range 0 to 15)
- a severity code to use when generating message prefixes (that is, E, F, or W)
- a prefix/noprefix specification which determines if a generated message prefix is to be appended to the text of messages in the indicated severity category. The general syntax is:

SEVERITY <identifier> <number> <severity text> { PREFIX | NOPREFIX}

For example:

```
severity e3 3 E prefix
```

The following severities are defined:

Category	Abbreviation	Meaning
E3	E	error, recovery possible
F6	F	non-recoverable error

Category	Abbreviation	Meaning
H8	H	help text
I1	I	information only
N5	N	communication error
P4	P	parameter to other messages
S9	S	successful operation
W2	W	warning

CATEGORY Statement

The CATEGORY statement is used to indicate the severity category to associate with subsequent message definitions. This association continues until another CATEGORY or SEVERITY statement is reached. Any number of CATEGORY statements are allowed per component/file. The CATEGORY statement consists of the CATEGORY keyword followed by a single token indicating the name of a severity level. Severity levels can be referenced before their definition via the severity command. For example:

```
category e3
```

PARAMETER Statement

For each component, the default substitution character is a caret (^). To specify a different substitution character, use the PARAMETER keyword, followed by a string, as in

```
PARAMETER "%"
```

The PARAMETER keyword may be used as often as desired. Each time it is invoked, the substitution character is changed.

The substitution character defined in a PARAMETER statement may be used within a message as long as it is preceded by a backslash (\), as in

```
EXP1 "This message contains a substitution character (\^)."
```

Use two consecutive backslashes (\\) to include a backslash in a message, as in

```
EXP2 "This message will display a \\ single backslash."
```

A number may follow a parameter marker if it is preceded by a backslash (\), as in

```
EXP3 "This message contains a number after a marker ^\2."
```

A number after the substitution character indicates the order in which parameters are substituted in a message. Parameter numbers start at one.

If no parameter number is supplied, then it is assumed that the parameter number is one. Such a message may contain only a single parameter.

Messages

A message definition is assumed to be in progress when a token is encountered at the beginning of a statement that is not one of: COMPONENT, NAME, PREFIX, CATEGORY, PARAMETER, SEVERITY, or the opening comment delimitor /*. The syntax of a message definition is quite simple: an identifier followed by a quoted string. The identifier is interpreted as the message name while the quoted string is taken to be the text of the message being defined. Double quotes can be embedded into the message string by using a double quote pair. Each message within a component must have a unique name. A line continuation can be achieved by including "\n" in the string, which will force the text following it to be displayed on a new line. If you want to continue the message text over more than one line, place a backslash (\) at the end of each line except the last one. For example:

```
file_error_message "Invalid file: ^1. \nLoad operation aborted"
system_error "The OS reported the following error: ""^1"" while"\ "processing."
```

Comments

Comments may occur anywhere within a message file except within any statement. They must be at the start or end of the statement. Comments are delimited by an opening "/*" and a closing "*/", as in

```
/*
*  This is a comment
*/
```

The following is an example of a service layer message file:

```
COMPONENT EXPENG
NAME EXPE
PREFIX EXPENG
SEVERITY E3 3 E PREFIX
BADTMPDIR "Invalid temporary directory."
NORECSIZE "Record size has not been defined."
BUFFTOOSMALL "Buffer too small."
BADOUTPUT "Invalid output file."
FILEREAD "File read failed."
FILECREATE "Unable to create file."
UNDEFOPER "Undefined operator."
FILESIZE "Unable to obtain the file size."
FILEOPEN "Unable to open file."
TOOMANYARGS "Too many arguments."
FLAGCOMBO "Cannot combine stable sort and delete duplicates"\ "flags."
INVKEYDESC "Invalid key description."
INTCALL_UDF "The internal_call()  function was not specified."
```

Service Message Compiler

Syntax

MPE/iX:	RUN COGMC.<version>.COGNOS;INFO="mode [options]"
OpenVMS, UNIX, Windows:	cogmc mode [options]

Mode

The Message Compiler mode is one of the following:

Mode	Use
-compile	Compiles from source.
-decompile	Decompiles a message file.
-about	Extracts version information from a message file.

Options

The Message Compiler options can be one or more of the following:

Option	Use
-output filename	Sets output filename for all modes.
-language "text"	Sets language descriptor text for -compile mode (for example, "English-Canadian").
-input filename	Sets input filename for -compile, -about and -decompile modes.

Option	Use
-character_set "character set name"	Specifies the character set used in text files for all modes.
-verbose	Forces the generation of verbose output messages for all modes.
-nocase	Forces case-insensitive parsing of the source file for -compile mode.
-command filename	Specifies a text file from which to read command line parameters for all modes.

Any other options are for internal use only.

Creating an Alternative Service Layer Message File

To create an alternative service layer message file,

1. Set up the required environment:

   MPE/iX: Use the COGNLSTAB and SRVCMSGS environment variables. Note that the file locations are POSIX locations. The <version> is the PowerHouse or PowerHouse Web version, for example PH849C. Specify the service layer message template file you wish to modify, such as SRVCMSGE.

   ```
   SETVAR COGNLSTAB "/COGNOS/<version>/COGLANG"
   SETVAR SRVCMSGS "/COGNOS/<version>/SRVCMSGE"
   ```

   OpenVMS: Use the logical SRVCMSGS to point to the service layer message file you wish to modify, such as SRVCMSGE.MSG.

   ```
   DEFINE SRVCMSGS PH_MESSAGE_LOCATION:SRVCMSGE.MSG
   ```

   UNIX: Use the environment variable SRVCMSGS to point to the service layer message file you wish to modify, such as srvcmsgs.msg in the English language subdirectory.

   ```
   setenv SRVCMSGS $PH_USR/msg/english/srvcmsgs.msg
   ```

   Windows: Change the locations specified in the SRVCMSGS entry in the powerhouse.ini or phweb.ini file to point to the service layer message file you wish to modify, such as srvcmsg_en.msg.

2. **MPE/iX:** Decompile the message file using the command

   ```
   RUN COGMC.<version>.COGNOS;INFO='-decompile -input <compiled file> -output
   <source file> -character_set ASCII'
   ```

   OpenVMS, UNIX, and Windows: Decompile the message file using the command:

   ```
   cogmc -decompile -input <compiled file> -output <source file>
   -character_set ASCII
   ```

3. Using an editor, make the modifications to each message that you want to change, and save the changes.

4. **MPE/iX:** Compile your messages using the following command:

   ```
   RUN COGMC.<version>.COGNOS;INFO='-compile -input <source file> -output <new
   compiled file> -character_set ASCII'
   ```

   OpenVMS, UNIX, and Windows: Compile your messages using the following command:

   ```
   cogmc -compile -input <source file> -output <new compiled file>
   -character_set ASCII
   ```

5. To use the newly compiled message file

   MPE/iX: Set file equations to restore the environment and reference the alternative message file. Note that the file locations are MPE/iX locations. For example,

   ```
   SETVAR COGNLSTAB="COGLANG.<version>.COGNOS"
   DELETEVAR SRVCMSGS
   FILE SRVCMSGS=MYMSG.MSG
   ```

   OpenVMS: Define the system logical SRVCMSGS to reference this alternative file.

   ```
   DEFINE SRVCMSGS PATHALTMSG:[ANOTHER.PATH]MYMSG.MSG
   ```

   UNIX: Modify the environment variable SRVCMSGS to reference this alternative file.

   ```
   setenv SRVCMSGS "/user/home/mymsgfile.msg"
   ```

Windows: Modify the SRVCMSGS entry in the powerhouse.ini or phweb.ini file. Or, use the SRVCMSGS environment variable. The ini file entry takes precedence over the environment variable.

Runtime Message Format

As an example from the sample message file provided above, the message that is entered as

```
INTCALL_UDF "The internal_call() function was not specified."
```

displays at runtime as:

```
EXPENG-E-INTCALL_UDF, The internal_call() function was not specified.
```

This message, displayed at runtime, is built from four pieces of information

- the component to which the message belongs
- the severity associated with the message
- the message identifier
- the message itself

```
<component>-<severity>-<message id>, <message>
```

Rules for Modifying Default Service Layer Messages

When you modify an alternative service layer message file, ensure that:

- the modified message contains the same number of substitution characters as the original message, if you modify a message that contains substitution characters
- only the message text, substitution parameter and the component name and short name are modified
- the message identifier, severity and prefix remain the same
- none of the following keywords can be used for component identifiers or names:
  CATEGORY
  COMPONENT
  PARAMETER
  PREFIX
  NAME
  SEVERITY

Designer Messages

You can create application-specific designer messages for use with each PowerHouse component and service layer message. Designer messages are meant to supplement default messages, whereas alternative messages are meant to replace default messages.

To create designer messages, simply change the messages as required in the following message files:

Designated Message File[1]	UNIX/Windows Environment Variable	Component
qkmsgdes	QKMSGDES	QUICK
qtpmsgdes	QTPMSGDES	QTP
qzmsgdes	QZMSGDES	QUIZ

[1]The file extension for **OpenVMS**, **UNIX** and **Windows** message files is .txt.

You can define both alternative message files and designer message files. The former replace standard PowerHouse messages wherever they occur, while the latter are used to replace hard-coded messages in your screen.

Designer messages allow you to change messages in your screens without having to recompile the screens. Instead, you simply change the designer messages as required in your designer message files. These messages are retrieved as required at execution time.

Format of Message File

A designer message file is an order-dependent text file. PowerHouse 4GL uses the record number to retrieve a message. A message may contain one or more message substitution characters, by default, a caret (^), which PowerHouse uses to insert additional text.

When modifying a message or translating a message into another language, you may want to alter the order of substitution. To do this, see (p. 271).

Example of a Designer Message File

```
•
•
•
Enter a valid employee name.
This value is not within the range for page length.

Notify ^. ^, billings are less than $300.
Not a valid project code.
•
•
•
```

Creating a Designer Message

In QUICK, the designer message file is created using the system editor. The designer message file must be a flat text file. Message number 1 is contained in the first record, message number 2 in the second record, and so on.

In QDESIGN, QTP, and QUIZ you can reference the designer messages in the SUBSTITUTE statement.

In QDESIGN, you can also reference the designer messages in

- the MESSAGE option of the ERROR, INFORMATION, SEVERE, and WARNING verbs
- the LOOKUP MESSAGE option of the FIELD statement
- the BALANCE MESSAGE option of the ITEM statement

For example, instead of specifying a string for a particular message, you can specify the number of a message in the QUICK designer message file, as in

```
> FIELD EMPLOYEE LOOKUP ON EMPLOYEES MESSAGE 10
```

In this example, QUICK uses the message number 10 in your QUICK designer message file. This allows you to change messages without recompiling screens.

If QUICK cannot find a message relating to the number used, it displays a standard message containing the missing message number and issues the appropriate warning or error message.

OpenVMS: After creating your Designer Message file, convert the message files using the following command:

```
$convert/fdl=PH_MESSAGE_LOCATION:MSG.FDL/PAD=%D32 designermsgfile.txt
designermsgfile.txt
```

Text Order Numbering

Both PowerHouse 4GL and designer message files support text order numbering.

Text order numbers control the order in which message strings replace substitution characters. You can use them only when a PowerHouse message contains more than one substitution character and when you want to explicitly control the order of message substitutions going into your message.

When you use multiple substitution characters in a single string and you don't specify text order numbers, all substitutions are made strictly in order of appearance. That is, the first value is substituted for the first substitution character, the second value for the second substitution character, and so on. This dependence on order can be limiting when a PowerHouse system is being translated from one language to another. Translation often requires the reordering of sentences, and consequently, message substitutions.

To modify the substitution order in a message, you must append text order numbers to the message substitution characters in that message. For example, suppose that in preparing a translated message file you decide to reverse the sentence structure of this message

```
Data name UPS may conflict with keyword UPSHIFT.
```

In this example, the message contains two substitution characters as follows:

```
Data name ^ may conflict with keyword ^.
```

To reverse the order in which the replacement text (in this case UPS and UPSHIFT) is inserted into the message, you can restructure the original message and change it so that it contains text order numbers, as in

```
Keyword ^2 may conflict with data name ^1.
```

At run time, messages are substituted into the string as follows:

```
Keyword UPSHIFT may conflict with data name UPS.
```

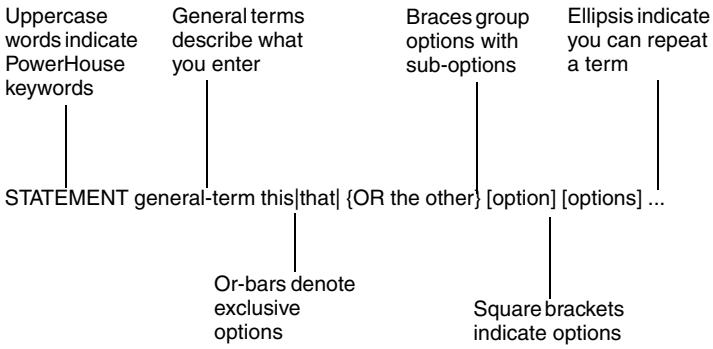
Chapter 5: PowerHouse Language Rules

Overview

This chapter contains detailed discussions of

- PowerHouse syntax and its presentation
- general terms that appear in syntax
- entering statements
- PowerHouse arrays, conditions, expressions, items, item datatypes, decimal alignment and scaling, and pattern matching
- QUICK screen commands
- blob support
- null value support
- conditions, summary operations, expressions and pattern matching in SQL

Syntax Symbols in PowerHouse



Uppercase and Lowercase

Words shown in uppercase are keywords (for example, STATEMENT and OR). While shown in uppercase in the syntax, keywords can be entered in uppercase, lowercase, or mixed case. For more information about abbreviating keywords, see (p. 281).

Words shown in lowercase are general terms that describe what you should enter (for example, general-term and this). For more information about general terms, see (p. 275).

Square Brackets

Square brackets ([ ]) indicate an option. (Anything not in square brackets must be entered.) Both keywords and general terms can be optional. If a keyword is optional, it's enclosed within square brackets. In this example,

SELECT record-structure [IF condition]

the IF condition is optional.

You can enter options enclosed in square brackets ([ ]) in any order unless their description states otherwise. For example, the syntax

SET NOVERIFY [DELETE] [ERRORS]

allows you to enter options like this:

```
> SET NOVERIFY DELETE ERRORS
```

or

```
> SET NOVERIFY ERRORS DELETE
```

Braces

Braces ({ }) indicate keywords and general terms that form a distinct group. In the example on (p. 273), the mutually-exclusive option, "OR the other", is a group that must be entered together.

Ellipsis

An ellipsis (...) indicates that you can repeat the preceding option or general term. For example, the syntax

SET option...

Indicates that you can specify more than one option, but you must specify at least one option.

Similarly,

SET JOB [option]...

indicates that you can specify as many options as are available for the SET JOB statement or not specify any at all.

Or-Bars

An or-bar (|) indicates that you can only specify one listed option; these options are mutually exclusive. In the example, you can enter

STATEMENT general-term this

or

STATEMENT general-term that

or

STATEMENT general-term OR theother

Stacked Syntax

Syntax is also mutually exclusive when it is stacked. For example,

SELECT [IF condition]
SELECT record-structure [IF condition]

Indented Syntax

Indented syntax indicates that a single statement or option is too long to fit on one line of text.

This indented syntax isn't mutually exclusive:

PAGE HEADING [ALIGN|NOALIGN] [report-group]
[KEEP [COLUMN] [HEADINGS]
[SKIP[n]]]

General Terms in PowerHouse

General terms are part of the statement syntax. They are terms that describe what you should enter.

This list defines the general terms that PowerHouse uses most frequently in PowerHouse syntax:

account	The name of an **MPE/iX** account.
application-line	A line in simulated terminal memory having the value of 1 to 240.
array	An item declared as a repeating item in the data dictionary.
asc	An application security class declared in the data dictionary.
case-expression-set	Compares the value of an item against a value or range of values and selects one expression-set to be used to determine the values for the linkitem.
case-processing	A specification for comparing the value of an item against a known value or series of values and performing actions based on the outcome of the comparison.
char	A single character (a letter, number, or special character) enclosed in double or single quotation marks.
character	A single character not enclosed in quotation marks.
characters	One or more characters not enclosed in quotation marks.
colon-variable	A variable-name prefixed by a colon (:) used in SQL to identify PowerHouse variables.
column	The name of a column in a table or view in a relational database declared in the data dictionary.
column-name	The name of a simple or derived column.
columnspec	A fully qualified column-name.
condition	A logical test that must be satisfied in order for a specified action to occur (see (p. 289)).
conditional-command-list	One or more commands separated by commas that specifies what commands the PRECOMMANDS or POSTCOMMANDS options on the DESIGNER procedure execute, and, optionally, under what conditions.
conditional-expression	A means of evaluating a series of expressions based on conditions. See (p. 298).
cursor	The name of a set of data declared on a DECLARE CURSOR statement.
cursor-name	The name of a cursor defined by the PowerHouse DECLARE CURSOR statement.
cursor-reference	A cursor-name or table-name named in a CURSOR statement.
database	The name of a relational database declared in the data dictionary.
dataset	The name of an IMAGE or Eloquence dataset.
datatype	Determines the format that PowerHouse uses when it stores the item.

date-expression	An expression that results in a six-digit or eight-digit date, or a datetime (see (p. 301)).
date-format	The sequence in which the day, month, and year portions of a date are entered and displayed. A date-format specified as a system-wide option must include the day, month, and year.
days	A number of days.
dd	A number representing a day of a month.
ddd	A number representing a day of the year.
defined-item	An item declared on a DEFINE statement in QDESIGN, QUIZ, or QTP.
derived-column	A column derived from an expression. To directly reference the derived column in PowerHouse, a name must be assigned to the expression, as in: **expression [AS column-name]**
dsc	A dictionary security class declared in the data dictionary.
element	The name of an element defined in the data dictionary.
entity	One of DATABASE, ELEMENT, FILE, SYSTEM, TRANSACTION, or USAGE.
etp	An execution time parameter declared in the data dictionary.
expression	One of date-expression, numeric-expression, or string-expression.
extension	A three character code indicating the type of file.
field	An item declared on a FIELD statement in QDESIGN.
file	A collection of data records. A file can contain more than one record-structure. In syntax, the general term file refers to a file declared in the data dictionary.
filelocation	The physical location.
filename	The name of a file without the filelocation (and, for UNIX files, without the suffix).
filespec	The specification for a physical file, as it is identified to the operating system. A file specification takes the general form: **MPE/iX:** **[*]filename[/lockword][.group[.account]** **OpenVMS:** **[device:][directory] filename[.extension][;<version>]** Square brackets are required around a directory name. **UNIX:** **[/] [directory /] ... filename** Windows **[drive:\] [directory\ ] ... filename** **Limit (OpenVMS, UNIX, Windows):** The maximum length for a filespec is 256 characters. Filespecs are restricted to alphanumeric and punctuation characters. The characters $ and leading question mark (?) have special meanings and are prohibited. Filespecs are case sensitive.
format	A set of options that govern how an item is displayed in QUIZ.
function-result	The result of the operation of a data-manipulation function.

group	The name of an **MPE/iX** group.
highlight	A screen enhancement. The AUDIBLE highlight is not available on the FIELD statement in QDESIGN.
hours	A number of hours with a value of 0 to 23.
hundredths	A number of hundredths of a second.
index	Provides a way to link records in one file with those in another. Indexes can be either primary or alternate. The first index is the primary index by default. All subsequent indexes are alternate indexes.
indexname	The name of a PowerHouse index as defined in the data dictionary or a relational database.
item	A record item (an item declared in the data dictionary), a predefined item, a temporary item, a global temporary, or a defined item.
keyword	Any word in a statement or command that has a specific meaning to PowerHouse.
linkage	A connection between record-structures.
linkitem	A segment of an index or a column in a relational table.
lockword	The lockword for an MPE or KSAM file on **MPE/iX**.
logical function	A function that can be tested directly in a condition. Logical functions can be used within SQL. Any expression within SQL must be an sql-expression but cannot reference defined-items, predefined-items, record-items, table-columns, or temporary-items.
logonid	The system logonid of a user. **MPE/iX:** has the general form **user.account** **OpenVMS:** An **OpenVMS** user name with a limit of 12 characters.
metacharacter	A PDL keyword describing a pattern matching metacharacter; either ALPHA, ANY, DIGIT, ESCAPE, LEFTP, NOT, NULL, OPTIONAL, OPTREP, OR, REPEAT, RIGHTP, or WILD.
minutes	A number of minutes with a value of 0 to 60.
mm	A number representing a month with a value of 1 to 12.
mmm	A three-letter month abbreviation; either JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC.
m or n	Values that you replace with numbers.
name	A unique name identifying a PowerHouse entity. You use a name 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 SYSTEM OPTIONS statement in the dictionary. You can specify a name up to 64 characters long. Names are automatically shifted to uppercase unless the program is run with the NOSHIFT parameter, or the SET NOSHIFT is in effect.

numeric-expression	A single number or a series of terms that yield a numeric result (see (p. 300)).
pattern	A string of characters that describes a value or a group of values (see (p. 351)).
open-name-string	A valid database open specification that is passed directly to the database server in order to gain access to the database.
predefined-condition	An automatically defined condition.
predefined-item	An automatically defined temporary item in QDESIGN; either FIELDTEXT, FIELDVALUE, OCCURRENCE, PATH, or SUBPATH.
predefined-value	Indicates a specific predefined value to display.
procedural-statement	A series of procedural statements enclosed within BEGIN and END control structures, a single procedural verb, or one of the control structures BLOCK TRANSFER, DISABLE, FOR, IF, WHILE, or WHILE RETRIEVING. If DISABLE is used, it must be the only procedural statement in the procedure.
project-list	A list of column-names selected by the query-expression. If duplicate names exist in multiple declare cursors, the option OF cursor-name can be used to identify the proper item.
query-expression	A query specification or the union of two or more query-specifications.
query-specification	Defines a collection of rows that will be accessible when the cursor is opened.
record	The name of a record-structure defined in the data dictionary.
record-complex	Consists of a data record from the primary record-structure in the ACCESS statement and one data record from each of the related "linked to" record-structures; together they form a compound record. PowerHouse builds a series of record complexes for each data record in the primary record-structure.
record-item	An item defined in a record-structure.
record-structure	The name of a record-structure defined in the dictionary or a table in a relational database.
report-group	In QUIZ, determines the content and format of detail lines, headings, and footings. The general form is: **[report-item]... [SKIP[n|PAGE]]** **[RESERVE n [LINES]]**
report-item	In QUIZ, specifies what to report, its position and format, and whether to print it on every line or only at a control break. The general form is: **[SKIP [N|PAGE]] [TAB n] content** **[format-options]** **[PRINT AT sort-item]**
report-name	In QUIZ, the name of a compiled PowerHouse report.
row	A line relative to the current screen; has a value of 1 to 24.
seconds	A number of seconds and has a value of 0 to 60.

segment	An item that is part of an index. An index can contain one or more segments.
screen-name	The name of a compiled screen.
sort-item	An item that you name in a SORT or SORTED statement in QUIZ or QTP. A change in the value of a sort-item signals a control break.
sql-conditions	Conditions which are limited to use within SQL syntax (see (p. 298)).
sql-datatype	One of BINARY, CHARACTER, DATE, DATETIME, DECIMAL, DOUBLE, FLOAT, INTEGER, INTERVAL, LONGVARBINARY, LONGVARCHAR, QUADWORD, SMALLINT, TIME, VARBINARY, or VARCHAR.
sql-date-expression	An sql-expression that yields a value of SQL type DATE, INTERVAL, TIME, or TIMESTAMP.
sql-expression	One of sql-date-expression, sql-numeric-expression, or sql-string-expression.
sql-numeric-expression	Has the general form: **sql-term [sql-operator sql-term]...**
sql-operator	Denotes a mathematical function and is one of +, -, *, or /.
sql-summary-operations	AVG, COUNT, SUM, MAX, and MIN operations used within SQL.
sql-substitution	An sql-substitution can be specified for any substitution variable defined on the DECLARE CURSOR statement. Two default sql-substitutions, WHERE and ORDERBY, will be inserted in generated SQL statements even if the corresponding substitution-variables do not exist on a DECLARE CURSOR statement. The syntax for an sql-substitution is: **substitution-variable (text)**
sql-substitution-variable	A name for SQL syntax that is substituted for that name when the statement is parsed.
sql-syntax	Any SQL syntax that is valid when substituted for the corresponding sql-substitution-variable.
string	A series of characters (letters, numbers, or special characters) enclosed in double or single quotation marks. In SQL a string must be enclosed in single quotation marks.
string-expression	A single string or series of items that yield a string result (see (p. 300)).
subfilespec	The filespec for a self-describing file.
subquery	Identical to a query-specification with two exceptions: the subquery must project a single-column table and the syntax of the subquery includes enclosing brackets.
subscript	A numeric expression that evaluates to a number used to reference a specific occurrence of an array.

substitution-variable	A variable-name prefixed by the double colon (::) which is used in SQL to identify the location for substitutions. The text is the default substitution if no other substitution is specified. **::variable-name(text)**
system-function	A function that returns a value set by PowerHouse.
tablespec	A fully qualified table-name.
table-name	The name of a table or view defined in a relational database.
temporary-item	An item declared on a TEMPORARY statement in QDESIGN or QTP, or a GLOBAL TEMPORARY statement in QTP.
term	Either string, number, item, expression, system-function, function-result, case-expression-set, parameter specification, USER system variable, column specification, or sql-summary-operation (see (p. 300)).
termtype	A terminal type supported by QUICK.
transaction	The name of a relational transaction declared on a TRANSACTION statement in QDESIGN, or one of the predefined transactions: Consistency, Query, or Update.
type	Determines what type of data an item holds and what type of operations are valid.
type-option	An option that assigns special attributes to NUMERIC or DATE items.
usage	The name of a usage declared in the data dictionary.
value	A single string or a single number, depending on the item type.
value-set	A single value or a range of values with the general form **value [TO value]**
yy	A number representing a year excluding the century.
yyyy	A number representing a year including the century.

Entering Statements

The PowerHouse default prompt is >. Once you start a PowerHouse component, you can enter one statement per line. For example

```
> FIELD EMPLOYEENUMBER OF EMPLOYEES REQUIRED &
>    NOCHANGE LOOKUP NOTON EMPLOYEES
```

If you want to split a line for better readability, or if your statement exceeds one line, you can split the statement across any number of lines by placing an ampersand (&) at the end of every incomplete line. For example

```
> FIELD EMPLOYEENUMBER OF EMPLOYEES &
>    REQUIRED &
>    NOCHANGE &
>    LOOKUP NOTON EMPLOYEES
```

Anything that appears after the continuation character in a line is ignored.

You can begin statements and continuation lines in any column.

You must separate each part of a statement (keywords, values, and entity names) by at least one space.

You can enter options in any order, unless otherwise specified.

Abbreviating Keywords

In most instances, you can abbreviate PowerHouse keywords with the first three or more characters. For example, you can abbreviate the REVISE statement to REV or ACCESS to ACC. In some cases, you can abbreviate keywords to fewer than three characters. For example, you can abbreviate the EXIT statement to EXI, EX, or E. PowerHouse displays a warning when you enter a keyword abbreviation that could refer to more than one keyword. For example, the abbreviation CON can refer to either CONSISTENCY or CONCURRENCY.

Avoiding Conflicts Between Keywords and Record or Item Names

If a record-structure name or item name is the same as a keyword or abbreviation, prefix the name with a percent sign (%) to avoid ambiguity. For example, %MOD refers to a record-structure or item named MOD rather than the data manipulation function named MOD.

What Happens When You Enter Statements

As you enter statements, PowerHouse checks for the proper syntax and content.

Statements are non-procedural and have a free-form syntax. In all components, statements must be entered in a logical structure. For more information, see the statements chapters in the *reference manuals for QUIZ, QTP, QDESIGN, and PDL*.

If PowerHouse finds an error or an unusual condition, an error or warning message is issued.

PowerHouse ignores:

- the remainder of the statement after issuing a syntax error (if you make a syntax error, a list of expected statements is displayed)
- completely blank lines
- anything after a semicolon (indicating comments), unless a semicolon occurs within a string

Entering Comments

A comment begins with a semicolon and continues to the end of the line. For example,

```
> SCREEN INVOICE ;create screen named INVOICE
> FILE INVOICES OCCURS 9
> FILE STOCK REFERENCE
> ;multiply the value entered for PRICE with the
> ;value entered for QUANTITY and then
> ;add the TAX amount
```

```
> DEFINE DOUBLECHECK NUMERIC*8 = PRICE * QUANTITY &
>     + TAX
```

Entering Conditional Compile Statements

Conditional compile statements include

@IF [NOT] name [[AND|OR] [NOT] name]...
[@ELSEIF name [[AND|OR] [NOT] name]...]
@ELSE
@ENDIF

You must include a predefined conditional compilation parameter or a name in the @IF and @ELSEIF statements.

You must precede conditional compile statements with an at-sign (@) in column 1. The statement following a conditional compilation statement must be on a separate line.

You can continue conditional compile statements by entering an ampersand (&) at the end of a line and then beginning the continuation line with an at-sign (@) in column 1.

Creating Compound Condition Statements

You can create compound conditions by using the logical operators NOT, AND, and OR. PowerHouse gives precedence in compound conditions in the following order:

1. NOT
2. AND
3. OR

Thus, PowerHouse will process the following:

NOT condition1 AND condition2 OR condition3

as

(((**NOT condition1**) **AND condition2**) **OR condition3**)

Types of Conditions

You can use conditional compilation for single, multiple or predefined conditions.

Single and Multiple Conditional Compilation

Single conditional compilation is when one conditional compile flag is specified; multiple conditional compilation is when two or more conditional compile flags are specified.

Predefined Conditional Compilation

PowerHouse predefines one or more conditional compile flags to simplify cross-platform development. These flags are always TRUE when PowerHouse is running on their associated platform (hardware/OS). These flags indicate which hardware and operating system PowerHouse is running on.

The following table lists these predefined conditional compile flags by hardware and operating system:

Hardware	Operating System	Predefined Conditional Compile Flags
HP e3000 PA-RISC	MPE/iX	HPMPEXL
HP 9000 PA-RISC	HP-UX	UNIX, HPUX, PARISC
HP Integrity Itanium 2	HP-UX	UNIX, HPUX, IA64
HP AlphaServer	OpenVMS	OPENVMS, VAXVMS, ALPHA
HP VAX	OpenVMS	OPENVMS, VAXVMS, VAX
HP Integrity Itanium 2	OpenVMS	OPENVMS, VAXVMS, IA64

Hardware	Operating System	Predefined Conditional Compile Flags
IBM RS/6000, pSeries	AIX	UNIX, IBMR2AIX
Intel IA32 PC	Windows	WINDOWS
Sun SPARC	Solaris	UNIX, SOLARIS

For each operating system in the preceding table, the conditional compile flags are listed in order from generic to specific. For example, for the HP-UX operating system, the UNIX flag is generic whereas the HPUX flag is specific. As a rule, use the most generic flag possible since specific flags reduce the cross-platform portability of your applications. Specific flags should be used only when some of the code is hardware dependent, such as floating point accuracy.

Operating System Commands

To execute an operating system command in PowerHouse:

MPE/iX:	begin a line with a colon (:).
OpenVMS:	begin a line with a dollar sign ($).
UNIX, Windows:	begin a line with an exclamation mark (!).

To write an operating system command to the current save file:

MPE/iX:	begin a line with two colons (::) followed by the command. In the save file, only one colon is stored.
OpenVMS:	begin a line with two dollar signs ($$) followed by the command. In the save file, only one dollar sign is stored.
UNIX, Windows:	begin a line with two exclamation marks (!!) followed by the command. In the save file, only one exclamation mark is stored.

Arrays in PowerHouse

An array (also called a repeating item) is an item in a record or a temporary item that the designer defines with multiple occurrences. For example, if you want to store twelve monthly totals in a record, you can define a single item, AMOUNT, with twelve occurrences. The resulting structure is an array. QSHOW lists this element with its name (AMOUNT) and attributes, and the corresponding item includes the number of occurrences (12).

Using Arrays in QDESIGN

To manipulate data in arrays in QUICK, you must create a field for each occurrence of an item defined with multiple occurrences. You do this by using the OCCURS WITH option of the CLUSTER statement in QDESIGN. QDESIGN does this automatically if you use the GENERATE statement to create your fields.

You cannot declare a repeating item within a file that repeats on the screen.

The following example shows how you can include all the occurrences of the item AMOUNT (as defined in a record-structure named BILLINGS) on a QUICK screen:

```
> SCREEN YRAMT
> FILE BILLINGS PRIMARY
> FIELD CUSTOMERNO OF BILLINGS
> FIELD CUSTOMERNAME OF BILLINGS
> CLUSTER OCCURS WITH AMOUNT OF BILLINGS
> FIELD AMOUNT OF BILLINGS
> CLUSTER
> BUILD
```

The YRAMT screen contains one field for each occurrence of the array AMOUNT. PowerHouse labels each of these AMOUNT fields identically with the label specified in the data dictionary. You can suppress the common label with either the ALIGN statement or the NOLABEL option of the FIELD statement. You can then use the TITLE statement to generate more meaningful labels, such as JAN for the first occurrence, FEB for the second occurrence, and so on for the individual occurrences of the array, AMOUNT.

Referencing an Occurrence

By default, each occurrence of a repeating item in an array appears on a screen with an ID-number, allowing the QUICK screen user to reference each occurrence.

QDESIGN generates FOR control structure loops in the applicable procedures, based on CLUSTER statements in the layout section. The FOR control structure sets the predefined item, OCCURRENCE. This function controls which occurrence of a repeating item PowerHouse addresses.

You can also address a specific occurrence of an item in an array using procedures, as in

```
> PROCEDURE INTERNAL FIXARRAY
>   BEGIN
>     FOR EACH ARRAY1
>       IF OCCURRENCE = 1
>         THEN LET ARRAY1 = 1
>       ELSE IF OCCURRENCE = 2
>         THEN LET ARRAY1 = 2
.
.
.
> END
```

Automatic Initialization of Arrays in QDESIGN

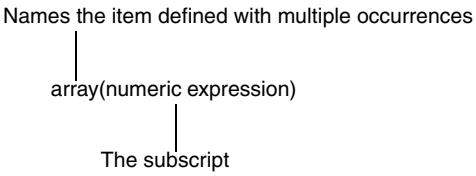
QUICK automatically handles the item initialization of arrays in the different record-structures declared on the screen. When arrays in different record-structures, or an array and a non-repeating item in different record-structures have the same name, QUICK performs automatic initialization in the same way as QTP. For information about automatic initialization in QTP, see (p. 287).

QUICK follows specific rules when initializing individual occurrences of an array based on occurrences of either another array or a non-repeating item. When arrays in different record-structures or an array and a non-repeating item in different record-structures have the same name, QUICK performs automatic initialization based on the target and source. The following rules apply to automatic initialization of arrays:

- If the target array has the same number of occurrences as the source array, there is a one-to-one correspondence between them.
- If the target array has fewer occurrences than the source array, there is a one-to-one correspondence between them, up to the last occurrence of the target array.
- If the target array has more occurrences than the source array, there is a one-to-one correspondence between them, up to the last occurrence of the source array. QUICK initializes the extra occurrences in the target array to blank, zeros or default values.
- If the target is an array and the source is a non-repeating item, QUICK initializes all occurrences of the array to the value of the item.
- If the target is a non-repeating item and the source is an array, QUICK initializes the item to the value of the first occurrence of the array.

Subscripting in QUIZ and QTP

In QUIZ and QTP, you can reference individual occurrences within an array with subscripts. The general syntax of a specific occurrence of an array is



The general term array is the name of the item defined with multiple occurrences; numeric-expression is the subscript. In the REPORT statement, you must use literal numbers as array subscripts, as in

```
amount(1)
```

When you reference an array occurrence in a DEFINE statement, you can use numeric expressions or items as array subscripts. For example, if the numeric item AMT is such that the expression

```
AMT + 6
```

returns a number that lies within the range defined for AMOUNT, you can use AMOUNT (AMT + 6) to reference an individual occurrence of the array.

The maximum number of item occurrences within an array that QTP can address is 4,096.

Array subscripts must always lie within the range of values defined for the array in the data dictionary. For example, if you define a single item AMOUNT with twelve occurrences, a subscript that references an individual occurrence of the array AMOUNT must be greater than or equal to 1 and less than or equal to 12.

Using Arrays in QUIZ

You can

- add some or all occurrences of an array
- reference individual occurrences of array items by subscripting

The SUM function can sum the entire array or a subset of the occurrences in an array. For a detailed description of the SUM function, see (p. 466).

To report an individual occurrence of an array, you must use a number rather than a numeric expression as the array subscript. Assume that PowerHouse stores a numeric value for each of the twelve calendar months in the corresponding twelve occurrences of MONTHLYBILLINGS. To report the value for January, enter

```
> REPORT MONTHLYBILLINGS(1)
```

If you reference an array without a subscript, PowerHouse reports only the first occurrence of the array. If you want to report all occurrences, you must explicitly reference each occurrence of the array by subscript or specify REPORT ALL.

In contrast to the report statement, the define statement allows you to use literal values, items, or numeric expressions as array subscripts. For example, if the item, currentmonth, contains a value in the range 1 to 12, you can report the occurrence of the array, MONTHLYBILLINGS, for the current month by entering

```
> DEFINE CURRENTAMOUNT = MONTHLYBILLINGS(CURRENTMONTH)
> REPORT CURRENTAMOUNT
```

You can't write arrays to QUIZ subfiles and later access them by individual occurrence. PowerHouse defines the subfile content with the REPORT statement; therefore, PowerHouse writes the array as a series of individual items with the same name. As a result, the only value that you can report from the array is the first one.

To write all of the occurrences of an array to a subfile, you must create a defined item for each array occurrence, and write the values of the defined items to the subfile. For example,

```
> DEFINE JANBILLINGS=MONTHLYBILLINGS (1)
.
.
.
> DEFINE DECBILLINGS=MONTHLYBILLINGS(12)
> REPORT JANBILLINGS ... DECBILLINGS
> SET SUBFILE NAME MONTHLYBILLINGS KEEP
```

Arrays in the EDIT Statement

QUIZ either edits the entire array or one particular occurrence within the array. If you use an array in an EDIT statement without a subscript, each occurrence in the array must pass the edit. If you use a subscript, you edit only that occurrence.

Using Arrays in QTP

In QTP, you can

- add some or all occurrences of an array
- perform calculations involving entire arrays
- access individual occurrences of an item

The SUM function can sum the entire array or a subset of the occurrences in an array. For a detailed description of the SUM function, see (p. 466).

Array subscripts are either literal numbers such as AMOUNT(3) or numeric expressions such as AMOUNT(AMOUNT+6). When you use a numeric expression as an array subscript and it results in a fractional value, QTP truncates the value.

If you try to reference an individual array occurrence with a subscript that lies outside the range defined for the array, QTP issues a calculation error.

You can use a subscripted array item almost anywhere that you can use an unsubscripted item. However, you can't use a subscripted array item as a sort-item or in the INCLUDE option list of a SUBFILE statement. You can bypass both of these restrictions by defining a new item equated to the array occurrence, as in

```
> define month = parm
> define sortitem = amount(month)
> sort on Sortitem
```

Arrays in the EDIT Statement

QTP either edits the entire array or one particular occurrence within the array. If you use an array in an EDIT statement without a subscript, each occurrence in the array must pass the edit. If you use a subscript, you edit only that occurrence.

Arrays in the ITEM Statement

You can use individual occurrences of an array in the item statement, as in

```
> item amount(12) final 1000
```

In this case, QTP references only the designated occurrence.

You can also reference all occurrences of an array in the item statement. For example, the statement

```
> item amount initial 0
```

initializes all 12 occurrences of the array amount to zero.

An array can be both the source and target in an ITEM statement, as in

```
> item array1 initial array2
```

In the preceding example, AMOUNT is declared to be OCCURS 12.

If both arrays have the same number of occurrences, QTP initializes each occurrence in ARRAY1 to the value of the corresponding occurrence in ARRAY2. If ARRAY2 has more occurrences than ARRAY1, QTP ignores the extra occurrences in ARRAY2. However, if ARRAY2 has fewer occurrences than ARRAY1, the run will terminate.

A non-repeating item cannot be the target of an array, but it can be the target of one occurrence of an array, as in

```
> ITEM COST FINAL ARRAY1(2)
```

QTP gives the item COST the value of the second occurrence of ARRAY1.

Performing Calculations on Entire Arrays

When the target of an ITEM statement is an unsubscripted array, any expression that is a part of that ITEM statement can also include the unsubscripted array. This is particularly useful for performing calculations on entire arrays. If you perform such calculations, the unsubscripted arrays referenced by the calculation must have at least as many occurrences as the unsubscripted target array. If this is not the case, a calculation error occurs.

For example, suppose you have defined three arrays: monthtotcurr, monthtotprev, and monthaverage. Each of these arrays contains twelve items (one for each month). The statement

```
> item monthaverage final &
>   (monthtotprev + monthTotcurr) / numyears
```

calculates twelve different amounts for the array monthaverage using the corresponding twelve values of the monthtotprev and monthtotcurr arrays. This ITEM statement is equivalent to the following statements:

```
> item monthaverage(1) final &
>   (monthtotprev(1) + monthtotcurr(1)) / Numyears
> item monthaverage(2) final &
>   (monthtotprev(2) + monthtotcurr(2)) / numyears
.
.
.
> item monthaveraGe(12) final &
>  (monthtotprev(12) + monthtotcurr(12)) / numyears
```

Automatic Initialization of Arrays in QTP

QTP follows specific rules when initializing individual occurrences of an array based on occurrences of either another array or a non-repeating item. When arrays in different record-structures or an array and a non-repeating item in different record-structures have the same name, QTP performs automatic initialization based on the target and source. The following rules apply to automatic initialization of arrays:

- If the target array has the same number of occurrences as the source array, there is a one-to-one correspondence between them.
- If the target array has fewer occurrences than the source array, there is a one-to-one correspondence between them, up to the last occurrence of the target array.
- If the target array has more occurrences than the source array, there is a one-to-one correspondence between them, up to the last occurrence of the source array. QTP initializes the extra occurrences in the target array to blank, zeros, or default values.

- If the target is an array and the source is a non-repeating item, QTP initializes all occurrences of the array to the value of the item.
- If the target is a non-repeating item and the source is an array, QTP initializes the item to the value of the first occurrence of the array.

Referencing a specific occurrence of an array in the ITEM statement precludes automatic initialization. If you use the ITEM statement to calculate the value of a specific occurrence of an array, you should initialize all the occurrences in the array that aren't calculated.

Conditions in PowerHouse

A condition is a logical test that must be satisfied in order for PowerHouse to perform a specific action.

The general form of a condition is

[NOT] condition1 [{OR|AND} [NOT] condition2]...

where condition is one of:

- logical function
- logical expression
- predefined condition

Logical Function

A logical function is one that can be tested directly in a condition:

MATCHPATTERN (string-expression, pattern-string)

MATCHUSER (string-expression)

VALIDPATTERN (string-expression)

For more information about functions, see (p. 359).

Logical Expression

Logical expressions can be tested as a condition and have the form:

operand operator expression

operand

The operand in a condition is one of the following:

- a string
- a number
- a record item
- a temporary item
- a global temporary item
- a defined item
- a system function

operator

The operator causes a TRUE or FALSE response in a condition, as in

LT	or	<	less than
LE	or	<=	less than or equal to
EQ	or	=	equal to
GT	or	>	greater than
GE	or	>=	greater than or equal to
NE	or	<>	not equal to

expression

See (p. 300).

Predefined Conditions in QDESIGN

Predefined conditions are automatically defined. The designer can use them to test the current state of QDESIGN.

The predefined conditions in QDESIGN are:

ACCESSOK	ALTEREDRECORD
BLOCKMODE (**MPE/iX**)	CHANGEMODE
CHARACTERMODE (**MPE/iX**)	COMMANDOK
CORRECTMODE	CURSOROPEN
DELETEDRECORD	ENTRYMODE
item EXISTS	item [IS] NULL|MISSING
FINDMODE	NEWRECORD
OSACCESS	PROMPTOK
RANGED (linkitem)	SELECTMODE
SQLOK	TRANSACTION

ACCESSOK

Tests whether the last attempted data retrieval succeeded. If you don't specify the OPTIONAL keyword for the retrieval, then QUICK can't test the ACCESSOK condition because a failed retrieval causes an error condition. The ACCESSOK condition relates to a QUICK session and not to a specific QUICK screen.

ALTEREDRECORD [OF record-structure]
DELETEDRECORD [OF record-structure]
NEWRECORD [OF record-structure]

Determines the status of a record currently in a record buffer as follows:

- The NEWRECORD condition is true if the data record is new.
- The ALTEREDRECORD condition is true if QUICK changes the data record.
- The DELETEDRECORD condition is true if QUICK marks the data record for deletion.

The following tables illustrate the way in which QUICK sets the record status at different points during Entry and Find mode processing. This table shows the status of predefined conditions in Entry mode:

Processing Point	NEWRECORD	ALTEREDRECORD	DELETEDRECORD
at Initialization	True	False	False
after field entry	True	True	False
U, UR, US, or UN in the Action field	False	False	False
DELETE verb encountered	True	True or False	True

This table shows the status of predefined conditions in Find mode:

Processing Point	NEWRECORD	ALTEREDRECORD	DELETEDRECORD
after first find	False	False	False
DELETE verb encountered	False	True or False	True
if data changed	False	True	False
U, UR, US or UN in the Action field	False	False	False

If you omit the OF record-structure qualifier, the status is that of the assumed record-structure. The assumed record-structure is the primary file unless it is changed with the SET ASSUMED statement. If there is no assumed record-structure, the status is that of the current record-structure.

BLOCKMODE
CHARACTERMODE (MPE/iX)

- The BLOCKMODE condition is true if the terminal has been switched to, or started in Block mode.
- The CHARACTERMODE condition is true if the terminal has been switched to, or started in Character mode

CHANGEMODE
CORRECTMODE
ENTRYMODE
FINDMODE
SELECTMODE

Determine which mode the screen is in.

- The CHANGEMODE condition is true if QUICK is in Find mode and is ready to accept changes. The CHANGEMODE condition is also true after a successful Update Stay (US) command and during execution of POSTFIND and DETAIL POSTFIND procedures.
- The CORRECTMODE condition is true if the screen is in Entry mode, has completed the standard entry sequence, and is ready to accept corrections.
- The ENTRYMODE condition is true if the screen is in the standard entry sequence or in the APPEND procedure.
- The FINDMODE condition is true if the screen performs Find mode initialization, retrieves data records, or displays retrieved data. For more information, see "Find Mode Processing" in Chapter 5, "QUICK's Processing Modes", in the *QDESIGN Reference*.
- The SELECTMODE condition is true if QUICK is in Select mode, and until a record is found and displayed on the screen. SELECTMODE is false when CHANGEMODE is true.

During the execution of the PATH and FIND procedures, both FINDMODE and SELECTMODE return a true value when the FIND process is triggered by the Select Action command.

COMMANDOK

Tests the status of the last operating system command executed by the COMMAND statement or the RUN COMMAND verb. The COMMANDOK condition is true unless an error was issued during the execution of the last operating system command.

CURSOROPEN(cursor-name)

This condition checks if an SQL cursor is currently open. It returns TRUE if the cursor is currently open; otherwise, FALSE is returned.

item EXISTS

EXISTS returns TRUE if the value for an item is not null; otherwise it returns FALSE.

item [IS] NULL|MISSING

IS NULL or IS MISSING returns TRUE if the value for the item is null; otherwise, it returns FALSE.

For example, you can test for null values in the DEFINE statement, as in

```
> DEFINE DISPLAY_ADDRESS CHARACTER*50 = &
>   "ADDRESS UNKNOWN" &
>   IF ADDRESS IS NULL ELSE ADDRESS
```

or, by using the INITIAL and FINAL options of the ITEM statement:

```
> ITEM FLAG &
>   FINAL "N/A" &
>   IF RELITEM IS MISSING &
>   ELSE " "
```

OSACCESS

Tests whether or not a QUICK screen user can access the operating system. Access to the operating system is controlled through the **osaccess|noosaccess** program parameters, the **dcl|nodcl** program parameters (**OpenVMS**), and the OSACCESS resource file statement.

PROMPTOK [FOR field]

Tests whether the QUICK screen user entered a value in response to the last ACCEPT, PROMPT, REQUEST, or SELECT verb, or if a value was entered into FIELDTEXT by an input procedure. The condition is true if the user entered a value.

The field option allows verification of an entry made in an explicit field. This is for use in Block Transfer constructs.

RANGED(linkitem)

Returns TRUE if the linkitem value terminates with a generic search character. The RANGED condition is used in the QDESIGN PATH procedure to set the SUBPATH variable for SQL.

SQLOK

Tests the status of the last SQL statement. The condition is TRUE unless an error has occurred during execution of the last SQL statement. To determine which error occurred, use the SQLCODE and SQLMESSAGE functions.

TRANSACTION transaction-name IS ACTIVE|INACTIVE

Determines whether a transaction is active or not. This predefined condition returns TRUE or FALSE depending on whether the QUICK transaction has been started (and has not yet been committed or rolled back).

Use this condition to decide whether a transaction should be committed or rolled back.

TRANSACTION transaction-name IS LOCALLY ACTIVE

Determines whether a transaction is locally active. This predefined condition returns TRUE or FALSE depending on whether the QUICK transaction is locally active (and has not yet been committed or rolled back). For more information about locally active transactions, see the *PowerHouse and Relational Databases* book.

Use this condition to decide whether a transaction should be committed or rolled back, or to imitate QUICK's automatic commit/rollback processing.

Predefined Conditions in QTP

Predefined conditions are automatically defined. They are:

ALTEREDRECORD	DELETEDRECORD
item EXISTS	item [IS] NULL|MISSING
NEWRECORD	RECORD

ALTEREDRECORD OF record-structure

The ALTEREDRECORD condition is true if QTP changes the data record. This condition, along with DELETEDRECORD and NEWRECORD, determine the status of a record currently in a record buffer.

DELETEDRECORD OF record-structure

The DELETEDRECORD condition is true if QTP marks the data record for deletion. This condition, along with ALTEREDRECORD and NEWRECORD, determine the status of a record currently in a record buffer.

item EXISTS

EXISTS returns TRUE if the value for an item is not null; otherwise it returns FALSE.

item [IS] NULL|MISSING

IS NULL or IS MISSING returns TRUE if the value for the item is null; otherwise, it returns FALSE.

For example, you can test for null values in the DEFINE statement, as in

```
> DEFINE DISPLAY_ADDRESS CHARACTER*50 = &
>   "ADDRESS UNKNOWN" &
>   IF ADDRESS IS NULL ELSE ADDRESS
```

or, by using the INITIAL and FINAL options of the ITEM statement:

```
> ITEM FLAG &
>   FINAL "N/A" &
>   IF RELITEM IS MISSING &
>   ELSE " "
```

NEWRECORD OF record-structure

The NEWRECORD condition is true if the data record is new. This condition, along with ALTEREDRECORD and DELETEDRECORD, determine the status of a record currently in a record buffer.

RECORD record-structure EXISTS

Tests whether QTP retrieved a record for the specified record-structure. The condition is true if the record-structure has been retrieved.

In the following example, QTP retrieves all the employee data records for which no billings exist:

```
> ACCESS EMPLOYEES LINK TO BILLINGS OPTIONAL
> SELECT IF NOT RECORD BILLINGS EXIST
```

You must ensure that the link to the BILLINGS record-structure is optional. Otherwise, all employee data records that do not have billings associated with them are discarded prior to the selection condition.

Using QTP Predefined Conditions

To determine what happened in the output phase, you can test the status of the output record-structure record buffer. You can use the predefined conditions ALTEREDRECORD, DELETEDRECORD, and NEWRECORD.

For example,

```
> REQ CUSTOMERUPDATE
> ACCESS SALESMASTER
> OUTPUT CUSTOMERMASTER UPDATE
>  ITEM CURRENTBALANCE &
>      FINAL (CURRENTBALANCE + SALESTOTAL)
>  ITEM DATELASTMOD FINAL SYSDATE &
>      IF ALTEREDRECORD OF CUSTOMERMASTER
> GO
```

Consider that the item, salestotal, in the input record-structure salesmaster has a value other than zero. This means that

- when SALESTOTAL is added to the value item, CURRENTBALANCE, of the output record-structure, the content of the output-file record buffer changes and the record status is set to "changed." The predefined condition, ALTEREDRECORD, becomes true.
- the output record-structure is updated only if the values in the record change. If the predefined condition, ALTEREDRECORD, becomes true, an update occurs.
- the last ITEM statement tests the predefined condition, ALTEREDRECORD. If TRUE, and an update is to be performed, then the statement changes the value of item, DATELASTMOD. The change in the date's value and the update output action are both caused by a change in some other part of the record.

This table shows the possible record status settings for the add, delete, and update output actions at each stage of processing.

Timing	ADD	UPDATE	DELETE
immediately after buffer initialization	new, unchanged, undeleted	new, unchanged, undeleted	new, unchanged, undeleted
immediately after successful record retrieval	n/a	old, unchanged, undeleted	old, unchanged, undeleted
immediately after unsuccessful record retrieval	n/a	new, unchanged, undeleted	new, unchanged, undeleted
after processing ITEM statements	new, changed, undeleted	old, changed, undeleted	n/a
after output action	new, changed, undeleted	old, changed, undeleted	old, unchanged, deleted

The following record status combinations don't appear in the table, and never occur in QTP:

- new, changed, deleted
- new, unchanged, deleted
- old, changed, deleted

Thus, deleted can only appear with old, unchanged.

When the record buffer is initialized, the record status is set to new, unchanged, undeleted (that is, the predefined condition, NEWRECORD, is true, and the predefined conditions, ALTEREDRECORD and DELETEDRECORD, are false).

Any ITEM statement that causes a change in the content of the record buffer, including automatic item initialization, causes QTP to set the record status to changed (the predefined condition, ALTEREDRECORD, is true).

For the DELETE and UPDATE output actions, QTP tries to retrieve the relevant data record from the output record-structure. If the retrieval is successful, QTP sets the record status to old, unchanged, undeleted (the predefined condition, NEWRECORD, is false). If the retrieval is unsuccessful, record status remains new, unchanged, undeleted.

After an ADD output action, record status is set to new (the predefined condition, NEWRECORD, is true). After a DELETE output action, QTP sets the record status to deleted (the predefined condition, DELETEDRECORD, is true). In the case of the UPDATE output action, only changed records are updated and the output action has no effect on record status.

Record status is not reset to new, unchanged, undeleted until the record buffer is reinitialized following output. The timing of reinitialization depends on whether output occurs at detail transaction time, at the beginning or end of a transaction set, or at the beginning or end of a control group.

The record status of a subfile can be tested. The timing of status resetting for a subfile depends on the timing of output to it. This is determined by the presence or the absence of an AT option in the SUBFILE statement. ITEM statements with the FINAL option are processed and cause the record status to be set to changed. Other options in ITEM statements for the subfile are invalid. When the subfile record is written, record status is set to old and remains so until the status is reset.

Predefined Conditions in QUIZ

Predefined conditions are automatically defined. They are item EXISTS, item [IS] NULL|MISSING, and RECORD record-structure EXISTS.

item EXISTS

EXISTS returns TRUE if the value for an item is not null; otherwise it returns FALSE.

item [IS] NULL|MISSING

IS NULL or IS MISSING returns TRUE if the value for the item is null; otherwise, it returns FALSE.

For example, you can test for null values in the DEFINE statement, as in

```
> DEFINE DISPLAY_ADDRESS CHARACTER*50 = &
>   "ADDRESS UNKNOWN" &
>   IF ADDRESS IS NULL ELSE ADDRESS
```

or, by using the INITIAL and FINAL options of the ITEM statement:

```
> ITEM FLAG &
>   FINAL "N/A" &
>   IF RELITEM IS MISSING &
>   ELSE " "
```

RECORD record-structure EXISTS

Tests whether PowerHouse retrieved a record for the specified record-structure. The condition is true if the record has been retrieved. In the following example, QUIZ retrieves all the employee data records for which no billings exist:

```
> ACCESS EMPLOYEES LINK TO BILLINGS OPTIONAL
> SELECT IF NOT RECORD BILLINGS EXISTS
```

You must ensure that the link to the BILLINGS record-structure is optional. Otherwise, all employee data records that do not have billings associated with them are discarded prior to the selection condition.

Simple Conditions

Simple conditions consist of an operand plus a single operator plus an expression.

Compound Conditions

You can create compound conditions by using the logical operators NOT, AND, and OR. PowerHouse gives precedence in compound conditions in the following order:

1. NOT
2. AND
3. OR

Thus,

NOT condition1 AND condition2 OR condition3

means

(((NOT condition1) AND condition2) OR condition3)

With OR, the compound condition is true if either or both conditions are true. With AND, the compound condition is true only if both conditions are true.

Order of Precedence

When you combine AND and OR, PowerHouse processes AND first, followed by OR. However, parentheses can alter the order of precedence.

PowerHouse evaluates conditions within parentheses in the order in which they appear, from the innermost set to the outermost set. When AND and OR are used together, use parentheses for clarity.

In the following example, the records selected are those with a STARTDATE greater than January 1, 1994 and a CITY equal to BOSTON, or those with a CITY equal to OTTAWA:

```
> SELECT IF STARTDATE GT 19940101 &
>     AND CITY = "BOSTON" &
>     OR CITY = "OTTAWA"
```

In the next example, the records selected are those with a STARTDATE greater than January 1, 1994, and a CITY equal to BOSTON or OTTAWA:

```
> SELECT IF STARTDATE GT 19940101 &
>     AND (CITY = "BOSTON" &
>     OR CITY = "OTTAWA")
```

Modifying Simple and Compound Conditions

You can modify both simple and compound conditions with the keyword NOT. NOT means that the reverse of the condition must be true for the test to be satisfied. The syntax for a NOT condition is

[NOT] condition1 [{OR|AND} [NOT] condition2]...

Conditional Command List

The conditional command list can be used with the ACTIONMENU, KEY, and MENUITEM statement, the PUSH verb, and with the PRECOMMANDS and POSTCOMMANDS options of the DESIGNER procedure. The general form of the conditional command list is:

command-list [IF condition
[ELSE command-list| IF condition]...
[ELSE command-list]]

command-list

One or more commands separated by commas. A command list has the general form:

command [,command]...

For a list of the available commands, see (p. 334).

IF condition

The first command list is executed if the first condition is satisfied. A condition is a logical test that has the general form:

[NOT] condition [AND|OR [NOT] condition]...

ELSE command-list IF condition

If no previous condition is satisfied, then the next command list is executed if its condition is satisfied.

ELSE command-list

If no previous condition is satisfied, then the last command list is executed. If the specification is omitted, and none of the previous conditions are satisfied, then QDESIGN proceeds as though the NULL command option was present.

Conditions and NULL Values

The following standard truth tables show the results of Boolean operations, including those using null operands:

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

If any operand in an expression has a null value, the result is set to NULL. Whenever a conditional expression evaluates to NULL then PowerHouse treats it as "not true".

A NULL result is similar to a FALSE result, except that NOT NULL results in NULL, whereas NOT FALSE results in TRUE. For example, if ITEM1 is null, and you specify

```
> DEFINE SWITCH &
>   CHARACTER*3 = "YES" &
>     IF ITEM1=5 &
>     ELSE "NO "
```

when PowerHouse tests whether ITEM1 equals 5, the result is NULL. Similarly, if "IF NOT ITEM1=5" is used, the result is also NULL. In both cases, PowerHouse assigns the value "NO" to item SWITCH because the conditional test failed.

Conditions in SQL

SQL conditions can only be used on SQL statements. For example, conditions on the WHERE and HAVING options of query specifications are used to qualify or disqualify rows or groups of rows for subsequent processing. The rows (or groups) that qualify are those for which the condition evaluates to TRUE.

The operators used in sql-conditions are:

<	less than	>	greater than
<=	less than or equal to	>=	greater than or equal to
=	equal to	<>	not equal to

sql-expression operator {sql-expression|subquery}

The condition specifies the comparison used to qualify or disqualify rows for subsequent processing. The subquery specification is identical to a query-specification with two exceptions: the subquery must project a single-column table and the syntax of the subquery includes enclosing brackets.

```
>  SET LIST SQL
>  SQL DECLARE X CURSOR FOR SELECT * FROM PAYMENTS &
>  WHERE EXTRACT(MONTH FROM PAYMENT_DATE) = 04
```

sql-expression operator {ALL|SOME|ANY} subquery

The ALL option determines whether the condition is true for all values returned by the subquery.

The SOME or ANY options determine whether the condition is true for any (one or more) value returned by the subquery. When the = operator is used, the SOME or ANY options are equivalent to using IN subquery.

The subquery specification is identical to a query-specification with two exceptions: the subquery must project a single-column table and the syntax of the subquery must include enclosing brackets.

sql-expression [NOT]BETWEEN sql-expression AND sql-expression

The BETWEEN condition specifies a range of values used to qualify or disqualify rows for subsequent processing.

```
>  DECLARE X CURSOR FOR &
>      SELECT *  FROM PAYMENTS &
>      WHERE PAYMENT_AMT BETWEEN 15000 AND 20000
```

columnspec [NOT] LIKE 'sql-pattern' [ESCAPE 'character']

The LIKE condition is used for pattern-matching. The column specification must identify a column of type character. If the ESCAPE option is not used, characters within the pattern are interpreted as follows:

- The underscore (_) matches any single character (alphabetic, numeric or special).
- The percent sign (%) matches zero or more characters (alphabetic, numeric or special).
- All other characters match themselves.

The ESCAPE option indicates that the character immediately following the ESCAPE character in a pattern is interpreted as a regular character rather than a metacharacter. The ESCAPE character can be any character not used explicitly in your pattern.

Note that the metacharacters and the escape character used in SQL pattern matching are not the same as those used in PowerHouse pattern matching.

columnspec IS [NOT] NULL

The condition IS NULL must be used to determine whether a value is null or not null. When a value is compared to NULL using other conditions, the result is always NULL.

Testing for null in PowerHouse uses different syntax than testing for null in SQL.

sql-expression [NOT] IN (value, value[...])|subquery

Determines whether the expression is equal to any (one or more) of the values in the list, or values returned by the subquery.

The subquery specification is identical to a query-specification with two exceptions: the subquery must project a single-column table and the syntax of the subquery must include enclosing brackets.

To find information about a group of provinces:

```
>   SQL DECLARE X CURSOR FOR &
>      SELECT * FROM PROVINCES WHERE &
>      PROVINCE IN ('ONT', 'PQ', 'BC', 'PEI')
```

[NOT] EXISTS subquery

The EXISTS condition is FALSE if the subquery evaluates to an empty set; otherwise the condition is TRUE.

The subquery specification is identical to a query-specification with two exceptions: the subquery must either project a single-column table, or must be SELECT *. The syntax of the subquery must include enclosing brackets.

Expressions in PowerHouse

An expression consists of a term or combination of terms that yield a value. A term is one of the following:

- a string
- a number
- an item
- an expression
- a system function
- a function result
- a NULL value

There are six types of expressions: string, numeric, date, case, conditional, and SQL.

When you use an expression, the type must be consistent with the use of the value that the expression yields. For example, don't use a string expression to yield a value for a numeric item.

SQL expressions, including case-processing, use different syntax and are documented on (p. 303).

String Expressions

A string expression can be a single string or a series of terms that yield a string result. A string expression has the general form

termA [+ termB]...

The plus sign (+) concatenates the second string (termB) and the first string (termA) to make a single string. The length of the result is the sum of the length of the terms. Concatenation doesn't eliminate spaces.

For example, a string expression consisting of

```
"SMITH    " + "," + "JONES    "
```

results in the string:

```
"SMITH    ,JONES    "
```

To concatenate SQL string expressions, use the || operator (p. 303).

Numeric Expressions

A numeric expression can be a single number or series of terms that yield a numeric result. A numeric expression has the general form

termA [operator termB]...

The arithmetical operator in a numeric expression is one of the following:

+	addition	-	subtraction
*	multiplication	/	division
^	exponentiation (raising to a power)		

To ensure that special characters function as you intend, add a space in front of them when you use them as operators. For example, if you want to subtract the item UNITS from the item TOTAL, you must add a space in front of the operator, as in

```
TOTAL - UNITS
```

The precedence for processing operators in a numeric expression is

1. exponentiation
2. multiplication and division
3. addition and subtraction

PowerHouse evaluates operators that are at the same level of precedence from left to right. You can use parentheses to override this order of precedence.

PowerHouse uses 8-byte floating point to evaluate numeric expressions. PowerHouse converts all numeric values, except those already in floating point, to floating point prior to performing any calculations. After the calculation, the result is converted to the target item's datatype when the value is saved in the item buffer.

The number of significant digits available in 8-byte floating point varies with the hardware architecture. For more information, see "FLOAT Datatype" (p. 318). If the datatype of the target item is a fixed point datatype, the evaluation result is truncated and fractional values are lost. If you require the fractional values, multiple by an appropriate power of 10. Also, due to the nature of floating point, expression evaluation and truncation may result in a loss of precision in the ones digit. To avoid this, you can apply a rounding factor by adding 0.5 and using the FLOOR function as in

```
> DEFINE X INTEGER SIZE 4 = FLOOR((INPUT_VALUE_1 * INPUT_VALUE_2) + 0.5)
```

You can raise negative numbers to integer powers (positive or negative), but fractional powers (such as -10^1.5) result in an error condition. You can raise positive numbers to any power (positive, negative, or fractional).

A number divided by zero always results in zero and not in an error condition. However, an error occurs if you use zero, or an expression that results in zero, as a subscript when accessing an array in QDESIGN.

If possible, you should use multiplication instead of exponentiation. (For example, use n*n rather than n^2.) Exponentiation executes more slowly than multiplication and is also less accurate. Using a constant is faster than either multiplication or exponentiation.

Date Expressions

A date expression is similar to a numeric expression since you can manipulate it like a numeric expression, with one exception: the result of the date expression must yield either a six-digit date in YYMMDD format (010525) or an eight-digit date in YYYYMMDD format (20010525). If you include a century prefix in the date item, then specify the eight-digit format. If you exclude a century prefix in the date item, then specify the six-digit format.

Conditional Expressions

Conditional expressions use one or more of the basic types of expressions. They allow you to evaluate a series of expressions based on conditions. The general form is

expression [IF condition
[ELSE expression IF condition]...
[ELSE expression]]

For process efficiency, arrange the conditions in the order that they are most likely to be met.

If null value support is enabled and you omit the final ELSE expression option and none of the conditions are satisfied, PowerHouse sets the value to:

- zero, if the type is numeric or date and from a non-relational item
- spaces, if the type is character and a non-relational item
- null, if the type is a relational item

expression

A string, date, or numeric expression.

IF condition

If a condition is satisfied, then PowerHouse evaluates the first expression and assigns the resulting value.

ELSE expression IF condition...

If a previous condition hasn't been satisfied and if the next condition is satisfied, then PowerHouse evaluates the expression and assigns the resulting value.

ELSE expression

If a previous condition hasn't been satisfied, then PowerHouse evaluates the expression and assigns the resulting value.

Case Processing

Case processing compares the value of an item against a known value or series of values and performs actions based on the outcome of the comparison.

If there is a match in a DEFINE statement, the resulting value is assigned to the defined item. If there is no match, the specified default is assigned. If no default is specified, zeros, spaces, or nulls are assigned.

When a defined item value is calculated based on the value of only one item, and those values are known, case processing is more efficient than a conditional expression. The general form of case processing used for the DEFINE statement is

```
CASE [OF] item
    WHEN value-set|EXISTS|NULL|MISSING
        {THEN|:} value|NULL|MISSING
    [WHEN value-set|EXISTS|NULL|MISSING
        {THEN|:} value|NULL|MISSING]...
    [DEFAULT value|NULL|MISSING]
```

In the CHOOSE statement, the case compares the value of an item against a value or range of values and selects one expression-set to be used to determine the values for the linkitem. If there is no match, the specified default is used. If no default is specified, no records are chosen.

The general form of case processing used for the CHOOSE statement is

```
CASE [OF] item
    WHEN value-set|EXISTS|NULL|MISSING
        {THEN|:} expression-set|NULL|MISSING
    [WHEN value-set|EXISTS|NULL|MISSING
        {THEN|:} expression-set|NULL|MISSING]...
    [DEFAULT expression-set|NULL|MISSING]
```

Expressions in SQL

Expressions may be used within SQL statements

- as part of the specification of the project list within a query specification
- as part of the condition within WHERE and HAVING options
- in assignments of values to columns (as in the INSERT and UPDATE statements).

There are five types of SQL expressions: string, numeric, date, case, and conditional.

Terms in SQL expressions can be a columns, functions, SQL summary operations, SQL case-expressions, strings, numbers, NULL values, USER system variables, or SQL date-literals.

An sql-date-literal can be:

DATE 'yyyy-mm-dd'
INTERVAL [+|-] 'days[hours[:minutes[:seconds[.[hundredths]]]]]'
TIME 'hours:minutes:seconds[.[hundredths]]'
TIMESTAMP 'yyyy-mm-dd hours:minutes:seconds[.[hundredths]]'

Within SQL statements, data manipulation and SQL data manipulation functions may be used. For a list of these functions, see (p. 359), where functions that can be used within SQL statements are identified as DMF and SQL-DMF.

String Expressions

A string expression can be a single string or a series of terms that yield a string result. A string expression has the general form:

termA [|| termB]

The || operator concatenates string expressions in SQL.

Numeric-Expressions

A numeric-expression can be a single number or a series of terms that yield a numeric result. A numeric-expression has the general form:

termA [operator termB]

The arithmetic operators in an SQL numeric-expression are

+	addition	-	subtraction
*	multiplication	/	division

Date Expressions

A date-expression is similar to a numeric-expression since you can manipulate it like a numeric-expression. An SQL date expression yields a value of type DATE, INTERVAL, TIME, or TIMESTAMP.

SQL Case Processing

Two forms of SQL case-processing are used in PowerHouse.

CASE
WHEN sql-condition THEN expression|NULL
[[WHEN sql-condition THEN expression|NULL]...]
[ELSE expression|NULL]
END

For information about SQL conditions, see (p. 298).

An example of the preceding form of case-processing is:

```
>   SQL DECLARE X CURSOR FOR &
>       SELECT EMPLOYEENO, &
```

```
>      CASE WHEN CURRENCY = 'US' &
>             THEN CASHADVANCE*1.20 &
>             ELSE CASHADVANCE &
> END &
> FROM ACCOUNTBALANCE
```

CASE expression
WHEN expression THEN expression|NULL
[[WHEN expression THEN expression|NULL]...]
[ELSE expression|NULL]
END

An example of the preceding form of case-processing is:

```
>   SQL DECLARE Y CURSOR FOR &
>      SELECT EMPLOYEENO, &
>      CASE CURRENCY &
>      WHEN 'US' THEN CASHADVANCE*1.20 &
>      ELSE CASHADVANCE &
>   END &
>   FROM ACCOUNTBALANCE
```

The two examples produce the same result.

Expressions within Program Variables

In PowerHouse SQL, program variables are not limited to simple references to items. PowerHouse numeric, string, and date expressions may be used whenever program variables are valid in an SQL statement. Since these expressions are program variables, they must be preceded by a colon (:) to identify them as value provided by the application. The expressions may consist of references to items and functions that can be evaluated by PowerHouse.

Expressions not preceded by a colon are evaluated by the database. All the terms in these expressions must be known to the database for evaluation to occur.

SQL Summary Operations

Summary operations used in SQL are AVG, COUNT, MAX, MIN, and SUM.

SQL summary operations cannot be nested. Null values in the argument are eliminated before the functions are applied.

The default, ALL, indicates that duplicate values are included in the calculations. DISTINCT indicates that duplicate values are eliminated before the function is applied.

AVG({[ALL] expression}|{DISTINCT columnspec})

The argument must be numeric. If the argument is an empty set, the operation returns a null value.

COUNT{([ALL] expression)}|{(DISTINCT columnspec)}

Counts all non-null values.

COUNT(*)

Counts all rows in a table without eliminating duplicates. If the argument is an empty set, COUNT returns a value of zero.

MAX({[ALL] expression}|{DISTINCT columnspec})

If the argument is an empty set, the operation returns a null value.

MIN({[ALL] expression}|{DISTINCT columnspec})

If the argument is an empty set, the operation returns a null value.

SUM({[ALL] expression}|{DISTINCT columnspec})

The arguments must be numeric. If the argument is an empty set, the operation returns a null value.

Items and Datatypes in PowerHouse

An item is a location where PowerHouse can store data. PowerHouse supports the following types of items:

- defined
- global temporary (QTP)
- predefined (QDESIGN)
- record
- temporary

Defined Items

A defined item in PowerHouse is a designer-created item that doesn't exist in the data dictionary; it is declared with a DEFINE statement. A DEFINE statement either prompts the user for information at execution-time or assigns a name to an expression that it evaluates when the data required is available.

Global Temporary Items (QTP)

A global temporary item is a designer-created item that doesn't exist in the data dictionary. You declare it at the beginning of a QTP run with a GLOBAL TEMPORARY statement, and it remains in effect for the duration of the run in which you declared it.

Predefined Items (QDESIGN)

A predefined item is an item automatically defined by QDESIGN and used in screen processing.

The predefined items are FIELDTEXT, FIELDVALUE, OCCURRENCE, PATH, and SUBPATH.

FIELDTEXT

Contains the most recent set of characters entered by the screen user (or designer via procedure code) in a field on the screen or formatted for display on the screen. PowerHouse determines the size of the FIELDTEXT predefined item by its current contents. PowerHouse includes any trailing blanks in the size. The size is the number of characters entered.

FIELDVALUE

Contains the numeric value of the user's most recent entry into a numeric-type or date-type field. PowerHouse determines the internal representation of the predefined item FIELDVALUE (PACKED, ZONED, and so on) by the datatype of the item associated with that field.

Use the FIELDTEXT and FIELDVALUE predefined items in field-related procedures. The FIELDTEXT predefined item changes when PowerHouse displays or accepts any value.

OCCURRENCE [OF record-structure|item]

Returns the occurrence number of the currently active FOR control structure.

If there is no active FOR control structure, the value 1 is returned. The OF qualifier is optional and is used for documentation purposes only.

The FOR control structure sets the predefined item OCCURRENCE. This item controls which occurrence of repeating records, items, or clusters is addressed by other verbs. The current setting of OCCURRENCE can be addressed procedurally, although subscripting is not allowed. Outside the range of a FOR control structure, the value of OCCURRENCE is 1.

Field procedures invoked by field verbs from within a FOR control structure are considered to be within the range of the FOR control structure, and the current setting of OCCURRENCE is unchanged.

A higher-level screen can invoke a lower-level screen by passing one occurrence of a file or item. The lower-level screen can have an independent FOR control structure. Although the indicated occurrence of the passed file or item is passed to lower-level screens, OCCURRENCE itself is not. The value of OCCURRENCE on the lower-level screen is always 1 unless a FOR loop is active there.

Only one FOR control structure can be active at one time; FOR loop nesting is not allowed within a procedure. However, when an INTERNAL procedure is performed from within a FOR loop, it can itself have a FOR loop.

The setting of occurrence in such situations is described by the following example:

```
> FILE A DESIGNER OCCURS 10
.
.
.
> PROCEDURE INTERNAL SHOWLOOP
>   BEGIN
>     LET X OF A = OCCURRENCE
>     FOR A
>       BEGIN
>         LET X OF A = OCCURRENCE
>       END
>     LET X OF A = OCCURRENCE
>   END
> PROCEDURE ENTRY
>   BEGIN
.
.
.
>     FOR A
>       DO INTERNAL SHOWLOOP
.
.
.
>   END
```

In this example, the INTERNAL procedure SHOWLOOP is performed ten times. On the fifth time, the INTERNAL procedure sets the value of X to the following values:

5,1,2,3,4,5,6,7,8,9,10,5

At any time, there is only one setting of the predefined item, OCCURRENCE.

You cannot address all the occurrences of a repeating item within a repeating file on the same screen. Instead, you must pass each occurrence of the file in turn to a subscreen and process the repeating item there.

PATH

Contains a value set by the PATH procedure. This value directs the FIND procedure to one of several retrieval alternatives that you can use to retrieve a record. The PATH predefined item is set to zero at the start of Entry and Find mode initialization.

For information about the PATH procedure, see Chapter 4, "QDESIGN Procedures Overview", in the *QDESIGN Reference*.

SUBPATH

SUBPATH is used in QDESIGN to distinguish between generic and exact match retrievals since different cursors are opened in each case. It is used with the PATH predefined item to determine which cursor to open for retrieval.

Record Items

A record item is an item declared in the data dictionary. The general form is

item [(subscript)] [OF record-structure]

Use the OF record-structure qualifier to clarify the record-structure in which PowerHouse stores the item.

The subscript option applies only to items defined as repeating items. The subscript must be a numeric expression.

For information about items defined as repeating, see (p. 284).

Temporary Items

A temporary item in PowerHouse is a designer-created item that doesn't exist in the data dictionary. You declare it with a TEMPORARY statement. It can be used by the current screen or be passed to a lower-level screen. In QTP, a TEMPORARY can be used in the current request.

To enter negative values on a screen for temporary items, add LEADING SIGN or TRAILING SIGN options to the corresponding FIELD statements.

How QDESIGN Searches for Items

If an item is declared without an OF record-structure qualifier, QDESIGN searches for the item using the following steps:

1. QDESIGN assumes that the item is a temporary or defined item.
2. If the item is not a temporary or defined item, then QDESIGN searches the current record-structure from the last declared FILE.
3. If the item is not in the current record-structure, QDESIGN searches the assumed record-structure.
4. If the item is not in the assumed record-structure, QDESIGN assumes it is in any other record-structure where it uniquely exists.
5. If the item is not located by steps 1 to 4, QDESIGN issues an error message.

QDESIGN sets the current record-structure to blank at the beginning of a screen design.

- When you enter a FILE statement, the record-structure declared becomes the current record-structure.
- When you enter a DEFINE or TEMPORARY statement, PowerHouse resets the current record-structure to blank.
- When you enter any statement that affects the screen layout, PowerHouse resets the current record-structure to blank.

In QDESIGN, you can set the assumed record-structure using the SET ASSUMED statement. Otherwise, the assumed record-structure defaults to the PRIMARY file.

How QTP Searches for Items

If you declare an item without an OF record-structure qualifier, QTP searches for the item using the following steps:

1. QTP assumes that the item is a temporary or defined item.
2. If the item is not a temporary or defined item, then QTP assumes it is in the current record structure. The current record-structure is set as follows:
   - When you enter an ACCESS statement, the primary input record structure becomes the current record structure
   - When you enter an OUTPUT statement, the specified record structure becomes the current record structure.
   - When you enter a SUBFILE statement, the specified subfile becomes the current record structure. If an item that is included in the subfile exists in more than one file, it must be qualified or an error is issued. This typically occurs when two subfiles are created from the same input file. If both subfiles include an item with the same name, the name must be qualified when included in the second subfile.
   - When you enter a DEFINE or TEMPORARY statement, QTP sets the current record structure to blank.

3. If the item is not in the current record structure, QTP assumes it is in any other record structure where it uniquely exists.
4. If the item is not located by steps 1 to 3, QTP issues an error message.

Item Types

An item has element attributes such as type and size as well as its own attributes.

The element attribute type can be one of CHARACTER, NUMERIC or DATE. It is referred to in this discussion as item type. The element attribute size is used to specify the number of characters or digits allowed in the element. Typically, it is used to control the display size. It is referred to in this discussion as item size.

The item attributes datatype and datatype size determine the internal format PowerHouse uses to store the item.

Item Datatypes

The item datatype (distinct from the item type) determines the format that PowerHouse uses when it stores the item. PowerHouse determines the datatype of an item from the statement that defines the item.

The PowerHouse datatypes are listed in the following table:

Datatype	Item Type	Description
BLOB	CHARACTER, NUMERIC	Stores large objects whose structure is not known to the database.
CHARACTER	CHARACTER	Stores any ASCII character items in one character per byte.
DATE	DATE	Stores 6 to 16 digit dates.
DATETIME	DATE	Stores 10, 12, 14, 15 or 16 digit date and time as two 4-byte integers.
FLOAT[1]	DATE, NUMERIC	Stores numeric items as standard 32 or 64-bit floating point numbers, precision 16 digits. The internal format of floating point numbers is platform specific.
FREEFORM[1]	DATE, NUMERIC	Stores numeric items as a series of characters that forms a number.
G_FLOAT **(OpenVMS)**	DATE, NUMERIC	Stores numeric items as 64-bit floating-point number range $10^{+/-38}$, precision 15 digits.
INTEGER[1] SIGNED| UNSIGNED[2]	DATE, NUMERIC	Stores numeric items as signed or unsigned binary numbers in 1, 2, 4, 6, or 8 bytes.
INTERVAL	NUMERIC	Stores the difference between two datetimes as a 64-bit floating point number.
JDATE[3]	DATE	Stores dates in Julian date format in 2 bytes.
NUMERIC[4]	DATE, NUMERIC	Stores numeric items as 64-bit floating-point numbers.
PACKED[1] SIGNED| UNSIGNED	DATE, NUMERIC	Stores numeric items as signed or unsigned packed decimal numbers.

Datatype	Item Type	Description
PHDATE[5]	DATE	Stores dates in PowerHouse date format in 2 bytes.
VARCHAR	CHARACTER	Stores one or more characters preceded by an integer representing the current string length.
VMSDATE (**OpenVMS**)	DATE	Stores the standard eight-byte (64-bit) OpenVMS date.
ZDATE	DATE	Stores dates in encoded date format, stored in 6 bytes.
ZONED[1] SIGNED| UNSIGNED	DATE, NUMERIC	Stores numeric items as signed or unsigned zoned decimal numbers.
ZONED[1] NUMERIC (**OpenVMS**)	DATE, NUMERIC	Stores numeric items as signed zoned decimal numbers.

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

[2] *PowerHouse permits storage of only positive numbers in unsigned integer items. Because no bit is reserved for a sign, you can store values twice as large as the maximum allowed for signed integers.*

[3] *In JDATE format, the higher-order seven bits are the year; the least significant nine bits are the day.*

[4]*NUMERIC is only a valid item type for QDESIGN, QTP, and QUIZ.*

[5] *In PHDATE format, the higher-order seven bits are the year; the following four bits are the month, and the least significant five bits are the day.*

Item Sizes

All items also have a logical size (also known as the element size) and a storage size (also known as the item size).

The logical size determines how many characters from the item will be displayed. The storage size specifies how much space the item takes in memory or in a data file.

You must consider the item datatype and storage size when you define a dictionary to match existing data files or if you need to match data from non-PowerHouse programs.

If you give a date item a numeric datatype, you must make sure the item is large enough to hold the date. A date that includes a century in the year has a logical size of eight digits; a date that doesn't include a century has a logical size of six digits.

Datatype and size in QDESIGN, QUIZ, and QTP has the general form:

```
datatype [*n] [SIZE m]
```

where n is the maximum number of characters or digits in the item, and m is the storage size in bytes.

	Defaults if n and m not specified		Defaults for m if n specified		Defaults for n if m specified	
Datatype	**n**	**m**	**n**	**m**	**m**	**n**
CHARACTER	1	1	1 to 32,767	same	1 to 32,767	same
DATE	--	2 or 4	--	2 or 4	2 or 4	--

	Defaults if n and m not specified		Defaults for m if n specified		Defaults for n if m specified	
Datatype	**n**	**m**	**n**	**m**	**m**	**n**
DATETIME	--	always 8	--	always 8	always 8	--
FLOAT	6	4	1 to 6	4	4	6
			7 to 16	8	8	9
FREEFORM	6	6	1 to 31	same	1 to 31	same
G_FLOAT	9	8	1 to 16	always 8	always 8	9
INTEGER	4	2	1 to 4	2	2	4
			5 to 9	4	4	9
			10 to 14	6	6	14
			15 to 31	8	8	18
INTERVAL	--	always 4	--	always 4	always 4	--
JDATE	--	always 2	--	always 2	always 2	--
NUMERIC	6	always 8	1 to 16	always 8	always 8	6
PACKED	6	4	1 to 31	FLOOR(n/2) + 1	1 to 16	2m-1
PHDATE	--	always 2	--	always 2	always 2	--
VARCHAR	1	3	1 to 32765	n+2	1 to 32767	m-2
VMSDATE	--	always 8	--	always 8	always 8	--
ZDATE	8	6	8	6	6	8
ZONED	6	6	1 to 31	same	1 to 31	same

n specifies the maximum number of characters or digits in the item. It is not valid for date-type items.

m indicates the storage size in bytes.

Non-Relational PowerHouse Datatypes

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. In some cases, a mapping that works well in RMS ISAM may not work well with C-ISAM or DISAM.

For relational PowerHouse datatypes see (p. 313).

PH Datatype	Item Size[1]	C Type	FORTRAN Type	COBOL Type	IMAGE Type (MPE/iX)	OpenVMS Type	RMS Type (OpenVMS)	C-ISAM (UNIX), DISAM (Windows)
CHARACTER	1	char	CHARACTER *n	X(n)	Xn,Un	CHARACTER	STRING	CHARTYPE
DATE	4	n/a	n/a	n/a	n/a	n/a	n/a	n/a
DATETIME	8	n/a	n/a	n/a	n/a	n/a	n/a	n/a

PH Datatype	Item Size[1]	C Type	FORTRAN Type	COBOL Type	IMAGE Type (MPE/iX)	OpenVMS Type	RMS Type (OpenVMS)	C-ISAM (UNIX), DISAM (Windows)
FLOAT	4	float	REAL/REAL *4	COMP-1	R2	F_FLOAT	n/a	FLOATTYPE
FLOAT	8	double	DOUBLE PRECISION or REAL*8	COMP-2	R4	D_FLOAT	n/a	DOUBLETYPE
FREEFORM	n	n/a	n/a	n/a	n/a	n/a	n/a	n/a
G_FLOAT (**OpenVMS**)	8	n/a	REAL*8	n/a	n/a	G_FLOAT	n/a	n/a
INTEGER SIGNED	1	n/a	n/a	n/a	n/a	INTEGER BYTE	n/a	n/a
INTEGER SIGNED	2	short	INTEGER*2	S9(4) COMP	I1,J1	INTEGER WORD	INT2	INTTYPE
INTEGER SIGNED	4	int	INTEGER*4	S9(9) COMP	I2,J2	INTEGER LONGWORD	INT4	LONGTYPE
INTEGER SIGNED	6	n/a	n/a	n/a	I3,J3	n/a	n/a	n/a
INTEGER SIGNED	8	n/a	n/a	S9(18) COMP	I4,J4	INTEGER QUADWORD	INT8	n/a
INTEGER UNSIGNED	1	n/a	LOGICAL*1	n/a	n/a	INTEGER BYTE	n/a	n/a
INTEGER UNSIGNED	2	unsigned short	LOGICAL*2	9(4) COMP	K1	INTEGER WORD	BIN2	n/a
INTEGER UNSIGNED	4	unsigned long	LOGICAL*4	9(9) COMP	K2	INTEGER LONGWORD	BIN4	n/a
INTEGER UNSIGNED	6	n/a	n/a	n/a	K3	n/a	n/a	n/a
INTEGER UNSIGNED	8	n/a	n/a	9(18) COMP	K4	INTEGER QUADWORD	BIN8	n/a
INTERVAL	8	n/a	n/a	n/a	n/a	n/a	n/a	n/a
JDATE	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a
PACKED SIGNED	n	n/a	n/a	S9(n) COMP-3	Pn	PACKED DECIMAL	DECIMAL	n/a
PACKED UNSIGNED	n	n/a	n/a	9(n) COMP-3	n/a	PACKED DECIMAL	DECIMAL	n/a
PHDATE	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a
VARCHAR	n	n/a	n/a	n/a	n/a	VARYING CHAR	n/a	n/a
VMSDATE (**OpenVMS**)	8	n/a	n/a	n/a	n/a	ABS DATE & TIME	n/a	n/a
ZDATE	6	n/a	n/a	n/a	n/a	n/a	n/a	n/a
ZONED SIGNED	n	n/a	n/a	S9(n) DISPLAY	Zn	ZONED OVERPUNCHED	STRING	n/a
ZONED UNSIGNED	n	n/a	n/a	9(n) DISPLAY	n/a	ZONED OVERPUNCHED	STRING	n/a
ZONED NUMERIC **(OpenVMS)**	n	n/a	n/a	n/a	n/a	ZONED NUMERIC	STRING	n/a

PH Datatype	Item Size[1]	C Type	FORTRAN Type	COBOL Type	IMAGE Type (MPE/iX)	OpenVMS Type	RMS Type (OpenVMS)	C-ISAM (UNIX), DISAM (Windows)

[1] *Item size is in bytes.*

Symbols

n/a *Not applicable. No direct equivalent exists for this PowerHouse storage type.*
n *Any number.*

Relational PowerHouse Datatypes (Part 1)

For more relational PowerHouse datatypes (Part 2), see(p. 315).

For non-relational PowerHouse datatypes, see (p. 311).

PH Datatype	Item Size[1]	ORACLE (OpenVMS, UNIX, Windows)	SYBASE (UNIX, Windows)	DB2 (UNIX, Windows)	SQL SERVER (Windows)	ODBC (UNIX, Windows)
blob[2]	n	BFILE, BLOB, CLOB, LONG, LONG RAW, NCLOB	IMAGE, TEXT	BLOB, CLOB, LONG VARCHAR	IMAGE, NTEXT(n) 255<=n, TEXT	SQL_LONGVARBINARY, SQL_LONGVARCHAR, SQL_WLONGVARCHAR
CHARACTER	n	CHAR(n), NCHAR(n), ROWID	BINARY, CHAR, NCHAR, NVARCHAR, TIMESTAMP, UNICHAR, UNIVARCHAR, VARBINARY, VARCHAR	CHAR(n), GRAPHIC	BINARY, CHAR, NCHAR, TIMESTAMP, UNIQUEIDENTIFIER	SQL_BINARY, SQL_CHAR, SQL_GUID, SQL_NCHAR, SQL_WCHAR
DATE	4	n/a	n/a	DATE	n/a	n/a
DATETIME	8	DATE	DATETIME, SMALLDATETIME	TIMESTAMP	DATETIME, SMALLDATETIME	SQL_TYPE_DATE, SQL_TYPE_TIMESTAMP
FLOAT	4	n/a	FLOAT(n) n<=15, REAL	FLOAT(n) 1<=n<=21, REAL	REAL	SQL_REAL
FLOAT	8	NUMBER no precision or scale	DOUBLE PRECISION, FLOAT no n, FLOAT(n) 16<=n	DOUBLE, FLOAT no n, FLOAT(n) 22<=n	FLOAT	SQL_DOUBLE, SQL_FLOAT
FREEFORM	n	n/a	n/a	n/a	n/a	n/a
G_FLOAT (**OpenVMS**)	8	n/a	n/a	n/a	n/a	n/a
INTEGER SIGNED	1	n/a	n/a	n/a	n/a	n/a
INTEGER SIGNED	2	NUMBER(p[,s]) 1<=p<=4	BIT, SMALLINT, TINYINT	SMALLINT	BIT, SMALLINT, TINYINT	SQL_BIT, SQL_SMALLINT, SQL_TINYINT
INTEGER SIGNED	4	NUMBER(p[,s]) 5<=p<=9	INTEGER, SMALLMONEY	INTEGER	INT	SQL_INTEGER

PH Datatype	Item Size[1]	ORACLE (OpenVMS, UNIX, Windows)	SYBASE (UNIX, Windows)	DB2 (UNIX, Windows)	SQL SERVER (Windows)	ODBC (UNIX, Windows)
INTEGER SIGNED	6	n/a	n/a	n/a	n/a	n/a
INTEGER SIGNED	8	NUMBER(p[,s]) 10<=p<=18	MONEY	n/a	BIGINT	SQL_BIGINT
INTEGER UNSIGNED	1	n/a	n/a	n/a	n/a	n/a
INTEGER UNSIGNED	2	n/a	n/a	n/a	n/a	n/a
INTEGER UNSIGNED	4	n/a	n/a	TIME[3]	n/a	n/a
INTEGER UNSIGNED	6	n/a	n/a	n/a	n/a	n/a
INTEGER UNSIGNED	8	n/a	n/a	n/a	n/a	n/a
INTERVAL	8	n/a	n/a	n/a	n/a	n/a
JDATE	2	n/a	n/a	n/a	n/a	n/a
PACKED SIGNED	n	NUMBER(p,s) 19<=p<=38 NUMBER(p) 10<=p, s=0	DECIMAL(p,s), NUMERIC(p,s)	DECIMAL(p,s), NUMERIC(p,s)	DECIMAL, MONEY, NUMERIC, SMALLMONEY,	SQL_DECIMAL, SQL_NUMERIC
PACKED UNSIGNED	n	n/a	n/a	n/a	n/a	n/a
PHDATE	2	n/a	n/a	n/a	n/a	n/a
VARCHAR	n	INTERVAL DAY TO SECOND, INTERVAL YEAR TO MONTH, NVARCHAR2(n), RAW, TIMESTAMP WITH [LOCAL] TIMEZONE, VARCHAR(n), VARCHAR2(n)	n/a	VARCHAR(n)	VARBINARY, NTEXT(n) 1<=n<=254, NVARCHAR, VARCHAR	SQL_VARBINARY , SQL_VARCHAR, SQL_WLONGVARC HAR, SQL_WVARCHAR
VMSDATE (**OpenVMS**)	8	n/a	n/a	n/a	n/a	n/a
ZDATE	6	n/a	n/a	n/a	n/a	n/a
ZONED SIGNED	n	n/a	n/a	n/a	n/a	n/a
ZONED UNSIGNED	n	n/a	n/a	n/a	n/a	n/a
ZONED NUMERIC **(OpenVMS**)	n	n/a	n/a	n/a	n/a	n/a

[1] *Item size is in bytes.*

[2] *PowerHouse treats these datatypes as blobs internally.*

[3] *The TIME datatype is mapped to an INTEGER UNSIGNED SIZE 4.*

Symbols

n/a Not applicable. No direct equivalent exists for this PowerHouse storage type.
n Any number.
p Precision.
s Scale.

Relational PowerHouse Datatypes (Part 2)

For more relational PowerHouse datatypes (Part 1), see (p. 313).

For non-relational PowerHouse datatypes, see (p. 311).

PH Datatype	Item Size[1]	Stored Procedure Result Set	ORACLE Rdb SQL (OpenVMS)	ORACLE Rdb RDO (OpenVMS)	CDD/ REPOSITORY (OpenVMS)	ALLBASE/SQL (MPE/iX)
blob[2]	n	LONGVARBINARY, LONGVARCHAR	LIST OF BYTE VARYING	SEGMENTED_ STRING	SEGMENTED STRING	LONGBINARY, LONGVAR-BINARY
CHARACTER	n	BINARY, CHARACTER	CHAR, NCHAR	TEXT	TEXT	BINARY, CHAR
DATE	4	DATE	n/a	n/a	n/a	DATE
DATETIME	8	DATETIME	DATE, TIMESTAMP	n/a	n/a	DATETIME
FLOAT	4	FLOAT	FLOAT(n) 1<=n<=24, REAL	F_FLOATING	F_FLOATING	FLOAT(n) 1<=n<=24, REAL
FLOAT	8	DOUBLE	DECIMAL(n) 19<=n, DOUBLE PRECISION, FLOAT no n, FLOAT(n) 25<=n<=53 NUMERIC(n) 19<=n	n/a	D_FLOATING	DOUBLE PRECISION, FLOAT no n, FLOAT(n) 25<=n<=53
FREEFORM	n	n/a	n/a	n/a	n/a	n/a
G_FLOAT (**OpenVMS**)	8	n/a	n/a	G_FLOATING	G_FLOATING	n/a
INTEGER SIGNED	1	n/a	n/a	SIGNED BYTE	SIGNED BYTE	n/a
INTEGER SIGNED	2	SMALLINT	DECIMAL(n) 1<=n<=4, NUMERIC(n) 1<=n<=4, SMALLINT, TINYINT	SIGNED WORD	SIGNED WORD	SMALLINT
INTEGER SIGNED	4	INTEGER	DECIMAL(n) 5<=n<=9, INTEGER, NUMERIC(n) 5<=n<=9	SIGNED LONGWORD	SIGNED LONGWORD	INTEGER
INTEGER SIGNED	6	n/a	n/a	n/a	n/a	n/a
INTEGER SIGNED	8	QUADWORD	DECIMAL(n) 10<=n<=18, BIGINT, NUMERIC(n) 10<=n<=18	SIGNED QUADWORD	SIGNED QUADWORD	n/a
INTEGER UNSIGNED	1	n/a	n/a	n/a	UNSIGNED BYTE	n/a
INTEGER UNSIGNED	2	n/a	n/a	n/a	UNSIGNED WORD	n/a
INTEGER UNSIGNED	4	TIME	TIME[3]	n/a	UNSIGNED LONGWORD	TIME[3]

PH Datatype	Item Size[1]	Stored Procedure Result Set	ORACLE Rdb SQL (OpenVMS)	ORACLE Rdb RDO (OpenVMS)	CDD/ REPOSITORY (OpenVMS)	ALLBASE/SQL (MPE/iX)
INTEGER UNSIGNED	6	n/a	n/a	n/a	n/a	n/a
INTEGER UNSIGNED	8	n/a	n/a	n/a	UNSIGNED QUADWORD	n/a
INTERVAL	8	INTERVAL	INTERVAL	n/a	n/a	INTERVAL
JDATE	2	n/a	n/a	n/a	n/a	n/a
PACKED SIGNED	n	DECIMAL, MONEY, NUMERIC, SMALLMONEY	n/a	n/a	PACKED DECIMAL	DECIMAL(p[,s]), NUMERIC(p[,s])
PACKED UNSIGNED	n	n/a	n/a	n/a	PACKED DECIMAL	n/a
PHDATE	2	n/a	n/a	n/a	n/a	n/a
VARCHAR	n	LONGVARCHAR, VARBINARY, VARCHAR	LONG VARCHAR, NCHAR VARYING, VARCHAR	VARYING STRING	VARYING STRING	VARBINARY, VARCHAR
VMSDATE (**OpenVMS**)	8	n/a	DATE VMS	DATE	DATE	n/a
ZDATE	6	n/a	n/a	n/a	n/a	n/a
ZONED SIGNED	n	n/a	n/a	n/a	RIGHT OVERPUNCHED NUMERIC	n/a
ZONED UNSIGNED	n	n/a	n/a	n/a	RIGHT OVERPUNCHED NUMERIC	n/a
ZONED NUMERIC **(OpenVMS)**	n	n/a	n/a	n/a	ZONED NUMERIC	n/a

[1] *Item size is in bytes.*

[2] *PowerHouse treats these datatypes as blobs internally.*

[3] *The TIME datatype is mapped to an INTEGER UNSIGNED SIZE 4.*

Symbols

n/a *Not applicable. No direct equivalent exists for this PowerHouse storage type.*
n *Any number.*
p *Precision.*
s *Scale.*

Relational Datatypes Specifics

TIME Datatypes

PowerHouse internally sees many relational TIME datatypes as unsigned integers that have the value HHMMSSTTT. Depending on the database, the thousandths of a second may or may not be zero. In other words, TIME is seen as a 9-digit integer but the last three digits may always be 0. For example, in ALLBASE/SQL, the thousandths are always 0. To display and enter data into an ALLBASE/SQL TIME column, the following FIELD statement is useful:

```
FIELD TIME_COLUMN INPUT SCALE 3 OUTPUT SCALE -3
```

You may optionally add `PICTURE "^^:^^:^^" SIGNIFICANCE 8 VALUES 0 TO 235959`.

To store SYSTIME into an ALLBASE/SQL TIME column, drop the fraction of a second:

```
LET <Name> = 1000 * FLOOR(SYSTIME/100)
```

Unsupported DB2 Datatypes

The following DB2 datatypes are not supported by PowerHouse: DBCLOB and LONG VARGRAPHIC.

ODBC Datatypes

The PowerHouse datatype may change depending on the size of the ODBC column. For example, an SQL_LONGVARCHAR of size 255 is considered a VARCHAR in PowerHouse, but if the size is 32K then it is treated as a BLOB.

An ODBC data type is an intermediate data type that is assigned by the ODBC driver for a given database data type. Then the ODBC datatype is converted by PowerHouse to the PowerHouse datatype. For example: A Microsoft SQL Server MONEY data type is translated by the ODBC driver into an SQL_DECIMAL datatype. SQL_DECIMAL is translated into the PowerHouse PACKED datatype.

A Microsoft SQL Server TIMESTAMP datatype, which is not a PowerHouse datetime value, is translated by the ODBC driver into SQL_BINARY. SQL_BINARY is translated into a PowerHouse CHARACTER datatype.

If you are not sure what the ODBC data source column type has been mapped to in PowerHouse, you can use the QSHOW SHOW RECORD statement to describe the table layout.

BLOB Datatype

A BLOB datatype stores large objects whose structure is not known to the database. The item types for a BLOB datatype are CHARACTER (C) and NUMERIC (N). BLOB datatypes are not applicable for non-relational databases.

Any blob content can be used internally in QUICK screens, QTP runs or QUIZ reports. You can concatenate any blob to a string, or to another blob, and assign the result to a blob. However, the display of blobs is limited by the blob contents. PowerHouse best supports text blobs as fields on QUICK screens or as QUIZ report items. PowerHouse supports binary and text blobs.

You may use the DO BLOB verb to call external utilities such as editors.

All PowerHouse components support the following operations:

- assign contents of a blob to a string
- assign a string to a blob (creating a new blob)
- assign the contents of one blob to another blob field (in the same or a different database)
- concatenate a blob and a string (the result is a blob or string)

For more information about blob support, see (p. 345).

OpenVMS: Blobs in PowerHouse 4GL on the host are limited to 32,767 bytes.

CHARACTER Datatype

PowerHouse stores one character per byte using the native character set of the host system.

DATE Datatype

Dates with the century included are handled as 8-digits numbers representing YYYYMMDD in an INTEGER UNSIGNED SIZE 4 datatype. Dates with the century excluded are handled as 6-digits representing YYMMDD encoded in a PHDATE datatype. For more information, see the INTEGER Datatype and PHDATE Datatype sections on (p. 320) and (p. 322), respectively.

Limit: This datatype can only be used as a type for the DEFINE, GLOBAL TEMPORARY, and TEMPORARY statements in either QDESIGN, QUIZ, or QTP.

DATETIME Datatype

DATETIME is stored as eight bytes where the first four bytes represent the date portion and the last four bytes represent the time portion. Since the century is always included in the date, the date portion is an eight-digit number. The time portion is stored as a nine-digit number.

For example, 1991/12/13 23:59:59.99 is stored in eight bytes where the value of the first four bytes as an integer is 19911213 (date portion) and the value of the last four bytes as an integer is 235959990 (time portion).

DATETIME is valid for DATETIME items in ALLBASE/SQL.

FLOAT Datatype

PowerHouse stores and manipulates items of FLOAT datatype as follows:

	Syntax	Description
MPE/iX:	[IEEE|NONIEEE] FLOAT[SIZE 4|8]	four or eight-byte, IEEE or NONIEEE floating point numbers. IEEE format (which is governed by an industry standard) is the native floating point format for MPE/iX. NONIEEE format is provided for compatibility with MPE/V machines.
OpenVMS:	FLOAT[SIZE 4|8]	four or eight-byte, floating point numbers Default: D_FLOAT
UNIX, Windows:	FLOAT[SIZE 4|8]	four or eight-byte, floating point numbers

For additional **MPE/iX** information, see (p. 319).

For additional **OpenVMS** information, see (p. 319).

For additional **UNIX/Windows** information, see (p. 320).

For elements defined in the dictionary with a display size of more than nine digits, PowerHouse assigns a default datatype of FLOAT SIZE 8 rather than INTEGER SIZE 6 or INTEGER SIZE 8. This is because PowerHouse converts item values to FLOAT SIZE 8 before performing any calculations.

Although some other datatypes are able to represent numbers greater than 16 digits without loss of precision, significant information can be lost when converting from a more precise datatype to FLOAT SIZE 8.

Advantages: Stores very large or very small numbers without losing significant information.

PowerHouse converts all numeric values to float size 8 before performing calculations. Float datatypes do not have to be converted.

Disadvantages: Floating point storage does not always represent fractions or very large numbers exactly because of hardware restrictions. For example, 0.20 may be represented as 0.1999... This loss of precision means that you should never test to see if an item is strictly equal to a constant. Instead, test to see if the value of the floating point item is within a small range of the constant. For example,

```
> ...IF FLOAT_NUM LE 2.1 AND
>       FLOAT_NUM GE 1.9
```

The internal representation of all monetary amounts should be scaled consistently, regardless of datatype. Therefore, if a monetary amount is declared as a datatype FLOAT, it should be given an INPUT SCALE of 2(10^2=100), which is the same input scale given to monetary amounts that are stored as integers.

The combination of format and size determines the range of values that may be represented without loss of precision or significance.

In floating point representation, numbers are represented by an exponent and a mantissa. The exponent represents the magnitude of a floating number (digits to the right). The mantissa represents the significant digits in a floating point number (digits to the left).

In a floating point datatype, a certain number of bits are used to represent the mantissa and remaining bits are used to represent the exponent. The number of bits used in each case determines the range of the mantissa and the exponent.

Loss of precision occurs when a number gets so large that digits on the right become inaccurate. Loss of significance occurs when a number gets so large that digits on the left are lost.

MPE/iX

The table below summarizes the characteristics of the four floating point types available on MPE/iX machines.

Format	Exponent (bits)	Mantissa	Smallest (value)	Largest (value)	Significance (digits)
Float 4 IEEE	8	23	1.1e-38	3.4e38	7
Float 4 NONIEEE	9	22	8.6e-78	1.1e77	6
Float 8 IEEE	11	52	2.2e-308	1.7e308	15
Float 8 NONIEEE	9	54	8.6e-78	1.7e308	16

PowerHouse running in Native mode will support all four floating point formats as item syntax. The default format is IEEE. If neither IEEE or NONIEEE is specified, a default format is taken from the dictionary options.

A default format may be declared in the dictionary. If no default format is specified, IEEE is assumed. The designer may change this default to NONIEEE in the SYSTEM OPTIONS statement.

If your application is running on a Compatibility Mode dictionary, the default format will be NONIEEE. There is no supported mechanism for using IEEE FLOAT formats.

OpenVMS

PowerHouse stores items of FLOAT datatype as either D or F floating point numbers. A PowerHouse float item can have a storage size of four or eight-bytes. A four-byte FLOAT item is represented by a VAX F_FLOAT item. You can optionally specify a G_FLOAT item.

Default: D_FLOAT

Advantages: Stores very large or very small numbers without losing significant information. For example, 198,753,481,513 is stored in a four-byte float as roughly 1.98753×10^{11}. Although precision (the digits to the right of the number) is lost, significance (the digits to the left) is not.

PowerHouse does not have to convert float items for calculations and comparisons because all values are converted to D_FLOAT floats before any calculations are performed.

Floating Point Compatibility on OpenVMS

PowerHouse uses the D-Floating data type to manipulate numerical data internally. The VAX D_Float precision is 56 bits. The AXP D_Float precision is 53 bits. Since the D_Floating data type is not fully supported on AXP, some loss of precision may occur. Where PowerHouse is precise to 16 digits, some values on AXP may get rounded. For example,

```
> SCREEN DATATYPE
> DEFINE MYVAL NUM*16 = 9999999999999999
> FILED MYVAL DISPLAY PREDISPLAY
> GO
```

The value for MYVAL differs on VAX and AXP. On AXP, MYVAL contains the value 10000000000000000 (1e+16) which does not fit in the display field.

UNIX, Windows

The table below summarizes the characteristics of the two floating point types available on UNIX and Windows machines.

Format	Exponent (bits)	Mantissa	Smallest (value)	Largest (value)	Significance (digits)
Float 4	8	23	1.1e-38	3.4e38	7
Float 8	11	52	2.2e-308	1.7e308	15

Range Overflow

Converting between floating point types can result in loss of precision, due to rounding.

Loss of precision can be detected by converting the source type to the target type, and then back to the source type. If the original and final values differ, then precision has been lost.

This procedure can become more complicated because the most precise FLOAT format available to PowerHouse for internal computations is FLOAT 8. The CHARACTERS function may be of use when the user wants to do comparisons on float values.

FREEFORM Datatype

PowerHouse stores items of FREEFORM datatype as a string of characters that forms a number. The FREEFORM datatype allows PowerHouse to read edited or formatted data from other systems. The FREEFORM datatype is not recommended in other cases and should not be used for segments in indexes.

Advantages: Allows you to read data from other systems.

Disadvantages: Not compact when stored in a data file; PowerHouse must convert FREEFORM items to FLOAT for calculations and comparisons. Cannot be used for index segments.

How PowerHouse Stores FREEFORM Values

PowerHouse stores FREEFORM values as whole numbers with no fractional portion. PowerHouse stores negative FREEFORM values with leading minus signs, whether or not you specify a leading or trailing sign for display purposes. However, PowerHouse will accept a fractional value as part of a FREEFORM number if it follows a decimal character.

Leading and Trailing Signs in FREEFORM Items

On input, a leading or trailing plus sign (+), leading or trailing spaces, and an embedded decimal character (by default, a period) are allowed in a FREEFORM item. A FREEFORM item accepts a negative value when the LEADING SIGN or TRAILING SIGN format-options are used and the specified PICTURE string includes corresponding spaces to contain the sign. For such items, a leading or trailing minus sign is allowed on input, and the sign is then displayed as specified by the PICTURE string.

Examples of FREEFORM Numbers

PowerHouse can read the following valid FREEFORM numbers:

123	-123
123-	+123
123.45	-123.45

INTEGER Datatype

PowerHouse stores items of INTEGER datatype as binary numbers. PowerHouse integers can have a storage size of one to eight bytes. The number is stored in 2's-complement representation.

An integer datatype can be signed or unsigned. For example, signed integers of size 2 can contain numbers between -32,768 and 32,767. Unsigned integers of size 2 can contain numbers between 0 and 65,535.

Advantages: Compact.

Disadvantages: PowerHouse must convert INTEGER values to FLOAT for calculations and comparisons.

INTERVAL Datatype

In PowerHouse, an INTERVAL item is manipulated as a floating point decimal number, with the fractional part representing time.

To display an INTERVAL item correctly, a picture which scales the value of the interval is needed. An input scale is not used, so on input an explicit decimal point separates the days and time fields.

When entering data, you can enter a four-digit day value followed by a decimal of up to eight digits. Entering 23595999 displays 23:59:59.990. If an input scale of -8 is used, the entire eight-digit time field must be entered.

JDATE Datatype

The storage size of a JDATE (Julian date) item is two bytes. The first seven bits represent the year and the last nine bits represent the day of the year (from 1 to 366).

To specify a century-excluded date in PDL, use DATE SIZE 6 on the ELEMENT or USAGE statement. To specify a century-excluded defined, temporary, or global temporary JDATE item, use JDATE CENTURY EXCLUDED or simply JDATE. The default is CENTURY EXCLUDED regardless of the CENTURY system option in the dictionary. A century excluded JDATE item has year values from 00 to 99. PowerHouse assumes that the date is within the default century.

If you want to define an item as type JDATE CENTURY EXCLUDED and derive its value from an eight-digit date, you must use the REMOVECENTURY function, as in

```
> DEFINE LONGDATE DATE CENTURY INCLUDED = 19990925
> DEFINE JULIANDATE JDATE = REMOVECENTURY(LONGDATE)
```

A JDATE item can derive its value directly from a six-digit date:

```
> DEFINE SHORTDATE DATE CENTURY EXCLUDED = 990431
> DEFINE JULIANDATE JDATE = SHORTDATE
```

To specify a century-included date in PDL, use DATE SIZE 8 on the ELEMENT or USAGE statement. To specify a century-included defined, temporary, or global temporary JDATE item, use JDATE CENTURY INCLUDED. For century-included JDATE items, the first seven bits represent the number of years after the year 1900, so the years from 1900 to 2027 may be represented.

In spite of the way JDATE is stored, it is processed internally as a six-digit number in the order YYMMDD or an eight-digit number in the order YYYYMMDD.

NUMERIC Datatype

NUMERIC [*n] is always handled as a FLOAT SIZE 8.

MPE/iX: If the dictionary SYSTEM OPTIONS is set to NOIEEE, then NUMERIC is set to FLOAT SIZE 4.

For more information about the FLOAT datatype, see (p. 309).

Note: This datatype can only be used as a type for the DEFINE, GLOBAL TEMPORARY, and TEMPORARY statements in one of QDESIGN, QUIZ, or QTP. It is not a valid datatype for PDL.

PACKED Datatype

PowerHouse stores items of PACKED datatype as packed decimal numbers. If the element size of a packed item is n, the storage size is

```
Floor(n/2) + 1
```

Packed items can have element sizes of 1 to 31 digits, and storage sizes of 1 to 20 bytes.

The byte that contains the least significant digit also contains the sign.

Each byte of a packed item contains two digits, except the lower-order bit on the last byte, which contains one digit and the item's sign. Thus, a six-byte PACKED item can contain 11 digits (6x2 =12, less one for the sign).

If datatype PACKED is specified without the SIGNED or UNSIGNED option, the default is PACKED SIGNED. The sign nibble of a PACKED item is the low-order nibble of the low-order byte. For a PACKED SIGNED item, a positive number is explicitly indicated by a hexadecimal C in the sign nibble.

When PowerHouse writes PACKED UNSIGNED items to a data file, it puts hexadecimal F in the sign field for positive values and hexadecimal D in the sign field for negative values.

Advantages: Compact; holds up to 31 digits.

Disadvantages: PowerHouse must convert PACKED items to FLOAT for calculations and comparisons. Large numbers (that is, those with more than 16 digits) lose precision when converted by PowerHouse for calculations.

PHDATE Datatype

The storage size of a PHDATE item is two bytes. The first seven bits are used for the year, the next four bits for the month, and the last five bits for the day.

To specify a century excluded date in PDL, use DATE SIZE 6 on the ELEMENT or USAGE statement. To specify a century-excluded defined, temporary, or global temporary PHDATE item, use PHDATE CENTURY EXCLUDED or simply PHDATE. The default is CENTURY EXCLUDED regardless of the CENTURY system option in the dictionary. A century excluded PHDATE item has year values from 00 to 99. PowerHouse assumes that the date is within the default century.

To specify a century-included date in PDL, use DATE SIZE 8 on the ELEMENT or USAGE statement. To specify a century-included defined, temporary, or global temporary PHDATE item, use PHDATE CENTURY INCLUDED. For century-included PHDATE items, the first seven bits represent the number of years after the year 1900, so the years from 1900 to 2027 may be represented.

Regardless of the way PHDATE is stored, it is processed internally as a six-digit number in the order YYMMDD or an eight-digit number in the order YYYYMMDD.

VARCHAR Datatype

A VARCHAR item has a maximum string length, but a variable storage length. A two-byte integer representing the current string length precedes the string. Trailing blanks entered into a field defined as VARCHAR are accepted as part of the entry; that is, they are significant. VARCHAR is valid for defined items, and for columns in relational tables.

Advantages: VARCHAR retains its actual size when added and stored.

Disadvantages: VARCHAR occupies space up to the item size plus two bytes in a non-relational database.

VMSDATE Datatype (OpenVMS)

The item datatype, VMSDATE, is an eight-byte date with a format specified by the data dictionary date picture in effect. The value stored represents the number of 100-nanosecond units of time elapsed from the base date (00:00:00:00 November 17, 1858). It can display either a two or four-digit year that is manipulated in the format YYMMDD or YYYYMMDD respectively. VMSDATE allows you to use the OpenVMS system date.

ZDATE Datatype

The ZDATE datatype is a century-included date. It is always SIZE 6. For example, the definition of a ZDATE element would be:

```
ELEMENT DATEFIELD DATE SIZE 8 DATATYPE ZDATE SIZE 6
```

Any other size is invalid, therefore the size specification on the datatype is optional. A ZDATE is represented by six alpha-numeric characters. The first character represents the first three digits of the century-included year and the remaining characters represent the rest of the numbers in the date—the last digit of the year, the month, and the day.

The century-included year is represented as follows:

First Character	Century Represented (first three digits)
0	190
1	191
2	192
…	
9	199
A	200
B	201
C	202
…	
Z	225

For example, 991231 is the internal representation for the century-included date 1999/12/31. The initial "9" represents "199". The remaining digits, "91231", represents the rest of the numbers in the date - the final digit in the year, the month and the day. C91025 represents 2029/10/25.

ZONED Datatype

PowerHouse stores items of datatype, ZONED, as zoned decimal numbers. The storage size of a zoned item is the same as the element size. Zoned items can be 1 to 31 digits.

Each byte of a zoned item contains one digit. The low-order byte also contains the sign (positive or negative), encoded in the high-order four bits.

Zoned items may be signed or unsigned; both can contain positive and negative values. PowerHouse displays positive values of unsigned items with no leading sign.

If datatype ZONED is specified without the SIGNED or UNSIGNED or NUMERIC options, the default option is ZONED UNSIGNED. Both ZONED SIGNED and ZONED UNSIGNED items can store positive or negative values. The sign of a ZONED item is an "overpunch" on the lower-order byte (which also contains the least significant digit).

The lower-order byte is represented by a character as follows:

	Positive Values ending in 0, 1 - 9	Negative Values ending in 0, 1 - 9
ZONED UNSIGNED	0 1 - 9	}, J - R
ZONED SIGNED	{, A - I	}, J - R

	Positive Values ending in 0, 1 - 9	Negative Values ending in 0, 1 - 9
ZONED NUMERIC (OpenVMS)	0 1 - 9	P q - y

A through I represents positive 1 through 9 respectively for ZONED SIGNED; J through R represent negative 1 through 9 for datatypes ZONED SIGNED and ZONED UNSIGNED; lowercase q through y represent negative 1 through 9 for datatype ZONED NUMERIC (**OpenVMS**).

Advantages: Holds up to 31 digits. Because ZONED UNSIGNED numbers have the same internal requirements as character items, they are useful for redefining character fields. They can also be used for segmented indexes.

Disadvantages: Not very compact; PowerHouse must convert ZONED items to FLOAT for calculations and comparisons.

Compatibility with Other Languages

Although the exact representation specified for zoned items is unimportant in a new application, you must specify the correct datatype for existing applications to avoid inadvertently creating data items that are incompatible with other languages. For example, if an item declared in COBOL is a PIC 9(4) DISPLAY item, PowerHouse correctly interprets it if it is defined in the data dictionary as either ZONED SIGNED or ZONED UNSIGNED. However, if a PowerHouse item contains a negative number and that item is read by a COBOL application and moved into a PIC 9(4) DISPLAY item, COBOL ignores the sign and treats the value as a positive number.

If PowerHouse writes a negative number in a ZONED SIGNED data item and COBOL does not move it but only redefines it as a PIC 9(4) DISPLAY, then the sign is not changed. However, if COBOL reads and displays the number, its last digit appears as the right brace (}) or as one of the alphabetic characters J through R.

OpenVMS: Use ZONED NUMERIC to be compatible with the numeric data type in DIBOL.

User-Defined Datatypes

User-defined datatypes are supported since they are defined using underlying system datatypes. For example, if a user-defined datatype called "LARGECHAR" is defined in terms of the SYBASE datatype "CHAR", PowerHouse treats it as CHARACTER item.

ORACLE Synonyms in PowerHouse

ORACLE synonyms are alternative user-defined names for certain database entities. These synonyms can be used in SQL DML statements supported by PowerHouse, and in certain PowerHouse statements.

The SQL DML statements supported by PowerHouse in which ORACLE synonyms can be used are:

- [SQL] CALL
- [SQL] DELETE
- [SQL] INSERT
- [SQL] UPDATE
- [SQL] DECLARE CURSOR (query-expression)
- [SQL] DECLARE CURSOR (stored-procedure)

The PowerHouse statements in which ORACLE synonyms can be used are:

- ACCESS (QUIZ and QTP only)
- CURSOR (QDESIGN)
- EDIT (QTP)
- FILE (QDESIGN)
- OUTPUT (QTP)
- TRANSACTION (PDL, QDESIGN)

To access a public synonym, you can either run the PowerHouse component with owner=public, or prefix the synonym name with "public.", as in "select * from public.name1".

Limitations to PowerHouse Statements

PowerHouse uses index information to perform many default tasks, such as creating default linkages, generating QDESIGN PATH procedures, and so forth. If the specified synonym is for an entity in a remote database—which means the synonym definition includes a database-linkname—or, if it references another user's synonym, then PowerHouse cannot retrieve index information from the base entity.

When PowerHouse does not have access to index information, standard defaulting based on indexes cannot be done. In these cases, the designer may be required to specify explicit linkage criteria, or provide ACCESS statements that can be used to generate the desired PATH procedure.

Attributes of Numeric Elements

The attributes of numeric elements govern the conversion of values from the input form (the group of characters entered by a QUICK screen user) to the internal storage form, and from the internal storage form to the display form (the group of characters that represents the value in a report or on a screen when it is recalled from storage).

Input conversion occurs in QUICK during data entry or data modification, and in QTP when run-time parameters are accepted from the user. Display conversion occurs in all PowerHouse products whenever data is reported or displayed.

The Input Conversion Process

The attributes that affect the input conversion process for numeric values are:

- item datatype
- element type
- leading sign
- trailing sign
- pattern
- allowed values
- input scale

Assume that a user entered the characters "-123.45" for an item that has these attributes:

Component	Attributes
item datatype	INTEGER SIGNED
item size	4
element type	size 8 decimal 2
leading sign	(
trailing sign	)
allowed values	-1000 to +1000
input scale	2

The process for converting from input to internal form is as follows:

1. The numeric value of the characters entered by the user is determined. (In the preceding example, the value is -123.45.)
2. This value is multiplied by ten raised to the power of the input scale. (In the preceding example, -123.45 is multiplied by 100 for a result of -12345.)
3. The value is compared with the previously scaled allowed values. (In the preceding example, -12345 is between -100000 and + 100000.) Negative values are accepted only if a nonblank leading and/or trailing sign is specified.
4. The value is "normalized" to a right-justified and blank-filled value with a leading minus sign (-) if negative, and a decimal point if required for comparison with a pattern. (The preceding example does not include a pattern.)
5. The scaled value is converted to the internal storage form (in this case, a signed four-byte binary) and stored.

Default Assumptions Governing Input

Unless otherwise specified, PowerHouse assumes that the values for numeric items are to be stored internally as integer values. Therefore, if an input scale is not declared, an assumed input scale is determined based on the number of decimal positions specified in the element type. (In the preceding example, the input scale attribute could have been omitted since the value assigned to it is equivalent to the number of decimal positions.)

If the allowed values attribute is omitted, acceptable values are defaulted according to the element type, and leading and/or trailing sign attributes. If no nonblank leading or trailing sign is specified, it is assumed that only positive values or zero are acceptable.

It is also assumed for all item datatypes other than FLOAT that the user cannot enter a value that contains more decimal positions than the number declared for that element in the dictionary, and that a value cannot exceed the number of digits allowed by the size.

If the allowed values attribute had not been included or if neither leading nor trailing sign had been specified in the preceding example, the implied acceptable values would have been in the range

```
0 to 9999.99
```

If a nonblank leading or trailing sign had been specified, the acceptable values would have been in the range

```
-9999.99 to +9999.99
```

The Output Conversion Process

The attributes that affect the output conversion process for numeric elements are:

- item datatype
- element type
- output scale
- numeric picture
- leading sign
- trailing sign
- fill character
- float character
- significance
- blank when zero

Assume that the value -123.45 is to be displayed with the following attributes:

Component	Attributes
item datatype	INTEGER SIGNED
item size	4
element type	size 8 decimal 2
output scale	0
picture	" ^^^^^^,^^ "
leading sign	(
trailing sign	)
fill character	*
float character	$

Component	Attributes
significance	5
blank when zero	yes

The process for converting the value from its internal storage form to the display form is as follows:

1. The internal value is multiplied by ten raised to the power of the output scale and rounded to the nearest integer value. (When ten is raised to the power of zero as in the preceding example, the value is not changed since $10^0=1$.)
2. The digits of the scaled integer value (without sign) are substituted into the picture from right to left until no more character positions are needed. (When the value 12345 is substituted into the picture "^^^^^^.^^ " the result is "^^^ 123.45 ".)
3. Zeros are substituted into the picture until the specified significance is attained. (In the preceding example, no zeros were substituted into the picture because the number of digits in the stored value 12345 met the significance of five as soon as the digit 3 was substituted into the picture.) If the stored value had been 5 instead of 12345, the result after significance would have been "^^^^^0.05". The number of significant positions is determined by counting all character positions in the picture (including the decimal point and the blank immediately to the left of the closing quotation mark), starting from the right.
4. The float character (if specified) is substituted. (In the preceding example, the result is "^^$123.45".) When a float character has been specified, sufficient substitution or nonsubstitution characters to display it must be declared in the picture. Using a nonsubstitution character (for example, a space) is recommended.
5. The leading and trailing signs, if specified, are substituted. If a trailing sign is specified, provision must be made for its insertion in the right-most portion of the picture. When a trailing sign has been specified, sufficient characters other than the substitution characters (^) must be declared in the picture. (In the preceding example, the result is "^($123.45)"). When a leading sign has been specified, sufficient characters (substitution or nonsubstitution) to display it must be declared in the picture. Using a nonsubstitution character (for example, a space) is recommended.
6. The fill character replaces all remaining characters in the picture. (In the preceding example, the result is "**($123.45)").

Default Assumptions for Display Attributes

If no picture is specified, a default picture is constructed based on the element type. For the element discussed previously, which has a size of eight and two decimal positions, the default picture would have been

"^^^^^^.^^"

By default no fill, float, leading sign, and trailing sign are assumed. Significance defaults to one greater than the number of decimal positions. The default output scale is zero. While these defaults are consistent with each other, specifying the picture, significance, leading sign, trailing sign, output scale, or float character may disturb the anticipated default picture and default significance. Therefore, it is recommended that you review the picture and significance whenever any of the other attributes that govern the output conversion process are specified.

Specifying Decimal Currencies

There are two common approaches to handling dollar amounts in PDL. The previous example illustrates the one that is most often used. Two decimal positions are declared for the element. At input, it is assumed an amount will be entered in dollars and cents, using a decimal point. This amount is scaled by an input scale of two and stored as cents. For display, the internal value is not scaled. Instead, a decimal point is inserted in the picture as an editing character.

Alternatively, the element could be declared with no decimal positions. In this case, it is assumed the amount will be entered in cents only; in other words, without a decimal point. The input scale would default to zero, which is suitable for the value being entered.

However, the default significance of one would not force the display of the decimal point for values less than one dollar. A significance of three or more should be specified. All other attributes governing display are as in the previous example.

Displaying Negative Values

The following table demonstrates how various combinations of numeric attributes are used in displaying negative values for the most common commercial notations.

Value	Leading sign	Trailing sign	Picture	Resulting Display	Comments
20	" "	" "	"^^^"	"20"	
-20	" "	" "	"^^^"	"###"	overflow
20	"-"	" "	"^^"	"20"	
-20	"-"	" "	"^^"	"-20"	leading sign
20	" "	"-"	"^^^"	"20"	
-20	" "	"-"	"^^^"	"20-"	trailing sign
20	" "	"CR"	"^^^"	"20"	
-20	" "	"CR"	"^^^"	"20CR"	negative credit
20	" "	"CR"	"^^^DR"	"20DR"	
-20	" "	"CR"	"^^^DR""	"20CR"	credit/debit
20	"("	")"	"^^"	"20"	
-20	"("	")"	"^^"	"(20)"	parentheses

Multiplication and Percentage Calculations

For most numeric elements used in business applications, it is preferable to store values as integers. This prevents rounding problems.

Elements such as the conversion factor, CONV-UNIT, and discount percentage, DISC-PCT, are a common exception.

Rounding and calculations are facilitated by storing actual fractional values as four-byte or eight-byte floating-point items. Scaling required to produce the integer value needed for display is performed by an output scale. The conversion factor, CONV-UNIT, is stored in the same units in which it is entered (that is, with an input scale of zero) and is scaled to an integer value for display by specifying an output scale of five.

In this case of the discount percentage element, DISC-PCT, the value is accepted as a percent (such as 12.3 for 12.3%) and an input scale of -2 is used to store the true fractional value as a real number (for example, .123). This facilitates calculations based on the element DISC-PCT. An output scale of four combined with a picture of "^^.^^%" creates the correct integer value for display. (In this case, 12.30% is displayed.)

Decimal Alignment and Scaling

To ensure proper decimal alignment for INTEGER, ZONED, and PACKED numeric items in PowerHouse, you must consider the role played by scaling. Scaling affects both the storage and display of these numeric datatypes because, by default, PowerHouse performs scaling on input and output data, and stores only whole numbers in INTEGER, ZONED, and PACKED numeric items. The exception is FLOAT datatype, discussed later in this section.

A default scaling factor is taken from the PICTURE option, defined in either the appropriate QDESIGN statement or the dictionary (if no PICTURE is defined on a QDESIGN statement). For example, a numeric item with the characteristics 9(5)V9(2) gives a scaling factor of 2, while 9(5)V9(1) gives a scaling factor of 1.

There are two types of scaling in PowerHouse, input scaling and output scaling. INPUT SCALE n means that an entered value is multiplied by the nth power of ten ("n" represents the specified number) before being stored. OUTPUT SCALE n means that a stored value is multiplied by the nth power of ten before being displayed.

The power of ten is called the "scale". An input scale of two means that the entered value is multiplied by 100 (10^2) before being stored. If an input or output scale is in the range -5 to +5, PowerHouse scales data by means of a single multiplication. If the scale is outside this range, PowerHouse scales data by means of exponentiation (that is, by a series calculation).

Scaling is required to correctly display output values containing a fraction for INTEGER, ZONED, and PACKED numeric datatypes. In PowerHouse, the display format for a numeric item is specified by the picture string. PowerHouse supplies a default picture string for an item, based on the dictionary definition.

For example, if an element is defined in the data dictionary as 9(2)V9(2), PowerHouse assigns "^^.^^" as the default picture string. On output, PowerHouse substitutes digits into the picture string from right to left, but uses only the digits from the whole number portion of the item value. Thus when a stored value of 12.34 is not scaled on output, and is formatted for output by a picture string of "^^.^^", the fractional portion is ignored, and substitution begins at the digit 2 and proceeds to the left, yielding .12 as the result. To obtain the desired display of 12.34, the value 12.34 must be multiplied (scaled) by 10^2 (100) to produce the whole number 1234. The rule is that if the picture string specifies n decimal positions, an entered value with n decimal positions must be multiplied by the nth power of ten in order to ensure the correct display.

In PowerHouse, numeric datatypes INTEGER, ZONED, and PACKED cannot represent fractional amounts. Input scaling provides the means for working around this restriction. If an element SAMPLE is defined in the dictionary as 9(2)V9(2), its corresponding item datatype is INTEGER SIZE 2, by default. An item of datatype INTEGER can represent the value 12, or the value 1234, but not the value 12.34.

Input scaling makes it possible to enter a value and retain the fractional portion. If the value 12.34 is entered with an input scale of 2, it will be multiplied by 100, yielding the whole number 1234. PowerHouse stores 1234, not 12.34. On output, the digits of the whole number are substituted into the default picture string "^^.^^". Each caret (^) in the picture string is a substitution character, and is replaced by a digit. The decimal point, however, is a nonsubstitution character, and is not replaced by a digit. Thus the displayed value is 12.34.

PowerHouse uses scaling only on the input and output of data; scaling is not used on the stored data. If PowerHouse applications perform only input and output, with no calculations, then the default values for INPUT SCALE, OUTPUT SCALE, and PICTURE ensure that the values of numeric items are displayed correctly, because one value is taken from the other. However, whenever you specify a calculation or condition that acts on the stored value in an item of datatype INTEGER, ZONED, or PACKED, you must compensate for the fact that the stored value is a scaled value (that is, no decimal points exist within the stored data).

Conditions and Scaled Values

A condition will test a stored value, which, because of input scaling, may not be necessarily the same as the entered value. For example, if element A is defined in the data dictionary as 9(3)V9(1), then the default INPUT SCALE is 1, and the stored value is ten times greater that the entered value. If you want to select values of A that are less than or equal to 10, you must compensate for input scaling on your conditional test by specifying a selection criterion of less than or equal to 100 (10^2). For example, you would use a statement like

```
>  SELECT IF A LE 100
```

to select all values for A that are less than or equal to 10.

VALUES Options and Scaled Values

Unlike the SELECT statement, the VALUES option of the FIELD statement in QDESIGN does not require you to compensate for input scaling. If you use the VALUES option to stipulate an acceptable range of values for an item to which input scaling has been applied, when you code the VALUES option you must code the exact values that define the range, rather than scaled versions. For example, you can stipulate that only values in the range 5 to 15 are acceptable for item A, as in

```
>  FIELD A VALUES 5 TO 15
```

The VALUES option must be coded this way, regardless of input scaling for the item.

Calculations and Scaled Values

If you specify calculations involving items of datatypes INTEGER, ZONED, or PACKED, you must compensate for input scaling in order to obtain correct results. For example, assume that the following elements are defined in the data dictionary:

Element Name	Display Size	Default Datatype	Decimal Positions	Default Input Scale	Entered Value	Stored Value
X	4	INT. SIZE 2	2	2	10.01	1001
Y	5	INT. SIZE 4	3	3	23.456	23456
Z	6	INT. SIZE 4	4	4	23.4567	234567

The expression X + Y sums the stored values of X and Y, and yields the value 24457, not 33.466 as expected. The expression that yields 33.466 is ((X/10^2)+(Y/10^3)).

When you specify a calculation involving datatypes INTEGER, ZONED or PACKED, the following steps will compensate for input scaling. They will also ensure proper alignment of the decimal point in the result:

1. Divide each value involved in the calculation by 10^n where n is the input scale for the item.
2. Perform the calculation.
3. Scale the result of the calculation to convert the number for storage or display.

If the result of a calculation has n decimal places, then in order not to lose the less significant digits, the result should be multiplied by the nth power of ten before it is stored or displayed.

In the previous example, the result 33.466 has three decimal places, so it should be multiplied by 10^3 (1000) before it is formatted by the picture string for display. Multiplication by 1000 can be accomplished by specifying an OUTPUT SCALE of 3 in the QDESIGN FIELD statement for the item, as in

```
>  DEFINE ANSWER1 NUMERIC*6 = ((X/100) + (Y/1000))
.
.
.
>  FIELD ANSWER1 PICTURE "^^^.^^^" OUTPUT SCALE 3
```

The advantage to using this method is that ANSWER1 contains the unscaled value of the sum. This value can be used in further calculations. The PICTURE for ANSWER1 requires an OUTPUT SCALE of 3. To accomplish the multiplication by 1000, embed it directly in the calculation, as in

```
>   DEFINE ANSWER2 NUMERIC*6 = &
>          ((X/100) + (Y/1000)) * 1000
.
.
.
>   FIELD ANSWER2 PICTURE "^^^.^^^"
```

Both ANSWER1 and ANSWER2 have an item datatype of NUMERIC*6. NUMERIC*6 is the default datatype for defined items, and is supplied by PowerHouse if no datatype is explicitly specified. NUMERIC*6 is a floating point representation that has the same storage format as FLOAT SIZE 8.

If either ANSWER1 or ANSWER2 is displayed, the result is 33.466. The floating point items are able to represent the fractional portion of the sum, so the last three digits of the result will appear correctly in the display.

In contrast, the last three digits can be lost if the result is stored in an INTEGER, rather than a FLOAT item:

```
>   DEFINE ANSWER3 INTEGER SIZE 4 = ((X/100) + (Y/1000))
.
.
.
>   FIELD ANSWER3 PICTURE "^^.^^^" OUTPUT SCALE 3
```

The internal calculation is done in floating point, but an INTEGER item cannot represent the fractional portion of the result. So, when the result is placed in an INTEGER item, truncation of the fractional portion of the number will occur, and ANSWER3 will contain 33. The OUTPUT SCALE 3 multiplies ANSWER3 by 1000, yielding 33000, and the formatted answer appears as 33.000. The fractional portion is lost.

It is possible to use datatype INTEGER and still obtain the correct answer for a calculation on stored data with a fractional portion. To accomplish this, the result of a calculation must be scaled (that is, converted to a whole number) before it is stored in the INTEGER item. Input scaling cannot be used because a calculation is being performed, rather than a value being entered. To scale the result, embed a multiplication by the approximate factor of 10 in the calculation. Then ANSWER4 will contain the whole number 33466, and the answer, formatted by the PICTURE "^^.^^^", will appear as 33.466. For example,

```
>   DEFINE ANSWER4 & INTEGER SIZE 4 =
>              (X/100) + (Y/1000)) * 1000
.
.
.
>  FIELD ANSWER4 PICTURE "^^.^^^"
```

Floating Point Calculations

To use items in calculations, it is most efficient to define them as datatype FLOAT. Floating point numbers can represent the fractional portions of numbers, so input scaling of floating point items is not necessary and can be overridden by specifying INPUT SCALE 0. When you use a floating point item with INPUT SCALE 0, the value that is stored is the same value that is entered, including any fractional portion. Floating point items must, however, be scaled for correct display. So when the INPUT SCALE is 0, you must specify OUTPUT SCALE n, where n is the number of decimal positions in the PICTURE string.

For example,

```
> TEMPORARY FLOATPOINT FLOAT SIZE 8
.
.
.
> FIELD FLOATPOINT &
>    PICTURE "^^^.^^^^" INPUT SCALE 0 OUTPUT SCALE 4
```

Note that in the previous example, the values of the input and output scales are the reverse of the default values. Where PowerHouse would, by default, have applied an input scale of 4 (based on the PICTURE "^^^.^^^^") and an output scale of 0, the reverse applies.

Examples of Calculations

With the same items as used in the table on (p. 331), you can obtain correct results by defining floating point items XF and YF (which represent the same values that were originally entered for X and Y). When you sum these, then scale the result for display, as in

```
> DEFINE XF FLOAT SIZE 8 = X / 100
> DEFINE XF FLOAT SIZE 8 = X / 1000
> DEFINE XFPLUSYF FLOAT SIZE 8 = XF + YF
.
.
.
> FIELD XFPLUSYF PICTURE "^^.^^^" OUTPUT SCALE 3
```

XF and YF will contain the unscaled values of X and Y. These values can be used in further calculations. If these values are not required for further calculations, the conversions of X and Y can be embedded in the calculation, as in

```
> DEFINE XPLUSY NUMERIC *6 = (X/100) + (Y/1000))
.
.
.
> FIELD XPLUSY PICTURE "^^.^^^" OUTPUT SCALE 3
```

The methods illustrated in the previous examples make easy work of a complex expression, because the values involved are converted to floating point before the expression is evaluated. The required OUTPUT SCALE must be determined by the number of decimal positions to be displayed, as specified in the PICTURE option.

Notes on Scaling Efficiency

In all the previous examples, multiples of ten are explicitly coded, and exponential notation is not used. This practice is recommended because exponentiation uses a series calculation, and therefore executes more slowly and is less accurate.

In complex calculations, efficiency can be improved by using algebraic factoring techniques to reduce calculations to simpler forms.

```
>   DEFINE CAL1 FLOAT SIZE 8 &
>          = ((X/100) / (Z/1000)) * (Y/1000)
>   DEFINE CAL2 FLOAT SIZE 8 &
>          = ((X/100) + (Y/1000)) / (Z/1000)
.
.
.
> FIELD CAL1 PICTURE "^^.^" OUTPUT SCALE 1
> FIELD CAL2 PICTURE "^^.^^" OUTPUT SCALE 2
```

The following example executes more quickly than the previous example.

```
> DEFINE CAL1 FLOAT SIZE 8 = (X * Y) / (Z * 100)
> DEFINE CAL2 FLOAT SIZE 8 = ((10 * X) + Y) / Z
.
.
.
> FIELD CAL1 PICTURE "^^.^" OUTPUT SCALE 1
> FIELD CAL2 PICTURE "^^.^^" OUTPUT SCALE 2
```

QUICK Screen Commands

The QUICK screen commands are listed in the tables on the following pages. For each screen command, there is a description, the syntax you use in command lists and conditional command lists, and the mnemonics you use during QUICK sessions.

Each of these commands has a default mapping to one or more function keys on each supported terminal type. You can find out what these mappings are for a terminal type by looking them up in the appropriate QKGO Context Binding Screen.

Using Screen Commands in Command Lists

In the following tables, the "QDESIGN syntax" column lists the syntax you use in command lists and conditional command lists. In this syntax, you will find the general terms, id-option and id-range.

id-option

Specifies the field on which the command operates. One of n, MARK, or PROMPT.

- n specifies an explicit field, where n is the field ID-number
- MARK specifies the field that is currently fieldmarked
- PROMPT prompts the user for an ID-number with a pop-up box

Default: If no id-option is specified, or if MARK is specified but no field is currently fieldmarked, then PROMPT is assumed.

id-range

Specifies the field or fields on which the command operates. One of n [TO m], MARK, or PROMPT.

- n [TO m] specifies an explicit range of fields, where n and m are the first and last field ID-numbers in the range
- MARK specifies the field that is currently fieldmarked.
- PROMPT prompts the user for ID-numbers with a pop-up box

Default: If no id-range is specified, or if MARK is specified but no field is currently fieldmarked, then PROMPT is assumed.

Action Commands:

Command[1]	Description	QDESIGN Syntax	Default Mnemonic	Function Key/ Alternate Function Key (OpenVMS)
Action Bar	Prompts you for Action commands by way of an Action bar and/or drop-down menus if they are available.	ACTIONBAR	BAR	GOLD_B F7
Action Field	Prompts you for Action commands by way of the Action field if it is available.	ACTIONFIELD	ACT	GOLD_A F8
Append	Prompts you to add data to the data already on the screen.	APPEND	A	KEY_PAD_8 GOLD_INSERT
Block Mode (**MPE/iX**)	Switches the terminal to Block mode, if supported.	BLOCK [MODE]	B	
Character Mode (**MPE/iX**)	Switches the terminal to Character mode.	CHARACTER [MODE]	C	
Delete	Deletes all entries on the screen or in the occurrence window.	DELETE	D	KEY_PAD_3 REMOVE
Delete Range	Deletes the fields at the specified IDs.	DELETE[id-range]	D-n/m	GOLD_REMOVE
Designer	Executes the named DESIGNER procedure for a field or fields.	DESIGNER name [id-range]		
Entry Mode	Prompts you to enter data in each field, then returns you to the Action field.	ENTRY [MODE]	E	KEY_PAD_7 INSERT
Exit (**OpenVMS**)	Terminates the current QUICK session. All changes since the last update are lost.		CONTROL/Z	F10
Extended Field Help[2]	Displays a detailed help message for a field or fields.	EXTENDED FIELD HELP [id-range]	??-n/m	GOLD_ HELP
Extended Help[2]	Displays a detailed help message for the screen.	EXTENDED HELP	??	GOLD_ PF2
Field Help[2]	Displays a one-line help message for a field or fields.	FIELD HELP [id-range]	?-n/m	HELP
Field Mark	Prompts you to enter data in each field by way of fieldmarking.	FIELDMARK	MARK	GOLD_M F9
Field Page Down	Scrolls down a page in the specified scrolling field.	PAGEDOWN [id-option]	PD-n	GOLD_J
Field Page Up	Scrolls up a page in the specified scrolling field.	PAGEUP [id-option]	PU-n	GOLD_K

Command[1]	Description	QDESIGN Syntax	Default Mnemonic	Function Key/ Alternate Function Key (OpenVMS)
Field Scroll Down	Scrolls down a line in the specified scrolling field.	SCROLLDOWN [id-option]	SD-n	GOLD_D
Field Scroll Up	Scrolls up a line in the specified scrolling field.	SCROLLUP [id-option]	SU-n	GOLD_U
Find Mode	Prompts you for a value (or values), then retrieves the specified data.	FIND [MODE]	F	KEY_PAD_4 FIND
First Record	Moves the occurrence window to the top of the cache. If there is no cache, or if the occurrence window is at the top, then no action is taken.	FIRST [RECORD]	FR	GOLD_F
Help[2]	Displays a one-line help message giving the allowed commands for the screen.	HELP	?	PF2
ID range	Moves you to each specified field in turn to enter or modify values.	ID [id-range]	n/m	
Information[2]	Displays the name of the current screen and the date it was created, as well as version and copyright information about PowerHouse.	INFORMATION	I	[Tab]
Last Record	Moves the occurrence window to the bottom of the cache. If there is no cache, or if the occurrence window is at the bottom, then no action is taken.	LAST [RECORD]	LR	GOLD_L
List[2]	Prints the data records in the occurrence window, not the complete cache contents.	LIST	L	KEY_PAD_2 F20
List All[2]	Prints the active screens in the current thread. For each screen, QUICK only prints the data records that are in the occurrence window, not the complete cache contents.	LIST ALL	L@	GOLD_ KEY_PAD_2 GOLD_F20
Logical Function Key 1 • • • Logical Function Key 8	Executes the commands defined in the association KEY statement or in QKGO.			GOLD_1 • • • GOLD_8

Command[1]	Description	QDESIGN Syntax	Default Mnemonic	Function Key/ Alternate Function Key (OpenVMS)
Modify	Allows you to Tab between fields in a panel, entering or changing data.	MODIFY	M	
Next	Retrieves next primary data record when there is a primary record structure with a repeating detail; otherwise, same as Next Data.	NEXT	N	KEY_PAD_5 GOLD_NEXT_ SCREEN
Next Data	Moves the occurrence window down one full window length.	NEXT DATA	[Return]	GOLD_N NEXT_SCREEN
Next Record	Moves the occurrences window down to the next record buffer.	NEXT RECORD	NR	GOLD_ CURSOR_ DOWN
Previous Data	Moves the occurrence window up one full window length.	PREVIOUS [DATA]	\	GOLD_KEY_ PAD_5
Previous Record	Moves the occurrence window up to the previous record buffer.	PREVIOUS RECORD	PR	GOLD_ CURSOR_ UP
Refresh[2]	Re-draws the active screen.	REFRESH	CONTROL/G	
Refresh All[2]	Re-draws the screens in the terminal buffer.	REFRESH ALL	CONTROL/W	
Refresh Line (**OpenVMS**)	Re-draws the current line.	REFRESH LINE	CONTROL/R	
Restore Keys	Resets logical function keys to map to the default terminal function keys.	RESTORE KEYS	K	
Restore Labels (**MPE/iX, UNIX**)	Resets the function key labels to those specified in the current PowerHouse application.	RESTORE LABELS	KL	
Return	Moves you up one screen. If there is no higher screen, exits QUICK.	RETURN	^	PF4 PREV_SCREEN
Return to Start	Moves you up to the startscreen (usually a menu screen). If there is no startscreen, exists QUICK.	RETURN TO START	^^^	GOLD_ ENTER GOLD_ PREV_ SCREEN
Return to Stop	Moves you up to the next-highest stopscreen. If there is no stopscreen, exits QUICK.	RETURN TO STOP	^^	GOLD_ PF4
Select Mode	First prompts you for a value (or values), then returns you to the Action field. You can enter more selection values. The specified data is then retrieved.	SELECT [MODE]	S	KEY_PAD_1 SELECT

Command[1]	Description	QDESIGN Syntax	Default Mnemonic	Function Key/ Alternate Function Key (OpenVMS)
Separator	Changes the rapid-fire separator character (by default, a semicolon) to the one you specify. Allows you to enter the default separator character as part of a field entry.	SEPARATOR	SEP char	
Shift[2]	Shifts function key levels by the number you specify, to the number you specify, or to the next level. By default, the new level remains in effect until you override it with another Shift command, make an entry, or return to the Action field. the LOCK option retains the new level until you override it with another Shift command.	SHIFT [n| {TO [LEVEL] n}] [LOCK]		
System	Takes you to the operating system prompt. To return to QUICK, enter the operating system's Exit command.	SYSTEM	**MPE/IX:** : **OpenVMS:** $ **UNIX:** !<shell-abbreviation> (e.g., !csh) **Windows:** !	DO
Toggle[2]	Cycles between screen threads.	TOGGLE [THREAD]	T	GOLD_T
Update	Updates and continues with the current mode. In Entry mode, QUICK stores the date, then prompts for more; in Find mode, QUICK retrieves the next set of data.	UPDATE	U	PF3
Update Next	If the screen does not have a primary record structure with an occurring detail, works the same as UPDATE. Otherwise, in Entry mode, if the user has not filled the cache, UPDATE prompts for more detail records. If the user has filled the cache or issued UPDATE NEXT, QUICK prompts for the next primary record. In Find mode, UPDATE retrieves the next set of detail records if they exits, otherwise it retrieves the next primary. UPDATE NEXT always retrieves the next primary.	UPDATE NEXT	UN	KEY_PAD_6 GOLD_F19
Update Return	Stores the data and returns to the parent screen. If there is no parent screen, exits QUICK.	UPDATE RETURN	UR	GOLD_ PF3

Command[1]	Description	QDESIGN Syntax	Default Mnemonic	Function Key/ Alternate Function Key (OpenVMS)
Update Stay	Stores the data and keeps the record buffer and cache intact.	UPDATE STAY	US	KEY_PAD_9 F19
User Break	Interrupts QUICK's processing and prompts at the Action field. Any internal loops, data retrieval and procedural logic are interrupted. If QUICK is run with debug enabled, calls Debugger.		**MPE/iX:** Ctrl-Y **OpenVMS, UNIX, Windows:** Ctrl-C	

[1] *For more information about how these commands map to function keys on various terminal types, see the section, "The Terminal Interface Configuration Screen" in Chapter 6, "Customizing QUICK with QKGO", in the QDESIGN Reference.*

[2] *Indicates Action and Data commands (commands that are valid in both Action and Data context). The behavior of these commands may be slightly different depending upon whether they are entered in Action or Data context. Compare the descriptions with those in the Data commands table for details.*

Data Commands:

Command[1]	Description	QDESIGN syntax	Default mnemonic	Function Key/ Alternate Function Key (OpenVMS)
Backout	Erases any entries or changes you have made, clears the cache (if there is one), and returns you to the Action field. Any entered or appended data records, or modifications to retrieved data records, are lost regardless of the current location of the occurrence window.	BACKOUT	^	PF4 PREV_SCREEN
Backup	Moves you to the previous field. QUICK scrolls the occurrence window as necessary.	BACKUP	\	KEY_PAD_ MINUS GOLD_F17
Duplicate	Duplicates the last entry you made in the field.	DUPLICATE	_	KEY_PAD_0
Exit (**OpenVMS**)	Terminates the current QUICK session. All changes since the last update are lost.		CONTROL/Z	F10
Extended Field Help[2]	Displays a detailed help message for a field (or fields).	EXTENDED FIELD HELP [id-range]	??-ID	GOLD_ HELP
Extended Help[2]	Displays a detailed help message for the field.	EXTENDED HELP	??	GOLD_ PF2 GOLD_HELP
Field Help[2]	Displays a one-line help message for a field (or fields).	FIELD HELP [id-range]	?-ID	HELP
Help[2]	Displays a one-line help message for the field.	HELP	?	PF2 HELP
Information[2]	Displays the name of the current screen and the date it was created, as well as version and copyright information about PowerHouse.	INFORMATIO N		[Tab]
List[2]	Prints the data records in the occurrence window, not the complete cache contents.	LIST		KEY_PAD_2 F20
List All[2]	Prints the active screens in the current thread. For each screen, QUICK only prints the data records that are in the occurrence window and not the complete cache contents.	LIST ALL		GOLD_ KEY_PAD_2 GOLD_F20
Next Field	Moves you to the next field. If there are no more fields or if you have filled the cache, prompts you at the Action field.	NEXT FIELD	[Return]	

Command[1]	Description	QDESIGN syntax	Default mnemonic	Function Key/ Alternate Function Key (OpenVMS)
Popup Field	Opens a pop-up Data field, if one is available.	POPUP	+	GOLD_P DO
Refresh[2]	Re-draws the active screen.	REFRESH	CONTROL/G	
Refresh All[2]	Re-draws the screens in the terminal buffer.	REFRESH ALL	CONTROL/W	
Refresh Line (**OpenVMS**)	Re-draws the current line.	REFRESH LINE	CONTROL/R	
Reverse Input Toggle	Accepts and displays data in reverse order (right to left).		|	GOLD_R
Selectbox	Opens up a pop-up selection box of valid values or captions, if one is available.	SELECTBOX	#	GOLD_S SELECT
Shift[2]	Shifts function key levels by the number you specify, to the number you specify, or to the next level. By default, the new level remains in effect until you override it with another Shift command, make an entry, or return to the Action field. The LOCK option retains the new level until you override it with another Shift command.	SHIFT [nl {TO [LEVEL] n}] [LOCK]		
Skip All	Takes you to the Action field unless there are unsatisfied required fields.	SKIP ALL	//	KEY_PAD_ ENTER F18
Skip Cluster	Takes you to the first field of the next repeating group of fields, unless there are unsatisfied required fields.	SKIP CLUSTER	/	KEY_PAD_ COMMA F17
Skip to n	Takes you to the specified field unless there are unsatisfied required fields.	SKIP TO n	/n	
Toggle[2]	Cycles between screen threads.	TOGGLE [THREAD]	T	

[1] *For more information about how these commands map to function keys on various terminal types, see the section, "The Terminal Interface Configuration Screen", in Chapter 6, "Customizing QUICK with QKGO", in the QDESIGN Reference.*

[2] *Indicates Action and Data commands (command that are valid in both Action and Data context). The behavior of these commands may be slightly different depending upon whether they are entered in Action or Data context. Compare the descriptions with those in the Data commands table for details.*

Action Bar Commands

Command	OpenVMS Function Key	OpenVMS Alternate Function Key
Accept	RETURN	
Next Option	CURSOR_DOWN	CURSOR_RIGHT
Previous Option	CURSOR_UP	CURSOR_LEFT

Field Marking Commands

Command	OpenVMS Function Key	OpenVMS Alternate Function Key
Accept	RETURN	
Next Option	CURSOR_DOWN	CURSOR_RIGHT
Previous Option	CURSOR_UP	CURSOR_LEFT

Line Edit Commands

Command	OpenVMS Function Key	OpenVMS Alternate Function Key
Clear Field **(MPE/iX, UNIX)**		
Delete Character	DEL_CHAR	
Delete Previous Character		
Delete to Start of Line	CONTROL/U	
Delete Word	CONTROL/J	F13
Input Completion	RETURN	
Insert Toggle	CONTROL/A	F14
Move Left 1 Character	CONTROL/D	CURSOR_LEFT
Move Right 1 Character	CONTROL/F	CURSOR_RIGHT
Move to End of Line	CONTROL/E	
Move to Start of Line	CONTROL/H	F12
Recall	CONTROL/B	CURSOR_UP

Menu/List/Selection Box Commands

Command	OpenVMS Function Key	OpenVMS Alternate Function Key
Accept	RETURN	
Cancel	REMOVE	PF4
Move Down	CURSOR_DOWN	
Move Up	CURSOR_UP	
Page Down	GOLD_CURSOR_DOWN	NEXT_SCREEN
Page Up	GOLD_CURSOR_UP	PREV_SCREEN

Popup Commands

Command	OpenVMS Function Key	OpenVMS Alternate Function Key
Accept	RETURN	
Cancel	REMOVE	PF4
Scroll Down	CURSOR_DOWN	
Scroll Up	CURSOR_UP	
Page Down	GOLD_CURSOR_DOWN	NEXT_SCREEN
Page Up	GOLD_CURSOR_UP	PREV_SCREEN

System Commands

Command	OpenVMS Function Key	OpenVMS Alternate Function Key
Refresh Screen	CTRL-G	
Refresh All	CTRL-W	

Text Edit Commands

Command	OpenVMS Function Key	OpenVMS Alternate Function Key
Clear Field **(MPE/iX, UNIX, Windows)**		
Delete Character	DEL_CHAR	
Delete Previous Character **(MPE/iX, UNIX, Windows)**		
Delete to Start of Line	CONTROL/U	

Command	OpenVMS Function Key	OpenVMS Alternate Function Key
Delete to End of Line	CONTROL/N	
Delete Word	CONTROL/J	F13
Input Completion	RETURN	
Insert Toggle	CONTROL/A	F14
Move Left 1 Character	CONTROL/D	CURSOR_LEFT
Move Right 1 Character	CONTROL/F	CURSOR_RIGHT
Move Up 1 Line	CURSOR_UP	
Move Down 1 Line	CURSOR_DOWN	
Move Up 1 Page	PREV_SCREEN	
Move Down 1 Page	NEXT_SCREEN	
Move to End of Line	CONTROL/E	
Move to Start of Line	CONTROL/H	F12
New Paragraph	CONTROL/P	INSERT_HERE
Recall	CONTROL/B	

Blob Support in PowerHouse

A blob (binary large object) is a data type only supported in relational databases. It is used for storing large objects of arbitrary size whose structure is not known to the database. Blobs are used to store large quantities of variable length text or objects such as images, video and sound. Blobs may require special routines and utilities to enter, display, change and update them. For more information about blobs, refer to your database documentation set.

Different databases have different datatypes for blob items. For a complete list, see "Relational PowerHouse Datatypes (Part 1)" (p. 313)

PowerHouse can reference the blob contents which can be used internally in QUICK screens, QTP runs, or QUIZ reports. You can concatenate any blob to a string, or to another blob, and assign the result to a blob. However, the display of blobs is limited by the blob contents. PowerHouse supports text blobs as fields on QUICK screens or as QUIZ report items.

For information about restrictions concerning blobs, see (p. 346).

You may use the DO BLOB verb to call external utilities such as editors to handle the contents of the blob. For more information about the DO BLOB verb, see Chapter 8 "QDESIGN Verbs and Control Structures," in the *QDESIGN Reference*.

All PowerHouse components support the following operations:

- assign contents of a blob to a string
- assign a string to a blob (creating a new blob)
- assign the contents of one blob to another blob field (in the same or a different database)
- concatenate a blob and a string (the result is a blob or string)

Using Blobs in PowerHouse Expressions

Assigning a Blob to a Character Item

When the target of an assignment is a character item (VARCHAR or CHARACTER), any blob assigned to that item is treated as a character item. If the blob size exceeds the size of the character item, it is truncated.

If the size of the source blob is larger than 32,767 bytes, then the target blob will be NULL after assignment.

In the following examples, the contents of the blob, BLOBITEM, are assigned to the character item, STRITEM.

```
LET STRITEM = BLOBITEM
DEFINE STRITEM CHARACTER*2000 = BLOBITEM
ITEM STRITEM FINAL BLOBITEM
```

In the next examples, BLOBITEM is concatenated to a string and assigned to the character item.

```
LET STRITEM  = BLOBITEM + "a string"
DEFINE STRITEM CHARACTER*2000 = BLOBITEM + "a string"
```

Using Blobs

When you use the GENERATE statement, the default options CHARACTER and FOR 1,18 are generated on any FIELD statements for blob items. These are optional.

In the following example, the item, DESCRIPTION, is a text blob of arbitrary size.

```
> FIELD DESCRIPTION OF EMPLOYEES &
>  FOR 1,50 &
>  POPUP FROM 20,1 TO 23,50 &
>  ON DATA
```

The following FIELD statement options are useful when dealing with large quantities of text:

- FOR to specify scrolling, multi-line fields
- POPUP to specify pop-up multi-line fields

Using Blobs in QUIZ

Multi-line fields, such as character, text blobs (maximum size 32,767 bytes) and varying character items, can be reported in QUIZ using the WRAP option in the report-item syntax.

Restrictions on Blobs

The restrictions on blobs are as follows:

- Only the contents of blobs smaller than 32,767 bytes can be manipulated. If the source blob is larger than 32,767 bytes, PowerHouse will give an error message.
- If the size of the source blob is larger than 32,767 bytes, then the target blob will be NULL after assignment.
- Pattern matching is supported, with the exception of QUICK in Select mode.
- Edits and other processing done in FIELDTEXT are performed only on the first 2,047 bytes of the blob.
- A blob field cannot be modified after the user does an Update Stay action. An error message

  ```
  "Record has been changed since you found it"
  ```

  will result if the user tries to modify the blob after an Update Stay. The record must be retrieved again, and then modified.
- The DUPLICATE, PICTURE, RJ, UPSHIFT, and DOWNSHIFT FIELD statement options don't apply to blob fields.

Null Value Support in PowerHouse

PowerHouse supports null values. Null values can occur in relational items, defined items, temporary items, and in predefined items such as FIELDTEXT and FIELDVALUE. PowerHouse maintains null values in expressions, conditions, and aggregate functions. Nulls can be assigned to items, and items can be tested to see if they are null.

PowerHouse works with the null value constraints of supported relational databases. If null values are not permitted at the database level, PowerHouse does not allow the use of null values.

Null values cannot be used in non-relational data structures, such as subfiles. When items containing null values are written to a non-relational data-structure, QUIZ, QTP, and QUICK use the default initialization values.

Enabling Null Value Item Initialization

By default, the use of null values is disabled. To indicate that items should be initialized to null, use the NULL VALUES ALLOWED option on the FILE or DATABASE statement in PDL.

If null values are not allowed, default initialization values are blanks for character items, and zeros for date and numeric items.

If null values are allowed in PowerHouse and the underlying database, the default item initializations are:

- blank for character items
- zeros for date and numeric items
- null for relational items regardless of datatype

Initializing Null Values in QUICK and QTP

In QUICK and QTP, relational items are initialized according to the following procedure:

1. If there is an ITEM INITIAL option for the ITEM statement, the item is initialized to this value.
2. If there is no ITEM INITIAL option, and an element with an initial value corresponds to this item, the item is initialized to the element initial value.
3. If neither of the above is true, the item is initialized to null if null is allowed for the item, or it is initialized to default values.

When initializing items of one relational record structure based on the items of another relational record, a null value is copied, provided the target item allows null values. Otherwise, the target item is initialized to default values.

When a non-relational data structure is initialized from a relational data structure and the source item has a null value, the non-relational item is initialized to default values.

Automatic Item Initialization

PowerHouse can read item values from relational record structures and write these values to items in non-relational record structures. Since null values are allowed in relational record structures, but are not allowed in non-relational record structures, PowerHouse provides initial values automatically when initializing items in non-relational data structures.

Entering and Displaying Null Values

The character used to display a null value can be redefined using the NULL VALUE CHARACTER option on the SYSTEM OPTIONS statement in PDL. QUICK and QUIZ use this character to display a null value in a field. When an item is null, the display character is left-justified in the field for a character or date item, and right-justified for a numeric item.

A null value can also be entered on a QUICK screen by using the QKGO Null Value Character.

Note: Both the Null Value Character and the QKGO key can be configured in QKGO.

Assigning Null Values

NULL can be defined as a term in any PowerHouse expression to assign a null value to a defined, temporary or relational item.

For example, NULL may be used in statements such as DEFINE, LET, and ITEM:

```
> DEFINE PROJECT-NAME CHARACTER*20 = &
>  CASE OF EMPLOYEES &
>     WHEN 1001 THEN "PRODUCTION" &
>     WHEN 1002 THEN "PROMOTIONS" &
>     DEFAULT NULL
> ITEM RELITEM INITIAL NULL
> LET RELITEM = NULL
```

Testing for Null Values

You can test for null values in PowerHouse using the pre-defined condition:

item [IS] NULL|MISSING

This condition can be used wherever a condition can be specified in PowerHouse. The MISSING keyword can be used in place of the NULL keyword in the PowerHouse predefined condition, but not in the SQL predefined condition. For more information about conditions, see (p. 289).

The condition IS NULL or IS MISSING must be used to determine whether a value is null or not null. When a value is compared to NULL using other conditions, the result is always NULL.

If any operand in an expression has a null value, the result is set to NULL. Whenever a conditional expression evaluates to NULL then PowerHouse treats it as "not true".

The following condition is used for testing null values in SQL

columnspec IS [NOT] NULL

The following example reports the number of employees in the record structure EMPLOYEES that have dependents and the number that do not:

```
> ACCESS EMPLOYEES
> DEFINE EXISTINGCOUNT = 1 &
>   IF DEPENDENTS EXISTS AND DEPENDENTS > 0 &
>   ELSE 0
> DEFINE NULLCOUNT = 1 &
>   IF DEPENDENTS IS NULL OR DEPENDENTS = 0 ELSE 0
> REPORT LASTNAME FIRSTNAME DEPENDENTS
> FINAL FOOTING SKIP 3 &
>   "Number of employees with dependents: " &
>   EXISTINGCOUNT SUBTOTAL &
>   "Number of employees without dependents:" &
>   NULLCOUNT SUBTOTAL
```

With the SELECT statement, you can specify record selection based on whether or not a null value occurs in an item by using the IS NULL predefined condition. For example,

```
> FILE EMPLOYEES IN REGIONAL
> SELECT IF DEPENDENTS IS NULL
```

or, in QTP or QUIZ, enter:

```
> ACCESS EMPLOYEES IN REGIONAL
> SELECT IF DEPENDENTS IS NULL
```

Operating on Null Values in PowerHouse

If NULL VALUES ALLOWED is specified in the dictionary, you can operate on null values. With the exception of summary-operations, if a term in a string, date, or numeric expression has a null value, the value of the expression is set to null. Similarly, if a null value is used as a parameter to a function, the result is null.

For summary-operations, null values are ignored. If all the values used in a summary-operation are null, the result is null. As an example, when a null value is encountered during an averaging calculation it is not included in the subtotal or count. The averaging is based only on the non-null values. The following table lists the summary-operations for each PowerHouse component.

PowerHouse Component	Summary-operations and Null Values
QDESIGN	SUM option on the ITEM and TEMPORARY statements
QTP	SUM function AVERAGE, MAXIMUM, MINIMUM, and SUBTOTAL summary-operation
QUIZ	AVERAGE, MAXIMUM, MINIMUM, PERCENT, RATIO, and SUBTOTAL summary-operation

Selective Record Retrieval Based on Null Values

With the SELECT statement, you can specify record selection based on whether or not a null value occurs in an item. For example, suppose that you want to retrieve only those records from a database where the DEPENDENTS item is null. In QDESIGN, enter

```
>   FILE EMPLOYEES IN REGIONAL
>   SELECT IF DEPENDENTS IS NULL
```

or in QTP, enter

```
>   ACCESS EMPLOYEES IN REGIONAL
>   SELECT IF DEPENDENTS IS NULL
```

NULL cannot be used as a value for a linkitem on statements such as the CHOOSE statement or the ACCESS statement since PowerHouse cannot pass NULL as retrieval criteria in a request to a database.

Controlling Null Value Entry in QDESIGN

The following table shows the null value specifications which affect how PowerHouse applies null value support in QDESIGN with relational tables.

If a view is specified in the FILE or CURSOR statement, QDESIGN cannot determine if any columns are NOT NULL. In these cases, QDESIGN does not generate the NULL VALUE NOT ALLOWED option for the FIELD statement unless the dictionary says NULL VALUES NOT ALLOWED

The output of the QSHOW SHOW RECORD statement will display the letter "r" to the left of each column where values are required.

Database column specification	PDL FILE or DATABASE option	Default QDESIGN FIELD option	QSHOW SHOW RECORD display
nulls allowed	NULL VALUES NOT ALLOWED	NULL VALUE NOT ALLOWED	"r"
nulls not allowed	NULL VALUES NOT ALLOWED	NULL VALUE NOT ALLOWED	"r"
nulls allowed	NULL VALUES ALLOWED	NULL VALUE ALLOWED (not generated)	
nulls not allowed	NULL VALUES ALLOWED	NULL VALUE NOT ALLOWED	"r"

A null value can be explicitly entered on a QUICK screen by using the QKGO Null Value Character. The Null Value Character can be configured on the Data Field Commands Screen in QKGO.

For a column that allows null values, the FIELD option, NULL VALUE NOT ALLOWED, prevents you from explicitly entering the null value character in the field. However, QUICK will store a null value if a null response (such as a carriage return) is entered. To prevent QUICK from supplying unintended null values, specify REQUIRED with NULL VALUE NOT ALLOWED.

QDESIGN automatically generates the REQUIRED option only for fields that correspond to indexes.

Pattern Matching in PowerHouse

A pattern is a special string of characters that you can use to validate values during data entry or to specify selective retrieval of data records.

Patterns have broad applications throughout PowerHouse. Typically, you define patterns for individual elements in the data dictionary. The other PowerHouse components use the pattern definitions specified in your data dictionary. You can specify patterns for matching purposes in QTP, QUICK, PDL, and QUIZ.

Types of Characters Used in Pattern Matching

Patterns are made up of two types of characters:

- exact-match characters
- metacharacters

Using Exact-Match Characters

The simplest type of pattern matching involves exact-match characters. Exact-match characters match only themselves. There is a literal match between each exact-match character and its corresponding digit or character of the item value. For example, if the pattern is "a2b", then only the value "a2b" matches the pattern.

All alphabetic and numeric characters used in pattern matching are exact-match characters:

A-Z	uppercase characters match uppercase characters
a-z	lowercase characters match lowercase characters
0-9	a digit matches itself
a blank (or a space)	matches itself

Nonprinting ASCII characters are not allowed in patterns.

The following non-alphabetic and non-numeric special characters match themselves:

+ { } - ` ’ ~ $. , % / "_

Note: The underscore matches itself if it is not in the Reserved Metacharacter string. If it is included in the Reserved Metacharacter string, it must be preceded in pattern matching by the escape character (!).

Using Metacharacters

You can construct useful patterns using metacharacters. A metacharacter is a character that describes a class of characters or something about the pattern rather than simply matching itself.

Default Metacharacter	PowerHouse Keyword	Function
^	ALPHA	Matches any single uppercase or lowercase alphabetic character. The characters in the upshift downshift table are also used as valid alphabetic characters. For example, the pattern "^^##" matches values such as "aa22" and "ab23". The metacharacter ^ matches uppercase or lowercase alphabetic characters, so the pattern would match both "ab23" and AB23".

Default Metacharacter	PowerHouse Keyword	Function
?	ANY	Matches any single character (alphabetic, numeric, or special). For example, in the pattern "a?b", the question mark is a metacharacter; it matches any single special, alphabetic, or numeric character. The pattern "a?b" matches the values "aAb", "aab", "a2b", and so on.
#	DIGIT	Matches any single numeric digit.
!	ESCAPE	Signifies that the character immediately following the escape character is interpreted as a regular character rather than as a metacharacter. For example, the pattern "^^^^!?" requires a question mark as the fifth character. You can't use this metacharacter with alphabetic or numeric characters (except !0). It doesn't affect non-alphabetic or non-numeric characters that aren't metacharacters.
( )	LEFTP, RIGHTP	Indicates the precedence in which the characters in the pattern are interpreted.
\	NOT	Disallows the character that immediately follows. For example, the pattern "###\0" accepts any three-digit number followed by any character other than zero.
!0	NULL	Matches a null entry. Precede 0 with the current escape metacharacter, which is, by default, an exclamation mark (!).
<	OPTIONAL	Matches zero or any occurrence of the immediately preceding character (or pattern string in parentheses). This is the "optional" character. For example, the pattern "a<" accepts the null value or an "a".
*	OPTREP	Matches zero or more occurrences of the immediately preceding character (or pattern string in parentheses). This is the "optional repeating" character. For example, the pattern "a*" accepts the null value or any number of "a"s.
|	OR	Matches the character (or characters enclosed in parentheses) on either side of this metacharacter. Use this metacharacter to specify alternatives. PowerHouse accepts a value that matches any one alternative.
>	REPEAT	Matches one or more occurrences of the immediately preceding character (or pattern string in parentheses). This is the "repeating" character. For example, the pattern "a>" doesn't accept the null value but does accept any number of "a"s.
@	WILD	Matches zero or more characters (alphabetic, numeric, or special). For example, the pattern "Th@" matches any value beginning with "Th". The pattern "@th" matches any value ending with "th". The pattern "M@th" matches any value beginning with "M" and ending with "th" (including the value "Mth").

Reserved Metacharacters

The following metacharacters aren't defined, and are reserved for future expansion of the PowerHouse pattern matching feature:

[ ] : = ; &

or

[ ] : = ; _ &

Note: The underscore can be removed from the Reserved Metacharacters strong or left in for backward compatibility.

Matching Characters that are Metacharacters

To match characters that are metacharacters in patterns, you must precede them with the escape metacharacter (!).

Changing the Metacharacters Used for Pattern Matching

Typically, the metacharacters used for pattern matching in PowerHouse applications are established in the dictionary. It is recommended that you do not change the dictionary default metacharacters listed in the table on (p. 351) except when they conflict with the character set of a language other than English.

The facility to change the reserved metacharacters is provided to support alternative-language character sets, although it is not recommended. The reserved metacharacters are associated in PDL with the RESERVED CHARACTERS option of the SYSTEM OPTIONS statement. You must match replacement characters one-for-one with the preceding reserved metacharacters. The first character specified replaces the left brace ([), the second replaces the right brace (]), and so on.

Precedence of Metacharacters in Pattern Matching

The normal sequence of processing is

1. !
2. ( )
3. \ * > <
4. concatenation
5. |

Use parentheses ( ) as metacharacters to override the normal order of interpretation of characters within the pattern. For example, the pattern "ablc" accepts the values "ab" or "c". The pattern "a(blc)" accepts the values "ab" or "ac".

Types of Patterns

The three types of patterns used in pattern matching are character, numeric, and date.

Character Patterns

Character patterns match whatever characters you specify.

Numeric Patterns

PowerHouse evaluates and reformats the value of a numeric entry by

1. Inserting a decimal character in the number in the correct position if you indicate that one is required in the PICTURE and INPUT SCALE options.
2. Suppressing leading zeros in the number up to the decimal character (if any). The one exception to this rule is an item that contains all zeros and does not require a decimal character. In this case, one zero is left: "00000" defaults to "0".
3. Providing a floating minus sign (-) if the number is negative.
4. Right-justifying and padding the number with blanks on the left.

If you specify an item names PRICE as numeric SIZE 5 DECIMAL 2 with an INPUT SCALE of two, PowerHouse converts the values entered by a user to the following standard forms for numeric items:

Entered value	Standard form
005.23	5.23
+5.23	5.23
-05.23	-5.23
-5.23	-5.23
0.01	.01
000.00	.00

You should design numeric patterns that match the standard form described in the previous table. It is not necessary to match the leading spaces since PowerHouse applications automatically extend the pattern to match the leading spaces by adding " *" to the start of the pattern.

For example, the pattern

"-<#*.##"

matches all the previous examples since PowerHouse programs extend the pattern to

" *-<#*.##"

Date Patterns

The standardized form of date items is the full date (year, month, day), converted to the date format and date separator that you specify for the item. If item PURCHASE-DATE has an MMMDDYY format and the SEPARATOR is a slash (/), the correct pattern to match all the dates for February is "FEB/##/##". Since normal date editing still applies, PowerHouse accepts the use of "##" for the DD and YY portions of the date.

Formal Pattern Matching Syntax

In Backus Naur form, the precise syntax of PowerHouse patterns is

<pattern>	::=<pattern>|<term>
	::=<term>
<term>	::=<term><factor>
	::=<factor>
<factor>	::=\<char>
	::=<token>*
	::=<token><
	::=<token>>
	::=<token>
<token>	::=(<pattern>)
	::=<char>
<char>	::=!<escape char>
	::=character other than \ | ( ) ! > < *

<escape char>	::=0 or any special character

Characters with an ASCII value of 31 or less (nonprinting ASCII characters such as carriage return and line feed) are not allowed in patterns.

Example Patterns

The following examples demonstrate how to construct useful patterns:

- A ten-digit telephone number of the form (506) 555-1212 that accepts left-justified entries with or without an area code:

  (!(###!)) <###-####

  Note that a blank is used here to match itself.
- The pattern "(\0)*" doesn't allow entries of zeros in any position.

  Note that the string "\0*" is not a valid pattern since both metacharacters operate on the same object.
- A Canadian postal code or a 5- or 9- digit U.S. zip code:

  (^#^ #^#)|(#####(-####)<)

Pattern Matching in SQL

columnspec [NOT] LIKE sql-pattern [ESCAPE character]

The LIKE condition is used for pattern-matching. The column specification must identify a column of type character. If no ESCAPE option is used, characters within the pattern are interpreted as follows:

- The underscore (_) matches any single character (alphabetic, numeric, or special).
- The percent sign (%) matches zero or more characters (alphabetic, numeric, or special).
- All other characters match themselves.

The ESCAPE option indicates that the character immediately following the ESCAPE character in a pattern is interpreted as a regular character rather than a metacharacter. The ESCAPE character can be any character not used explicitly in your pattern.

Note that the metacharacters and the escape character used in sql-pattern matching are not the same as those used in PowerHouse pattern matching.

Using the SOUNDEX Option

The SOUNDEX option systematically abbreviates words and names according to the rules of English phonetics. You use SOUNDEX with either the GENERATE or SHOW statement in QSHOW.

The SOUNDEX option takes the following form:

SOUNDEX(string-expression [,numeric-expression])

string-expression

Specifies the word to be coded.

numeric-expression

Controls the number of characters in the resulting code string. This parameter is optional.

Default: 4 characters

The optional numeric expression entered after the string expression determines the size of the key used for the search. The size of the number controls the number of names that the string matches: the larger the number, the fewer names are retrieved. The default of 4 is adequate for most searches, but increasing the number can be useful in some circumstances. For example, if most of the file names in your data dictionary begin with STOCK, searching for sound-alike files with a key of size 5, 6, or 7 will probably produce better results.

SOUNDEX Option Rules

The SOUNDEX option uses the following rules to generate soundex codes:

1. If adjacent letters are identical, then only the first occurrence of the letter is kept.
2. The first character is always retained.
3. The vowels A, E, I, O, U, Y and consonants W and H are dropped (except when they are the first character in the string).
4. For each of the remaining letters, except the first, a numeric value is assigned, as follows:

Number assigned	For these letters
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R

5. If adjacent assigned numeric values are equal, then only the first occurrence is kept.
6. If there are insufficient letters to produce a result with the number of characters that are specified by the size parameter (numeric-expression), then the remainder is filled with zeros.

The preceding rules work for most words or names that conform to English spelling conventions. However, the SOUNDEX function may not produce satisfactory results for data that contains many non-English words or names.

Chapter 6: Functions in PowerHouse

Overview

This chapter describes PowerHouse functions in detail. For each function, you'll find

- detailed syntax descriptions
- detailed function discussions
- examples where applicable

About Functions in PowerHouse

Four kinds of functions are used in PowerHouse:

- data manipulation functions (string, numeric, date)
- logical functions
- system functions
- SQL data manipulation functions

Any number of functions can be combined or nested as long as the results are compatible, as in

```
> DEFINE FUNC = &
>      MOD(ABSOLUTE(CEILING(TRANS_AMT)), &
>      INDEX(VENDOR_CODE,"HIJ"))
```

String, numeric, and date data manipulation functions are used in the following form:

function(**expression**)

The result is called a function result. A function result can also be used within an expression as one of the terms of the expression.

Functions with system-generated input are used in the following form:

function

System functions are used to monitor aspects of the operating system and program processing. System functions can be tested at any time and you can use them as terms in an expression.

Note: If an expression evaluates to null, then the result of a function applied to that expression is also set to null.

In the descriptions that follow, a short example is included. Underlining indicates relative positioning; it isn't part of the input or the result.

Summary of PowerHouse Functions

The following table summarizes the PowerHouse functions:

Function	Description	Type	Input	Result	Qdesign	Quiz	Qtp
ABSOLUTE	Returns the absolute value of a number.	DMF	N	N	✔	✔	✔
ADDCENTURY	Converts a 6-digit date to 8 digits.	DMF	D,N	D	✔	✔	✔

Function	Description	Type	Input	Result	Qdesign	Quiz	Qtp
ASCII	Converts a number to a character string.	DMF	N	S	✔	✔	✔
ATTRIBUTE	Returns the specified system option being used in the current session.	SF	S	S	✔		
AUDITSTATUS	Returns a single character indicating the current audit trail status of a record buffer.	SF	*	S	✔		
Bit Extract	Extracts bits from a number.	DMF	N	N	✔	✔	✔
BITEXTRACT	Extracts bits from a value.	SQL-DMF	N	N	✔	✔	✔
CEILING	Rounds an integer up.	DMF	N	N	✔	✔	✔
CENTER|CENTRE	Centers characters in a string.	DMF	S	S	✔	✔	✔
CENTURY	Extracts the century from a date item or expression.	DMF	D	N	✔	✔	✔
CHARACTERS	Specifies an item that is addressed as a character string.	SF	N	S	✔	✔	✔
CHARACTER_ LENGTH CHAR_LENGTH	Returns the size of a string expression in characters.	SQL-DMF	S	N	✔	✔	✔
CHECKSUM	Returns a checksum for a string.	DMF	S	N	✔	✔	✔
COMMANDCODE	Returns the status code issued when an operating system command is executed.	SF	*	S	✔		
COMMANDMESSAGE	Returns the text of any warning or error message issued when executing an operating system command.	SF	*	S	✔		
COMMANDSEVERITY (**OpenVMS**)	Returns the severity level of the most recently executed operating system command.	SF	*	S	✔		

Function	Description	Type	Input	Result	Qdesign	Quiz	Qtp
COMMANDSTATUS (**OpenVMS**)	Returns the numeric value stored in $STATUS.	SF	*	N	✔		
CONTENTS	Returns the contents of a blob as a character string.	SF	S	S	✔	✔	✔
DATE	Calculates the date that is a specified number of days from the base date.	DMF	N	D	✔	✔	✔
DATEEXTRACT	Extracts part of a date item, such as the month or hour.	DMF	D	N	✔	✔	✔
DAYS	Returns a quantity of days from a date.	DMF	D	N	✔	✔	✔
DECIMALTIME	Returns the fractional quantity of time between two dates.	DMF	D	N	✔	✔	✔
DECRYPT	Decodes an encrypted key.	DMF	S	S	✔	✔	✔
DELETESYSTEMVAL	Deletes values defined at the operating system level.	LF	S	B	✔	✔	✔
DOWNSHIFT	Shifts characters to lowercase.	DMF	S	S	✔	✔	✔
ENCRYPT	Creates an encryption key.	DMF	S	S	✔	✔	✔
EXTRACT	Extracts the requested part of a datetime or interval value expression.	SQL-DMF	D	N	✔	✔	✔
FIRST	Returns the value of the item in the first occurrence of the associated file.	SF	*	D, N, S	✔		
FLOOR	Rounds a number down to an integer.	DMF	N	N	✔	✔	✔
FORMATNUMBER	Formats a number according to a format string.	DMF	N,S	S	✔	✔	✔
GETSYSTEMVAL	Retrieves values defined at the operating system level.	SF	S	S	✔	✔	✔

Function	Description	Type	Input	Result	Qdesign	Quiz	Qtp
HEXDECODE	Converts a hexadecimal string to an ASCII string.	DMF	S	S	✔		
HEXENCODE	Converts an ASCII string to a hexadecimal string.	DMF	S	S	✔		
INDEX	Finds the starting position of one substring within another.	DMF	S	N	✔	✔	✔
INTERVAL	Returns the days/time value of a string or number as a fractional value	DMF	D	N	✔	✔	✔
JCW (**MPE/iX**)	Returns an integer indicating the value of the specified JCW.	SF	S	N	✔	✔	✔
LASTDAY	Sets a date to the last date of the month.	DMF	D	D	✔	✔	✔
LEFT JUSTIFY|LJ	Left-justifies characters in a string.	DMF	S	S	✔	✔	✔
LINKVALUE	Returns the highest, lowest or equal value of a linkitem in the VIA option of an ACCESS statement in QDESIGN or the CHOOSE statement in QUIZ and QTP.	SF	*	D,N,S	✔	✔	✔
LOGONID	Returns a username and an account name (**MPE/iX**), or a user-id (**OpenVMS**), or a user logonid (**UNIX, Windows**).	SF	*	S	✔	✔	✔
LOWER	Downshifts a string expression.	SQL-DMF	S	S	✔	✔	✔
MATCHPATTERN	Compares a string to a pattern.	LF	S	B	✔	✔	✔
MATCHUSER	Determines whether a user belongs to a given application security class.	LF	S	B	✔	✔	✔
MISSING	Returns NULL.	SF	*	NULL	✔	✔	✔
MOD	Returns a remainder after division.	DMF	N	N	✔	✔	✔

Function	Description	Type	Input	Result	Qdesign	Quiz	Qtp
NCONVERT	Converts a character string to a number.	DMF	S	N	✔	✔	✔
NULL	Returns NULL.	SF	*	NULL	✔	✔	✔
OCCURRENCE	Returns the occurrence number for the currently active FOR construct.	SF	*	N	✔		
OCTET_LENGTH	Returns the size of a string expression in bytes.	SQL-DMF	S	N	✔	✔	✔
OLDVALUE	Returns the existing value of the item during editing.	SF	*	D, N, S	✔		
PACK	Packs characters in a string.	DMF	S	S	✔	✔	✔
PORTID	Identifies the terminal device.	SF	*	S	✔	✔	✔
POSITION	Gives the starting position of the first string in the second string.	SQL-DMF	S	N	✔	✔	✔
PROCESSLOCATION	Returns the value of the **procloc** program parameter.	SF	*	S	✔		
RANDOM	Returns a random number.	SF	N	N	✔		✔
RECORDLOCATION	Returns the physical record number of the current occurrence of a data record.	SF	*	N	✔		
REMOVECENTURY	Converts an 8-digit date to 6 digits.	DMF	D	D	✔	✔	✔
REVERSE	Reverses the characters in a string so that characters entered left to right appear right to left. Blanks appear to the left.	DMF	S	S	✔	✔	✔
RIGHT JUSTIFY|RJ	Right-justifies characters in a string.	DMF	S	S	✔	✔	✔
ROUND	Returns a rounded number.	DMF	N	N	✔	✔	✔

Function	Description	Type	Input	Result	Qdesign	Quiz	Qtp
SCREENLEVEL	Returns the level number of the current screen.	SF	*	N	✔		
SETSYSTEMVAL	Assigns values at the operating system level.	LF	S	B	✔	✔	✔
SHIFTLEVEL	Returns the current shift level of the function keys.	SF	*	N	✔		
SIGNONACCOUNT (**MPE/iX**)	Returns a string of up to eight characters that contains the user's logon account.	SF	*	S	✔	✔	✔
SIGNONGROUP (**MPE/iX**)	Returns a string of up to eight characters that contains the user's logon group.	SF	*	S	✔	✔	✔
SIGNONUSER	Returns a string of up to eight characters that contains the user's logon user id.	SF	*	S	✔	✔	✔
SIZE	Returns the size of a string.	DMF	S	N	✔	✔	✔
SOUNDEX	Creates a phonetic code from a string.	DMF	S,N	S	✔	✔	✔
SPREAD	Adds an additional space between each character of a string expression.	DMF	S	S	✔	✔	✔
SQLCODE	Returns the status code of the last SQL statement executed.	SF	*	N	✔		
SQLMESSAGE	Returns the error message that explains the status code of the last SQL statement executed.	SF	*	S	✔		
SUBSTITUTE	Retrieves messages from a designer message file. Replaces any substitution characters in a string or designer message.	DMF	S, N	S	✔	✔	✔
SUBSTRING	Extracts a portion of a string expression.	SQL-DMF	S,N	S	✔	✔	✔

Function	Description	Type	Input	Result	Qdesign	Quiz	Qtp
Substring Extract	Extracts a substring from a string.	DMF	S,N	S	✔	✔	✔
SUM	Sums items in an array.	DMF	N	N		✔	✔
SYSDATE	Returns the current system date.	SF	*	D	✔	✔	✔
SYSDATETIME	Returns the current system date and time.	SF	*	D	✔	✔	✔
SYSNAME	Returns the dictionary title specified in the data dictionary as a 40-character string.	SF	*	S	✔	✔	✔
SYSPAGE	Returns the QUIZ report page number.	SF	*	N		✔	
SYSTIME	Returns the current system time.	SF	*	N	✔	✔	✔
TERMTYPE	Returns a string expression containing full specifications for the terminal type.	SF	*	S	✔		
TRUNCATE	Removes trailing blanks from a string.	DMF	S	S	✔	✔	✔
UIC (**OpenVMS, UNIX**)	Returns user's [gid,uic].	SF	*	S	✔	✔	✔
UPPER	Upshifts a string expression.	SQL-DMF	S	S	✔	✔	✔
UPSHIFT	Shifts characters to uppercase.	DMF	S	S	✔	✔	✔
VALIDPATTERN	Checks a pattern string.	LF	S	B	✔	✔	✔
VMSTIMESTAMP (**OpenVMS**)	Returns the current system date and time.	SF	*	D	✔	✔	✔
WEBLOGONID	Returns the authenticated username.	SF	*	S	✔		
ZEROFILL	Replaces leading spaces with zeros.	DMF	S	S	✔	✔	✔

Function	Description	Type	Input	Result	Qdesign	Quiz	Qtp
S - string	*N - numeric*	*D - date*	*B - boolean*	** - system-generated*			
Types							
DMF	*data manipulation function*						
SQL-DMF	*SQL data manipulation function*						
LF	*logical function*						
SF	*system function*						

ABSOLUTE

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Numeric	Numeric	✔	✔	✔

Returns the absolute value of a number.

Syntax

ABSOLUTE(numeric-expression)

numeric-expression

A single number or a series of terms that yields a numeric result. For information about numeric results, see (p. 303).

Examples

```
Input: ABSOLUTE(-465)
Result: 465

Input: DEFINE ITEMA NUM*5 = 10
       DEFINE ITEMB NUM*5 = 24
       DEFINE ABSNUM NUM*5 = ABSOLUTE(ITEMA - ITEMB)
Result: ABSNUM = 14
```

ADDCENTURY

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Date, Numeric	Date	✔	✔	✔

Converts a 6-digit date to an 8-digit date.

Syntax

ADDCENTURY(date-expression[,numeric-expression])

date-expression

Specifies the input date.

numeric-expression

Overrides the default century that is established in the dictionary. This parameter is optional.

Examples

```
Input: ADDCENTURY(900525)
Result: 1990/05/25

Input: ADDCENTURY(010704,20)
Result: 2001/07/04
```

When you mix 6-digit and 8-digit dates in an expression, or compare dissimilar date types in a condition, use either the REMOVECENTURY function or the ADDCENTURY function. For example, if the system options are set to century excluded, SYSDATE is a 6-digit date. You can use ADDCENTURY to define an 8-digit version of this system date, as in

```
> DEFINE LONGSYSDATE DATE CENTURY INCLUDED &
>    FORMAT YYYYMMDD = ADDCENTURY(SYSDATE)
```

or to make comparisons, as in

```
>   SELECT IF LATEDATE = ADDCENTURY(SYSDATE)
```

Use the ADDCENTURY function to derive the value of an 8-digit date from a 6-digit date. For example:

```
> DEFINE SHORTDATE DATE CENTURY EXCLUDED = 851128
> DEFINE LONGDATE DATE CENTURY INCLUDED = &
>   ADDCENTURY(SHORTDATE)
```

ASCII

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Numeric	String	✔	✔	✔

Converts a number to a character string.

Syntax

ASCII(numeric-expression1[,numeric-expression2])

numeric-expression1

Specifies the number to be converted.

numeric-expression2

Specifies the size of the output string. This is an optional parameter. If it is present, the output string is right-justified and zero-filled to the size specified.

Discussion

The ASCII function ignores digits to the right of the decimal point.

Examples

```
Input: ASCII(236)

Result: 236
Input: ASCII(236,5)
Result: 00236
```

The following example extracts the month portion of today's date assuming CENTURY INCLUDED:

```
> DEFINE NUMTOCHAR CHAR*8 = ASCII(SYSDATE,8)
> DEFINE MONTHPORTION CHAR*2 = NUMTOCHAR[5:2]
```

ATTRIBUTE

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	String	String	✔		

Returns the specified character being used within the current Application.

Syntax

ATTRIBUTE(class, option)

class

CHARACTER

option

One of the following:

Option	Returns
MESSAGE	the current message substitution character
MULTILINE	the current multi-line column heading character
PICTURE	the current picture substitution character
RETRIEVAL	the current generic retrieval character

Discussion

The ATTRIBUTE function returns the specified dictionary system option character (generic retrieval, message substitution, multiline column heading, or picture substitution) being used within the current session.

Examples

```
Input: ATTRIBUTE (CHARACTER, RETRIEVAL)
Result: @
```

To verify which character is currently being used as the message substitution character, you could establish a defined item, as in

```
DEFINE SUBCHAR CHAR*1 = ATTRIBUTE (CHARACTER, MESSAGE)
```

AUDITSTATUS

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔		

Returns a single character indicating the current audit trail status of a record buffer.

Syntax

AUDITSTATUS [OF record-structure]

record-structure

The name of a record-structure defined in the dictionary or table in a relational database.

Discussion

The character is one of

- C (for old changed record)
- D (for old deleted record)
- N (for new record)

If the OF file qualifier is omitted, the status is that of the assumed file.

The purpose of the AUDITSTATUS function is to document the data record status in an AUDIT file. If you want to test the data record status during screen processing for any other reason, you should use the predefined conditions, NEWRECORD, DELETEDRECORD, and ALTEREDRECORD. The predefined conditions are used because AUDITSTATUS returns a value of C for all old and undeleted data records whether they are changed or not.

AUDITSTATUS cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

Bit Extract

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Numeric	Numeric	✔	✔	✔

Extracts bits from a number.

Syntax

numeric-expression[start:length]

The square brackets are required syntax in this function.

numeric-expression

Specifies the input number. Before bit extraction takes place, the numeric-expression is converted to a 2-byte unsigned integer. Bit numbering starts at zero, which is the most significant bit of the 2-byte integer.

start

Specifies the starting position of the extract. The most significant bit is 0; the least is 15.

length

Specifies the length of the extract.

Discussion

The Bit Extract function cannot be performed on any values that cannot be converted into a 2-byte integer. For example, PHDATE datatypes or 4 or 8-byte integers containing values too large for a 2-byte integer.

Examples

```
Input: DEFINE ITEMX INTEGER UNSIGNED SIZE 2 = 3[13:3]
Result: 3
```

bit extract	numeric-exp	bit pattern	result
255[7:4]	255	0000 0000 1111 1111	7
1[0:1]	1	0000 0000 0000 0001	0
1[15:1]	1	0000 0000 0000 0001	1
3[13:3]	3	0000 0000 0000 0011	3

BITEXTRACT

Type	Input	Result	QDESIGN	QUIZ	QTP
SQL-DMF	Numeric	Numeric	✔	✔	✔

Extracts bits from a numeric value.

Syntax

BITEXTRACT(numeric-expression,start,number)
Limit: Valid only in SQL.

numeric-expression

Specifies the input value.

start

Specifies the starting position (bit number) for the extract. Bit numbering starts at zero.

number

Specifies the number of bits to extract.

CEILING

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Numeric	Numeric	✔	✔	✔

Rounds an integer up.

Syntax

CEILING(numeric-expression)

numeric-expression

Specifies the input number.

Discussion

This function is commonly used to round up monetary values to the nearest dollar. Whole numbers are not increased.

Examples

```
Input: CEILING(79.04)
Result: 80

Input: CEILING(79)
Result: 79

Input: CEILING(-79.04)
Result: -79

Input: CEILING(-79)
Result: -79

Input: > DEFINE ITEMX NUM*8 = CEILING(45.6 - .5)
Result: 46
```

CENTER|CENTRE

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔	✔	✔

Centers characters in a string.

Syntax

CENTER(string-expression)

CENTRE(string-expression)

string-expression

Specifies the string to be centered.

Discussion

The CENTER function centers nonblank text within the length of a string expression.

Example

```
Input: CENTER("word    ")
Result:  ..word..
```

The length of a character item on the left of the equal sign (=) in a DEFINE statement has no effect on the evaluation of the string expression on the right of the equal sign. The result of the string expression is moved into the defined item from left to right. If the length of the expression result is longer than the defined item, the extra characters are truncated. If the length is shorter than the defined item, the defined item is padded with blanks. For example, the string-expression, called name, a 20 character item, is centered in a 30 character item, c_name.

```
> DEFINE c_name CHARACTER*30 = CENTER (name)
```

The result is the name centered in 20 characters and then moved into c_name from left to right with 10 blanks added to the right. So, using "Williams" would result in 6 blanks, the name "Williams" and 16 blanks.

To correctly center the name in the defined item, use the SUBSTRING function to expand the item length to be the same as the defined item.

```
> DEFINE c_name CHARACTER*30 = CENTER (name[1:30])
```

Ensure that you use the SUBSTRING function on the item and not on the function result. In this example, the SUBSTRING function [1:30] must be placed inside the parentheses of the CENTER function. Putting the SUBSTRING function outside the parentheses does not change the item length.

CENTURY

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Date	Numeric	✔	✔	✔

Extracts the century from a date item or expression.

Syntax

CENTURY([**date-expression**[**,century-expression, start-year-expression**]])

If no parameters are used, the function returns the value specified in the DEFAULT CENTURY option of the SYSTEM OPTIONS statement.

Limit: The function returns a value from 1 to 99.

date-expression

If the date has a non-zero century, the century from the date-expression is returned, regardless of the other parameters specified.

If the date-expression has a century value of zero and no other parameters are specified, the return value will be determined using the values specified in the INPUT CENTURY option of the SYSTEM OPTIONS statement. If INPUT CENTURY is not specified, the function returns the DEFAULT CENTURY.

If the date-expression has a century value of zero and both century-expression and start-year-expression are specified, the function uses these to calculate the return value.

century-expression

The century to be used if the year of the date-expression is equal to or greater than the year of the start-year-expression. The value of century + 1 will be used if the year of the date-expression is less than the year of the start-year-expression.

Limit: If the century-expression is specified, the start-year-expression must also be used.

start-year-expression

The lower limit of the century window. The upper limit is always 99.

Limit: If the start-year-expression is specified, the century-expression must also be used.

Examples

If SYSDATE is a century-included date, the following example would return the current century from the SYSDATE. If SYSDATE is a century-excluded date, PowerHouse uses the INPUT CENTURY option of the SYSTEM OPTIONS statement to determine the century. If the INPUT CENTURY option is not used, PowerHouse returns the DEFAULT CENTURY.

```
Input: CENTURY(SYSDATE)
Result: the current century
```

If no parameters are used, the function returns the value specified in the DEFAULT CENTURY option of the SYSTEM OPTIONS statement. For example:

```
Input: CENTURY()
Result: the DEFAULT CENTURY
```

In the following example, the function returns "19" because the year, 97, is greater than 76, the start-year.

```
Input: CENTURY(971231,19,76)
Result: 19
```

In the next example, the function returns "20" because the year, 00, is less than 76, the start-year. The function uses the century-expression and start-year-expression parameters to calculate the return value.

```
Input: CENTURY(001231,19,76)
Result: 20
```

When the century and start year parameters are not specified, PowerHouse uses the INPUT CENTURY option of the SYSTEM OPTIONS statement to determine the century. If the INPUT CENTURY option is not used, PowerHouse returns the default century.

```
Input: CENTURY(980127)
Result: 19 ( assuming INPUT CENTURY 19 FROM 85)
```

CHARACTERS

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	Numeric	String	✔	✔	✔

Specifies an item that is addressed as a character string.

Syntax

CHARACTERS(numeric-item)

numeric-item

Specifies the item that is to be addressed as a character string.

Discussion

The CHARACTERS function allows the numeric item to be concatenated to the binary representation of a character item so that it can be used to combine common selection items. No internal data conversion occurs; therefore, the bit pattern remains unchanged. This allows you to use numeric items as parameters of functions that require character-type parameters.

Limit: Expressions are not allowed.

Examples

You can use the characters function with many PowerHouse functions that require character-type parameters. For example, you can turn on highlighting on DEC terminals by combining numeric items with character items or building strings of nonprinting characters.

```
> DEFINE ESC INT*2 = 27
> DEFINE ESCAPE CHAR*1 = CHAR(ESC)
> DEFINE BOLD CHAR*10 = ESCAPE + "[1m"
> DEFINE NOBOLD CHAR*10 = ESCAPE + "[0m"
```

You can also combine parts of an index from individual fields where one or more of the fields is numeric:

```
> FILE MIXED
> ACCESS VIA PUREINDEX &
>   USING FIRSTPART + CHARACTERS(SEQNO) &
>   REQUEST FIRSTPART, SEQNO
```

CHARACTER_LENGTH|CHAR_LENGTH

Type	Input	Result	QDESIGN	QUIZ	QTP
SQL-DMF	String	Numeric	✔	✔	✔

Returns the size of a string expression in characters.

Syntax

CHARACTER_LENGTH(string-expression)

CHAR_LENGTH(string-expression)
Limit: Valid only in SQL.

string-expression

Specifies the input string.

CHECKSUM

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	Numeric	✔	✔	✔

Returns a checksum for a string.

Syntax

CHECKSUM(string-expression)

Range: -32768 to 32767

Default: INTEGER SIGNED SIZE 2 item

string-expression

Specifies the string for which a checksum is to be generated.

The CHECKSUM function takes a string-expression and returns an integer value (a checksum) for that string. The value returned can be used later to determine if the string has been changed.

Discussion

Calculating Check Digits with the CHECKSUM Function

You can use the CHECKSUM function to calculate a check digit for data items such as account numbers. A check digit is a single digit that you compute using a specific algorithm. If you use a check digit to form part of a data item, you can check the validity of the item by using the algorithm that generated the check digit.

For example, when an account is first created, the checksum function can be used in conjunction with the absolute function, the mod function, and the characters function to calculate a check digit. If you attach the check digit to the end of the original account number, you can later use it for data validation. Suppose that you want to create a new account number, and you begin with the number 6478, which is stored in the item account-number. Use the checksum function to generate a check digit for this number, as in

```
ABSOLUTE(MOD(CHECKSUM(CHARACTERS(ACCOUNT-NUMBER)),10))
```

The MOD function returns the last digit of the value returned by the CHECKSUM function, and the ABSOLUTE function ensures that this digit is positive. Attach this check digit (in this case, the number 3) to the end of the original number, yielding a final account number 64783.

Using this type of account numbering system, you can easily detect the entry of invalid account numbers without accessing your files. When an account number is entered, you split it into two parts: one part is composed of all the digits except the final digit, and the other part is composed solely of the final digit. The check digit of the first part is recalculated using the method just described and compared to the final digit. If the recomputed check digit and the final digit of the number entered are not identical, an invalid account number has been entered.

Note: This type of scheme does not detect all data entry errors. In the previous example, if an even number of errors were made entering the account number, it is possible that the same checksum would be produced. Consequently, you should not use it to replace lookup options, which verify data against a file of acceptable entries.

Generating Compressed Indexes with the CHECKSUM Function

You can use the checksum function to generate compressed indexes for efficient access to high-volume online files. For example, instead of using an index of 30-character names, you can use the checksum function to generate much shorter indexes, as in the following example:

```
> SCREEN CHECKSUM
> TEMPORARY TEMPNAME CHARACTER*30 RESET AT MODE
```

```
> TEMPORARY CHECKNAME NUMERIC*1 RESET AT MODE
> FILE EMPLOYEES
> ACCESS VIA NAMEINDEX USING &
>     (NAME[1:1] + NAME[3:1] + NAME[5:1] + NAME[7:1] &
>     + CHARACTERS (CHECKNAME)) REQUEST LASTNAME
> ITEM NAMEINDEX FINAL &
>    (NAME[1:1] + NAME[3:1] + NAME[5:1] + NAME[7:1] &
>    + CHARACTERS (CHECKNAME))
> SELECT IF LASTNAME OF EMPLOYEES = TEMPNAME
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
.
.
.
.
.
.
> PROCEDURE PATH
>  BEGIN
>    REQUEST NAME OF EMPLOYEES
>     IF PROMPTOK
>      THEN LET PATH = 1
>     IF PATH = 0
>      THEN ERROR "KEY REQUIRED."
>  END
> PROCEDURE POSTPATH
>  BEGIN
>    LET TEMPNAME = NAME OF EMPLOYEES
>    LET CHECKNAME = CHECKSUM(NAME)
>  END
```

The temporary item, CHECKNAME, is set to the value returned by the CHECKSUM function in the POSTPATH procedure. The CHARACTERS function in the ACCESS statement then addresses the value in the temporary item, CHECKNAME, as a character string. The ITEM statement also uses the CHARACTERS function for the same purpose when assigning the final value to NAMEINDEX.

The ITEM statement with the FINAL option assigns a value to the abbreviated index, NAMEINDEX, when a data record is updated. An index composed of abbreviated names occupies less space than an index composed of full names. While the abbreviated names are not as unique as the full names, the degree of uniqueness is sufficient for most applications. When Find mode is initiated, the user is prompted at the NAME field, but data is retrieved via the abbreviated names in the index, NAMEINDEX, using the value of the DEFINED-NAMEINDEX item. If duplicate names exist in the record-structure's associated file, the name that the user entered is used to select the correct data record. The name that the user enters in the NAME field is overwritten and thus lost. Therefore, before data record selection can occur, you must use the POSTPATH procedure. Before the NAME field is overwritten, the POSTPATH procedure saves the name in the TEMPNAME item, and that item is subsequently used for record selection.

Using the CHECKSUM Function for Data Record Security

You can also use the CHECKSUM function for security purposes. By comparing an original checksum with a recomputed value, you can detect whether or not data has been altered. When a checksum is used for this purpose, you should calculate a checksum for a part of each data record rather than for the entire data record.

For the best possible security, use the CHECKSUM function together with the ENCRYPT function. For example, suppose you want to store encrypted passwords in a file of employee data records. When you originally enter employee data, you can concatenate the source password before it is encrypted with the employee's name, compute a checksum for the resulting string, and store the checksum inconspicuously somewhere else in the data record. When you later retrieve employee data records, you can decrypt the password, concatenate it with the employee's name, and recompute the checksum. You can then compare this new checksum to the original value. If they are not identical, you know that the original data has been altered.

Combining the ENCRYPT and CHECKSUM functions can provide a high degree of security. Security can be further enhanced by imaginative applications of the NCONVERT and substring extract functions in conjunction with the CHECKSUM function.

Example

```
Input: CHECKSUM("Top Secret")
Result: 17913

Input: > ACCESS ACCOUNTS-REC
       > DEFINE ITEMX INT*4 = &
       >   ABS(MOD(CHECKSUM(ASCII(CUST-NO)),10))
```

COMMANDCODE

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔		

Returns the status code issued when an operating system command is executed.

Syntax

COMMANDCODE

Discussion

The COMMANDCODE function returns information on error messages issued when trying to execute an operating system command. If the COMMANDCODE function returns an empty string, then the command returns an exit status of 0.

COMMANDCODE cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

MPE/iX

The status code is either 0 (OK), CIWARN# (warning), or CIERR# (error). For example, `IF COMMANDCODE = "CIERR907"` identifies if a LISTF was successful or not.

OpenVMS

The format of the returned string is FACILITY-SEVERITY-ID. The string is converted from the value saved in the DCL reserved symbol $STATUS when the DCL command completes execution.

UNIX

If it's a string in the form "UX-WRN #xxxx", the command exits with the status xxxx (in hexadecimal). If it's of the form #xxxx", the command terminates with a signal, returning status xxxx (in hexadecimal). If the string is of the form "UX-ERR #nnnn", the command cannot be executed because of a UNIX error number nnnn.

Windows

If it's a string in the form "WIN-WRN #xxxx", the command exits with the status xxxx (in hexadecimal). If it's of the form #xxxx", the command terminates with a signal, returning status xxxx (in hexadecimal). If the string is of the form "WIN-ERR #nnnn", the command cannot be executed because of a Windows error number nnnn.

COMMANDMESSAGE

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔		

Returns the text of any warning or error message that appears when the operating system command is executed.

Syntax

COMMANDMESSAGE

Discussion

Like the COMMANDCODE function, the COMMANDMESSAGE function returns information on error messages issued when trying to execute an operating system command. However, the information that the COMMANDMESSAGE function returns is more detailed.

If the COMMANDMESSAGE function returns an empty string, the command returns an exit status of 0.

COMMANDMESSAGE cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

MPE/iX

For errors, the MPE/iX error message is returned, for example

```
"Non-existent file (CIERR 907)."
```

OpenVMS

The format of the returned string is FACILITY-SEVERITY-ID.

UNIX

If the COMMANDMESSAGE function returns an empty string, the command returns an exit status of 0. For exit status codes other than 0, a string "Command exited on 'nnn'signal" is returned, where nnn is a description of the signal (for example, Hangup). For exec errors, the UNIX error code description is returned.

Windows

If the COMMANDMESSAGE function returns an empty string, the command returns an exit status of 0. For exit status codes other than 0, a string "Command exited on 'nnn'signal" is returned, where nnn is a description of the signal (for example, Hangup). For exec errors, the Windows error code description is returned.

COMMANDSEVERITY (OpenVMS)

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔		

Returns the severity of the most recently executed operating system command.

Syntax

COMMANDSEVERITY

Discussion

The value is extracted from the DCL symbol $SEVERITY and is one of

I Information

S Success

W Warning

E Error

F Fatal

COMMANDSEVERITY cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

COMMANDSTATUS (OpenVMS)

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Numeric	✔		

Returns the numeric value saved in the DCL reserved symbol $STATUS when the COMMAND verb completes execution.

Syntax

COMMANDSTATUS

Discussion

COMMANDSTATUS cannot be

- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

CONTENTS

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	String	String	✔	✔	✔

Returns the contents of a blob as a character string.

Syntax

CONTENTS(item)

Limit: The maximum size of the resulting string is 32,767 bytes.

item

Specifies a blob from a relational database.

Discussion

The CONTENTS function ensures that PowerHouse works with the blob contents rather than the blob id. Since blob ids are no longer used as of PowerHouse 8.4xC, the CONTENTS function is no longer required. However, it is retained for backwards compatibility

For more information about blobs, see (p. 345).

Example

Use the CONTENTS function to copy the contents from one blob to another blob:

```
LET TOBLOB = CONTENTS( FROMBLOB )
```

or, to concatenate a string to the end of a blob and assign the result to another blob.

```
LET TOBLOB = CONTENTS( FROMBLOB ) + "a string"
```

DATE

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Numeric	Date	✔	✔	✔

Calculates the date that is a specified number of days from the base date.

Syntax

DATE(numeric-expression)

DATE(numeric-expression, date-expression)

numeric-expression

Specifies a number of days.

Part of a day is expressed as a fraction. For example, 100.5 means one hundred and a half days.

date-expression

Specifies the base date.

The base date may be century included or excluded, and may include a time in the form "hhmmsstt".

When the date-expression is not used, and the numeric-expression is positive, the resulting date is greater than or equal to January 1, 1900 (which is a Monday).

When the date-expression is not used, and the numeric-expression is negative or zero, the resulting date is less than or equal to December 31, 1899 (which is a Sunday).

Discussion

The DATE function calculates the date that is a number of days from the base date. The number of days is specified by the numeric-expression. If the base date is not supplied using the date-expression, the default base date of December 31, 1899 is used.

If the CENTURY INCLUDED system option is in effect, the DATE function returns an 8-digit date.

If the CENTURY EXCLUDED system option is in effect, the function returns a six-digit date. If the result is not within the default century, a data conversion error occurs.

Examples

If the target of the function does not allow a date/time result, the fractional portion of the numeric-expression is ignored in the result. For example,

```
Input: DEFINE X DATE = DATE(100.5)
Result: 1900/04/10
```

In this example, X is defined as DATETIME. Therefore, half a day (.5) is included in the result as 12:00:00.00.

```
Input: DEFINE X DATETIME = DATE(100.5)
Result: 1900/04/10 12:00:00.00
```

A time portion in the form "hhmmsstt" can be specified on the date-expression. For example,

```
Input: DEFINE X DATETIME = DATE(100.5,19990101.12304478)
Result: 2000/04/10 00:30:44.78
```

This DEFINE statement calculates the date 15 days from today:

```
DEFINE NEWDATE DATE = DATE(DAYS(SYSDATE) + 15)
```

If the CENTURY EXCLUDED system option is in effect, the following will cause a data conversion error because the function result is not within the default century. The default century is 19.

```
DEFINE X DATE=DATE(100,19991231)
```

The following examples show the default base date in both date and date/time formats:

```
Input: DEFINE X DATE = DATE(0)
Result: 1899/12/31

Input: DEFINE Y DATETIME = DATE(0)
Result: 1899/12/31 00:00:00.00
```

DATEEXTRACT

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Date	Numeric	✔	✔	✔

Extracts part of a date item, such as the month or hour.

Syntax

DATEEXTRACT(date-item,extract-option)

date-item

Specifies a date item.

extract-option

You can specify the following options for the DATEEXTRACT function:

Option	Returns
YEAR	a 4-digit year
MONTH	a number between 1 and 12
DAY	a number between 1 and 31
HOUR	a number between 0 and 23
MINUTE	a number between 0 and 59
SECOND	a number between 0 and 59
DATE	the date portion of the specified item as a number in the form YYYYMMDD
TIME	the time portion of the specified time as a number in the form HHMMSSTTT (where TTT represents thousandths of seconds) For the date item types that do not store the time portion (PHDATE and JDATE), TIME will always be 0.

Discussion

When DATEEXTRACT is used to extract the TIME, 9 digits are returned representing hours, minutes, seconds, tenths of a second, hundredths of a second, and thousands of a second. However, the thousands of a second will always be zero because it is greater than the precision of the floating point value.

Example

To extract the month from the system date,

```
> DEFINE X DATE = SYSDATE
> DEFINE MONTHVALUE = DATEEXTRACT(X,MONTH)
```

DAYS

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Date	Numeric	✔	✔	✔

Returns a quantity of days from a date.

Syntax

DAYS(date-item)

date-item

Specifies the input date.

Discussion

If the century prefix is 19 or greater, DAYS converts the number of days since and including Monday, January 1, 1900. If the century prefix is less than 19, the date is converted to a negative number that represents the days prior to Sunday, December 31, 1899.

Examples

```
Input: DAYS(010101)
Result: 366

Input: DAYS(19010101)
Result: 366

Input: DAYS(18990101)
Result: -364
```

The following example will calculate the difference between two dates:

```
Input: DEFINE DIFF NUM*5 = DAYS(19920131) - DAYS(19911230)
Result: 32
```

The following example calculates the day of the week for START_DATE which could be any date or date-time item type.

```
> DEFINE DAY_NUMBER = MOD( DAYS( START_DATE ), 7 )
> DEFINE DAY_OF_WEEK CHAR*9 =CASE OF DAY_NUMBER &
>   WHEN 0 THEN "Sunday    " &
>   WHEN 1 THEN "Monday    " &
>   WHEN 2 THEN "Tuesday    " &
>   WHEN 3 THEN "Wednesday " &
>   WHEN 4 THEN "Thursday  " &
>   WHEN 5 THEN "Friday    " &
>   WHEN 6 THEN "Saturday   " &
```

As the base date is a Sunday, when DAY_NUMBER is 0 the date is a Sunday.

DECIMALTIME

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Date	Numeric	✔	✔	✔

Returns the fractional quantity of time between two dates.

Syntax

DECIMALTIME(date-item [,base-date])

date-item

Specifies the input date or date-time item.

base-date

Specifies a date to be used as the base in calculating the amount of time between the two dates.

Default: Sunday, December 31, 1899 00:00:00:00

Discussion

The DECIMALTIME function returns the time period between two dates expressed as a number of days, with the remainder expressed as a fraction of a day. The time portion of this function only applies to the DATETIME datatypes.

Example

In the following example, the START_DATE is set to June 7, 1993 18:00:00 and the BASE_DATE is set to June 6, 1993 12:00:00

```
> DEFINE INTERVAL NUMERIC = &
  DECIMALTIME(START_DATE,BASE_DATE)
```

The value assigned to INTERVAL is 1.25, which represents one day and 6 hours.

DECRYPT

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔	✔	✔

Decodes an encrypted key.

Syntax

DECRYPT(string-expression1,string-expression2)

string-expression1

Specifies the string to be decoded.

string-expression2

Specifies the decryption key.

Limit: The decryption key must be the same as the one that was used originally during encryption.

Discussion

For more information about decryption and encryption, see (p. 402).

Example

```
Input: DECRYPT(CODE,"key")
Result: a decrypted value for the encrypted item CODE
```

DELETESYSTEMVAL (MPE/iX, UNIX, and Windows)

For DELETESYSTEMVAL (**Open VMS**), see (p. 397).

Type	Input	Result	QDESIGN	QUIZ	QTP
LF	String	Boolean	✔	✔	✔

Allows the deletion of values defined at the operating system level.

Syntax

DELETESYSTEMVAL(string-expression)

string-expression

Specifies the name of the file equation (**MPE/iX**) or environment variable (**UNIX, Windows**) to be deleted.

Discussion

The DELETESYSTEMVAL, GETSYSTEMVAL, and SETSYSTEMVAL functions support the deletion, retrieval, and assignment of operating system values. This provides execution-time control of values through the definitions made at the operating system level and allows values to be passed between PowerHouse components.

DELETESYSTEMVAL returns a logical result of True if it successfully deletes the environment variable. Otherwise it returns False. As this is a logical function, it must be used in a conditional expression.

At parse time, use of the DELETESYSTEMVAL function will cause a syntax error if either of the following conditions exist:

- the **noaccess** program parameter is specified
- the OSACCESS resource file option equals OFF

Because the function accesses operating system values, its use is not permitted when operating system access is restricted.

There is no effect on the function in compiled screens, reports and runs if the **noaccess** program parameter or OSACCESS=OFF resource file statement is used at runtime.

Examples

The following QUICK example attempts to delete a variable named ENV1. If it can't be deleted, the error message is displayed.

```
.
.
.
> PROCEDURE EXIT
>   BEGIN
> ;   Clean up environment variable
>     IF NOT DELETESYSTEMVAL("ENV1")
>     THEN BEGIN
>     ERROR "Could not delete systemvalue ENV1."
>     END
.
.
.
```

The following example shows how a global value is created and referenced in each of the PowerHouse components: QUICK and QUIZ.

```
> SCREEN REPORT_MENU MENU NOMODE ACTION LABEL &
> "Enter 1 for Employee address, 2 for Positions." &
>   AT 3,2
```

```
> PROCEDURE DESIGNER 1
>   BEGIN
>     IF NOT SETSYSTEMVAL ("REP_NUM",(ASCII( 1)))
>       THEN BEGIN
>        ERROR "Could not set variable."
>       END
>     ELSE
>       BEGIN
>        CLEAR SCREEN
>        RUN COMMAND "quiz auto=REPORT1"
>        REFRESH ALL
>       END
>   END
> PROCEDURE DESIGNER 2
>   BEGIN
>     IF NOT SETSYSTEMVAL ("REP_NUM",(ASCII(2)))
>       THEN BEGIN
>        ERROR "Could not set variable."
>       END
>     ELSE
>       BEGIN
>        CLEAR SCREEN
>        RUN COMMAND "quiz auto=REPORT2"
>        REFRESH ALL
>       END
>   END
> PROCEDURE EXIT
>   BEGIN
> ;   Clean up variable
>     IF NOT DELETESYSTEMVAL("REP_NUM")
>      THEN BEGIN
>       ERROR "Could not delete variable."
>      END
>   END
```

The values are then picked up by QUIZ.

```
> ; REPORT 1
> ACCESS EMPLOYEES
> DEFINE REP_NUM NUM*3 =  &
>   NCONVERT(GETSYSTEMVAL("REP_NUM"))
> SORT ON LASTNAME
> INITIAL HEADING "This is REPORT " REP_NUM
> REPORT EMPLOYEE LASTNAME FIRSTNAME STREE
> GO
> ;REPORT 2
> ACCESS EMPLOYEES LINK TO POSITION
> DEFINE REP_NUM NUM*3 = &
>     NCONVERT(GETSYSTEMVAL("REP_NUM"))
> SORT ON POSITION
> INITIAL HEADING "This is REPORT " REP_NUM
> REPORT EMPLOYEE LASTNAME FIRSTNAME TITLE
> GO
```

The following example shows how a global value is created in QUICK. The QTP run executed depends on the value of the QTP_PARMS variable.

```
> SCREEN REPORT_MENU MENU NOMODE ACTION LABEL &
>   "Enter 1 to begin" AT 3,4
> TEMPORARY REPORT_REQ CHAR*10
> FIELD REPORT_REQ LABEL &
>   "Enter required report name: " AT 5,4 &
>   DATA AT 6,15  &
>   HELP  &
> "Enter 1 for address report 2 for position report."
> PROCEDURE DESIGNER 1
>   BEGIN
>     ACCEPT REPORT_REQ
>     IF NOT SETSYSTEMVAL("QTP_PARMS",&
>       ("REPORT" +  REPORT_REQ ))
```

```
>     THEN BEGIN
>       ERROR "Could not set variable."
>     END
>     ELSE RUN COMMAND "qtp auto=$QTP_PARMS" &
>       REFRESH ALL
>   END
> PROCEDURE EXIT
>   BEGIN
> ;   Clean up environment variable
>     IF NOT DELETESYSTEMVAL  &
>       ("QTP_PARMS")
>      THEN BEGIN
>       ERROR "Could not delete variable."
>      END
>   END
> GO
```

DELETESYSTEMVAL (OpenVMS)

For DELETESYSTEMVAL (**MPE/iX, UNIX,** and **Windows**), see (p. 394).

Type	Input	Result	QDESIGN	QUIZ	QTP
LF	String	Boolean	✔	✔	✔

Allows the deletion of values defined at the operating system level.

Syntax

For OpenVMS logical names:

DELETESYSTEMVAL
(string-expression1 [,LOGICAL [,string-expression2]])

For DCL global symbols:

DELETESYSTEMVAL(string-expression1, SYMBOL)

string-expression1

Specifies the name of the OpenVMS logical name or DCL symbol to be deleted. If the LOGICAL or SYMBOL keyword is not explicitly used, a logical name is assumed.

string-expression2

Optionally specifies the logical name table to delete the logical name from, such as LNM$JOB or LNM$GROUP.

Default: LNM$PROCESS

LOGICAL

Specifies retrieval of an OpenVMS logical name. If the LOGICAL or SYMBOL keyword is not explicitly used, a logical name is assumed.

SYMBOL

Specifies retrieval of a DCL symbol. Local symbols are retrieved before global symbols.

Discussion

The DELETESYSTEMVAL, GETSYSTEMVAL, and SETSYSTEMVAL functions support the deletion, retrieval, and assignment of operating system values. This provides execution-time control of values through the definitions made at the operating system level and allows values to be passed between PowerHouse components.

DELETESYSTEMVAL returns a logical result of True if it successfully deletes the environment variable. Otherwise it returns False. As this is a logical function, it must be used in a conditional expression.

At parse time, use of the DELETESYSTEMVAL function will cause a syntax error if either of the following conditions exist:

- the **noaccess** or **nodcl** program parameter is specified
- the OSACCESS resource file option equals OFF

Because the function accesses operating system values, its use is not permitted when operating system access is restricted.

There is no effect on the function in compiled screens, reports and runs if the **noaccess** or **nodcl** program parameter or OSACCESS OFF resource file statement is used at runtime.

Examples

The following QUICK example attempts to delete a logical named LOG1 in the LNM$JOB logical name table. If the logical name can't be deleted, the error message is displayed.

```
.
.
.
> PROCEDURE EXIT
>   BEGIN
> ;   Clean up logical name
>     IF NOT DELETESYSTEMVAL("QUIZ_PARMS", &
>       LOGICAL, "LNM$JOB")
>     THEN BEGIN
>       ERROR "Could not delete logical name."
>     END
.
.
.
```

The following example shows how a global value may be created and referenced in QUICK and QUIZ.

```
> SCREEN REPORT_MENU MENU NO MODE ACTION LABEL &
> "Enter 1 for Employee address list 2 for Position list:" &
>   AT 3, 2
> PROCEDURE DESIGNER 1
>   BEGIN
>     IF NOT SETSYSTEMVAL ("REP_NUM", (ASCII( 1)), &
>        LOGICAL, "LNM$JOB")
>     THEN BEGIN
>       ERROR "Could not set logical name."
>      END
>     ELSE
>      BEGIN
>        CLEAR SCREEN
>        RUN COMMAND "QUIZ AUTO=REPORT1"
>        REFRESH ALL
>      END
>   END
> PROCEDURE DESIGNER 2
>   BEGIN
>     IF NOT SETSYSTEMVAL ("REP_NUM", (ASCII (2)), &
>       LOGICAL, "LNM$JOB")
>      THEN BEGIN
>        ERROR "Could not set logical name."
>       END
>      ELSE
>       BEGIN
>     CLEAR SCREEN
>     RUN COMMAND "QUIZ AUTO=REPORT2"
>     REFRESH ALL
>     END
>   END
> PROCEDURE EXIT
>   BEGIN
> ;    Clean up logical name
>      IF NOT DELETESYSTEMVAL("REP_NUM", &
>         LOGICAL, "LNM$JOB")
>       THEN BEGIN
>         ERROR "Could not delete logical name."
>       END
>    END
```

The values are then picked up by QUIZ.

```
> ; REPORT 1
> ACCESS EMPLOYEES
> DEFINE REP_NUM NUM*3 = &
>   NCONVERT(GETSYSTEMVAL("REP_NUM"))
```

```
> SORT ON LASTNAME
> INITIAL HEADING "This is REPORT " REP)NUM
> REPORT EMPLOYEE LASTNAME FIRSTNAME STREET
> GO

> ;REPORT 2
> ACCESS EMPLOYEES LINK TO OSITION
> DEFINE REP_NUM NUM*3 = &
>   NCONVERT(GETSYSTEMVAL("REP_NUM"))
> SORT ON POSITION
> INITIAL HEADING "This is REPORT " REP_NUM
> REPORT EMPLOYEE LASTNAME FIRSTNAME TITLE
> GO
```

The following example shows how a global value may be created and referenced in QUICK and QTP.

```
> SCREEN REPORT_MENU MENU NOMODE ACTION LABEL &
>   "Enter 1 to begin" AT 3,4
> TEMPORARY REPORT_REQ CHAR*10
> FIELD REPORT_REQ LABEL &
>   "Enter required report name: " AT 5,4 &
>   DATA AT 6,15 &
>   HELP &
> "Enter 1 for address report 2 for position report."
> PROCEDURE DESIGNER 1
>   BEGIN
>      ACCEPT REPORT_REQ
>      IF NOT SETSYSTEMVAL("QTP_PARMS", &
>        ("REPORT" + REPORT_REQ ) &
>         , LOGICAL, "LNM$JOB")
>      THEN BEGIN
>        ERROR "Could not set logical name."
>      END
>      ELSE RUN COMMAND "QTP AUTO=QTP_PARMS"
>        REFRESH ALL
>      END
> PROCEDURE EXIT
>   BEGIN
> ;   Clean up logical name
>     IF NOT DELETESYSTEMVAL &
>        ("QTP_PARMS",LOGICAL, "LNM$JOB")
>      THEN BEGIN
>        ERROR "Could not delete logical name."
>      END
>   END
> GO
```

The values are then picked up by QTP which creates a temporary subfile to pass to QUIZ to report on.

```
> ;REPORT 1
> SET NOSTATISTICS
> ACC EMPLOYEES
> SUBFILE HOLD INCL EMPLOYEE, FIRSTNAME, &
>    LASTNAME, STREET, CITY, POSTALZIP
> GO
> $QUIZ AUTO=HOLD
> EXIT

> ;REPORT 2
> SET NOSTATISTICS
> ACC EMPLOYEES
> SUBFILE HOLD INCLUDE EMPLOYEE, FIRSTNAME, &
>    LASTNAME, POSITION
> GO
> $QUIZ AUTO=HOLD
> EXIT
```

QUIZ report to report on the subfile created in the QTP run.

```
> ACC *HOLD
> REP ALL
> GO
> EXIT
```

DOWNSHIFT

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔	✔	✔

Shifts uppercase characters to lowercase characters.

Syntax

DOWNSHIFT(string-expression)

string-expression

Specifies the string to be downshifted.

Discussion

Non-alphabetic characters remain unchanged. If present in the dictionary, the alternate language downshift table is used.

Example

```
Input: DOWNSHIFT("TEXT")
Result: text
```

The following example formats LASTNAME with the first letter in uppercase and the remaining letters in lowercase:

```
> DEFINE NAME CHAR*20 = &
>  UPSHIFT(LASTNAME[1:1]) + &
>  DOWNSHIFT(LASTNAME[2:19])
```

ENCRYPT

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔	✔	✔

Creates an encryption key.

Syntax

ENCRYPT(string-expression1,string-expression2)

string-expression1

Specifies the string to be encoded.

string-expression2

Specifies the encryption key. Encryption keys can contain nonprinting control codes as well as characters.

Discussion

Encryption Guidelines

Encryption goes beyond PowerHouse application security. Application security protects sensitive information from unauthorized access from within PowerHouse. However, file-dump utilities and programs written in other languages can be used to make copies of the files regardless of PowerHouse application security. You can help prevent unauthorized access to data at the operating system level by encrypting sensitive PowerHouse data.

You will achieve the best results when encrypting data if you

- use double encryption keys that contain at least five characters
- ensure that double encryption keys are as distinct from each other as possible, with a minimum of shared characters
- never use identical keys for double encryption
- always truncate trailing blanks from string expressions that are used as encryption keys
- use encryption keys that are inconspicuous. For example, use strings that are already contained in the source code, such as field labels, report headings, help messages, and so on.

Double Encryption

Double encryption refers to the nesting of the encrypt function, and ensures better security than single encryption. When using encryption, you should always doubly encrypt items, as in

```
> DEFINE ENCODEDITEM CHARACTER*13 &
>   = ENCRYPT(ENCRYPT("SOURCE STRING","FIRST"),"SECOND")
```

Here the string "SOURCE STRING" is encrypted using the "FIRST" key. The result of this operation is in turn encrypted using the "SECOND" key, giving the string "FCDNYG:DDIACF".

In order to decode a doubly encrypted item, you must use double decryption by nesting the decrypt function. When you use double decryption, the sequence of the keys in the expression must be the reverse of the sequence used for encryption. In the previous example, you can decrypt the item encodeditem by entering

```
> DEFINE DECODEDITEM CHARACTER*13 &
>   = DECRYPT(DECRYPT(ENCODEDITEM,"SECOND"),"FIRST")
```

The encoded item ENCODEDITEM is decrypted using the "SECOND" key. The result of this operation is in turn decrypted using the "FIRST" key.

Examples

To encrypt only the data in a field, excluding trailing spaces, use the TRUNCATE function before you encrypt.

To decrypt correctly, the sequence of the encryption keys must be reversed.

```
>   SCREEN ENCRYPT ACTIVITIES FIND
>   FILE EMPLOY1 OCCURS 10 TIMES
>
>   DEFINE CODED CHARACTER*20= &
> ENCRYPT(ENCRYPT(TRUNC(LASTNAME),"secret"),"agent")
>
>   DEFINE DECODED CHARACTER*20 = &
>      DECRYPT(DECRYPT(TRUNC(CODED),"agent"),"secret")
>
>   TITLE "Last Name"  at 4,4
>   TITLE "Encrypted"  at 4,26
>   TITLE "Decrypted"  at 4,55
>   SKIP 1
>   CLUSTER OCCURS WITH EMPLOY1
>   ALIGN (,,4) (,,26)(,,55)
>   FIELD LASTNAME OF EMPLOY1
>   FIELD CODED
>   FIELD DECODED
>   GO
```

The following screen shows the LASTNAME field values and the encrypted values. The third column displays the results of the correct decryption:

MODE: F ACTION:

Last Name	*Encrypted*	*Decrypted*
Burton	*Pwth"{*	*Burton*
BURDEN	*PWTXT [burden*	*BURDEN*
BARTON	*PCTH^[*	*BARTON*
NEALE	*\GGPT*	*NEALE*
LYNCH	*^[H_Y*	*LYNCH*
LYON	*^[ir*	*LYON*
STANLEY	*AVGR]PM*	*STANLEY*
Smith	*Aoohy*	*Smith*
Jones	*Xmhyb*	*Jones*
Palmer	*Bcjqtg*	*Palmer*

If each data record in an employees file contains the items address, civilstatus, and password, and employee passwords are to be encrypted, then the "CIVIL" and "ADDRESS" strings are good choices for double encryption keys, as in

```
> define ENCRYPTITEM character*13 &
>  = ENCRYPT(ENCRYPT(PASSWORD,"CIVIL"),"ADDRESS")
```

These encryption keys meet the minimum length recommended and contain no characters in common.

EXTRACT

Type	Input	Result	QDESIGN	QUIZ	QTP
SQL-DMF	Date	Numeric	✔	✔	✔

Extracts the requested part of a datetime or interval value expression.

Syntax

EXTRACT(extract-option FROM date-expression)
Limit: Valid only in SQL.

extract-option

You can specify the following options for the EXTRACT function:

Option	Returns
YEAR	a 4-digit year
MONTH	a number between 1 and 12
DAY	a number between 1 and 31
HOUR	a number between 0 and 23
MINUTE	a number between 0 and 59
SECOND	a number between 0 and 59

date-expression

The date-expression must be a datetime or interval expression.

FIRST

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Date, Numeric, String	✔		

Returns the value of the item in the first occurrence of the associated file on the screen.

Syntax

FIRST(item)

item

Refers to a record item (an item declared in the data dictionary), a predefined item, a temporary item, or a defined item.

item [OF file]

Discussion

FIRST cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

If one of the items in a record on a multiple-record screen always has the same value for each screenload of data, you can position the field for the item outside of the cluster boundaries. If a field is outside of the cluster, QDESIGN treats the field as part of the first occurrence of the cluster.

To specify that the value of each occurrence of the item is to take the same value as the first occurrence of the item, use the system function first, as in

```
> SCREEN BILL
> FILE BILLINGS OCCURS 14
> ITEM EMPLOYEE FINAL FIRST(EMPLOYEE)
> FILE EMPLOYEES REFERENCE
> TITLE "BILLING BY EMPLOYEE" AT 1,31
> FIELD EMPLOYEE OF BILLINGS LOOKUP ON EMPLOYEES
> CLUSTER OCC WITH BILLINGS
> FIELD BILLING OF BILLINGS
> FIELD PROJ OF BILLINGS
.
.
.
```

FLOOR

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Numeric	Numeric	✔	✔	✔

Rounds an integer down.

Syntax

FLOOR(numeric-expression)

numeric-expression

Specifies the input number.

Discussion

The FLOOR function decreases a fractional value to the next lowest integer. Whole numbers are not decreased.

This function is commonly used to round monetary values down to the nearest integer.

Examples

```
Input: FLOOR(79.04)
Result: 79

Input: FLOOR(-34.44)
Result: -35

Input: DEFINE ITEMX NUM*8 = FLOOR(45.6 + .5)
Result: 46
```

FORMATNUMBER

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Numeric and String	String	✔	✔	✔

Formats a number according to a format string.

Syntax

FORMATNUMBER(numeric-expression,string-expression1)

FORMATNUMBER(numeric-expression,[string-expression1], string-expression2)

FORMATNUMBER(numeric-expression,[string-expression1], [string-expression2], string-expression3)

numeric-expression

Specifies the number to be formatted.

string-expression1

Specifies the format string to be used if the number is positive. Also specifies the format string to be used if the number is zero and string-expression3 is not specified. If the number is positive and string-expression 1 is not specified, overflow characters (#) are displayed.

string-expression2

Specifies the format string to be used if the number is negative. If the number is negative and string-expression 2 is not specified, overflow characters (#) are displayed

string-expression3

Specifies the format string to be used if the number is zero. If the number is zero and string-expression3 is not specified, the number is formatted using string-expression1. If string-expression1 is not specified, overflow characters (#) are displayed.

Discussion

FORMATNUMBER takes a number, formats it based on a format string, and returns a variable length-string. A format string is a string of characters that indicates how the number is to be formatted.

FORMATNUMBER recognizes both the integer and fractional portions of the number to be formatted. The numeric expression is a floating point number and the designer must ensure that the number input to the function is exactly the number desired. PowerHouse doesn't perform any scaling of the number before applying the format string. The number is rounded automatically to the number of decimal places indicated by the format string.

The last string expression must be followed by a closing parenthesis. Each string expression is optional except the last (that is, the last comma must be followed by a string expression).

If a negative number rounds to zero, the zero format string is used. If there is no zero format string, the negative format string is used. If there is no negative format string, overflow characters (#) are displayed.

If the number to be formatted is NULL, the function result is NULL.

Format String Special Characters

The following special characters are valid in a format string:

0	Zero substitution	{}	Fill text
?	Blank Substitution	”	Quoted text
#	NULL substitution	""	Quoted text
.	Decimal character	\	Escape character

Substitution Characters

A substitution character identifies a location where a digit from the number is substituted into the format string. The type of substitution character determines what is displayed when the digit is zero. Non-zero digits are always displayed. If there aren't enough substitution characters to display the non-zero digits to the left of the decimal place, overflow characters (#) are displayed. Non-zero digits to the right of the decimal place are automatically rounded to the number of decimal digits indicated by the format string.

Zero Substitution Character

A substitution character of zero (0) indicates that a leading or trailing zero should be displayed. It is used to force the display of leading and trailing zeros. For example:

```
    FORMATNUMBER(amount, "00,000.00")
returns "01,234.56" if amount is 1234.56
returns "00,123.40" if amount is 123.4
returns "00,000.00" if amount is 0
```

Blank Substitution Character

A substitution character of a question mark (?) indicates that a leading or trailing zero should be displayed as a blank. It is used to suppress the display of leading and trailing zeros. For example:

```
    FORMATNUMBER(amount, "$??,??0.?? US")
returns "$ 1,234.56 US" if amount is 1234.56
returns "$   123.4  US" if amount is 123.4
returns "$     0    US" if amount is 0
```

Note: When a digit is blank due to the blank substitution character, any text not enclosed in quotes that is between the blanked digit and the decimal or the nearest substitution character towards the decimal (whichever is closest) is also blanked. A blank is substituted for the zero digit and the blank takes a character position. Therefore, the dollar sign and the "US" do not float.

Text enclosed in quotes is never blanked out. For example:

```
    FORMATNUMBER(phone, "???'-'????")
returns "123-4567" if phone is 1234567
returns "   -    " if phone is 0
```

Null Substitution Character

A substitution character of a crosshatch (#) indicates that a leading or trailing zero should be discarded. It is used to provide for floating characters. For example:

```
    FORMATNUMBER(amount, "$##,##0.## US")
returns "$1,234.56 US" if amount is 1234.56
returns "  $123.4 US " if amount is 123.4
returns "    $0 US   " if amount is 0
```

Note: When a digit is discarded due to the null substitution character, any text not enclosed in quotes that is between the discarded digit and the decimal or the nearest substitution character (whichever is closest) is also discarded. In the second example, the comma is discarded. In the third example, the decimal is also discarded. A discarded digit or character does not take a character position. Therefore, the dollar sign and the "US" float.

The default for null substitutions is that the left side is blank filled to fill the number of positions discarded to the left of the decimal. The right side is blank filled to fill the number of positions discarded to the right of the decimal.

Text enclosed in quotes is never discarded. For example:

```
    FORMATNUMBER(id, "###'-'###'-'###")
returns "123-456-789" if id is 123456789
returns "   -123-456" if id is 123456
returns "      --123" if id is 123
returns "         --" if id is 0
```

Decimal Character

The decimal character is a period (.). It identifies the beginning of fractional digits within a format string.

The decimal character itself is displayed only if fractional digits are displayed (which depends on the substitution characters used). If text immediately follows the decimal character, that text is displayed instead of the decimal character. If the text is not enclosed in quotes, it is displayed only if fractional digits are displayed. If the text is enclosed in quotes, it is displayed regardless of whether fractional digits are displayed or not. When the decimal character or text not enclosed in quotes isn't displayed, it is blank filled if the next substitution character is a question mark (?) or discarded if the next substitution character is a crosshatch (#). For example:

```
    FORMATNUMBER(weight, "??0.?? Kg")
returns " 10.5  Kg" if weight is 10.5
returns " 10    Kg" if weight is 10.0

    FORMATNUMBER(weight, "??0.0? Kg")
returns " 10.0  Kg" if weight is 10.0

    FORMATNUMBER(weight, "??0.'.'?? Kg"
returns " 10.   Kg" if weight is 10.0

    FORMATNUMBER(weight, "##0. point ## Kg"
returns " 10 point 5 Kg " if weight is 10.5
returns " 10 Kg         " if weight is 10
returns "  0 Kg         " if weight is 0
```

There is no dictionary override for the decimal character in a format string. To display a comma rather than a decimal point, include the comma as text immediately following the decimal character, as in:

```
    FORMATNUMBER(amount, "??? ??0.,??")
returns " 12 345,67" if amount is 12345.67
```

If there is no decimal character in the format string, the logical decimal position is immediately to the right of the rightmost substitution character.

Only the first decimal character is interpreted as a decimal. Subsequent decimal characters are treated as text enclosed in quotes and will always be displayed.

Fill Text

The default fill character is a blank. Fill text is substituted into character positions discarded by null substitution. You can override the default fill character by enclosing text in braces ({}). For example:

```
    FORMATNUMBER(price, "{*}$###,##0.00")
returns "**$1,000.00" if price is 1000.00
returns "******$1.23" if price is 1.23
returns "******$0.00" if price is 0
```

Fill text characters are inserted in the same location as the fill text appears in the format string. The number of characters inserted is the number of characters discarded by null substitution. For example:

```
    FORMATNUMBER(value, "##{*}###.#")
returns "***12.3" if value is 12.3
returns "1*234.5" if value is 1234.5
```

If more than a single character is used as the fill text, individual characters from the fill text are used to fill discarded characters in a one-to-one correspondence. The fill text is repeated to fill the space of discarded characters. For example:

```
    FORMATNUMBER(amount, "{VOID } #,###,###,###"
returns "VOID 1,234,567" if amount is 1234567
returns "VOID V 123,456" if amount is 123456
returns "VOID VOID VOI " if amount is 0

    FORMATNUMBER(phone, "({___) ___-____}###) ###-####"
returns "(123) 456-7890" if phone is 1234567890
returns "(___) 123-4567" if phone is 1234567
returns "(___) ___-____" if phone is 0
```

When there are two fill text groups, the one on the left fills positions discarded to the left of the decimal, while the one on the right fills positions discarded to the right of the decimal. For example:

```
    FORMATNUMBER(volume, "US {*}###,##0.###{!} GAL")
returns "US **1,000!!!! GAL" if volume is 1000
returns "US ******1.23! GAL" if volume is 1.23
```

You can affect the alignment and justification of the formatted result by using an empty fill text string, that is, opening and closing braces with no text in between, or by positioning the decimal fill text to the left of the decimal character. For example:

```
    FORMATNUMBER(quantity, "{ }{ }##,###.###"
returns "      1.23" if quantity is 1.23
returns "      12.3" if quantity is 12.3
returns "     1,234" if quantity is 1234

    FORMATNUMBER(quantity, "{}#,###,##0")
returns "1,235" if quantity is 1234.5
returns "12" if quantity is 12
```

If there is an unmatched brace PowerHouse assumes there is a closing brace at the end of the format string. If there is an extra closing brace, it is treated as quoted text. Only two fill text groups are allowed. Subsequent fill text groups are treated as quoted text.

Quotes

Single or double quotes are used to enclose text that is not to be interpreted as special characters. If the format string is enclosed in double quotes, any quoted text inside the format string must be enclosed in single quotes and vice versa. To represent a quote or a backslash within quoted text, place a backslash before it.

If there is an unmatched quote, PowerHouse assumes there is a closing quote at the end of the format string.

Escape Character

The backslash (\) is the escape character. It is used to indicate that the character immediately following the backslash is not to be interpreted as a special character. For example:

```
    FORMATNUMBER(value, "\{###,##0\}")
returns "  {1,234}" if value is 1234
```

Within fill text, only the backslash is a special character. To represent a right brace or a backslash within fill text, put a backslash before it.

Leading and Trailing Signs

Leading or trailing signs to indicate positive or negative numbers are included as text. For example:

```
    FORMATNUMBER(quantity, "+###0", "-###0")
returns " +123" if quantity is 123
returns " -123" if quantity is -123

    FORMATNUMBER(balance, "$##,##0 CR", "$##,##0 DR")
returns " $1,234 CR" if balance is 1234
returns " $1,234 DR" if balance is -1234
```

```
    FORMATNUMBER(balance, " ##,##0 ", "(##,##0)"
returns "  1,234 " if balance is 1234
returns " (1,234)" if balance is -1234
```

You can suppress the negative indication if desired. For example:

```
    FORMATNUMBER(quantity, "##0", "##0")
returns "123" if quantity is -123
```

Zero Values

To represent a zero value with a blank or special text, include a format string for the value zero. For example:

```
    FORMATNUMBER(value, "###0", , " ")
returns " " if value is 0

    FORMATNUMBER(value, "###0", , " N/A")
returns " N/A" if value is 0
```

GETSYSTEMVAL (MPE/iX, UNIX, and Windows)

For GETSYSTEMVAL (**OpenVMS**), see (p. 413).

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	String	String	✔	✔	✔

Retrieves values defined at the operating system level.

Syntax

GETSYSTEMVAL(**string-expression**)

string-expression

Specifies the name of the system variable (**MPE/iX**) or environment variable (**UNIX, Windows**) to be retrieved.

Discussion

The DELETESYSTEMVAL, GETSYSTEMVAL, and SETSYSTEMVAL functions support the deletion, retrieval, and assignment of operating system values. This provides execution-time control of values through the definitions made at the operating system level and allows values to be passed between PowerHouse components.

GETSYSTEMVAL returns a string equal to the value of the environment variable. If the system value does not exist or cannot be retrieved for any reason, a string of zero-length is returned.

Example

For examples showing the use of the DELETESYSTEMVAL, GETSYSTEMVAL, and SETSYSTEMVAL functions, see (p. 394).

GETSYSTEMVAL (OpenVMS)

For GETSYSTEMVAL (**MPE/iX**, **UNIX**, and **Windows**), see (p. 412).

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	String	String	✔	✔	✔

Retrieves values defined at the operating system level.

Syntax

For OpenVMS logical names:

GETSYSTEMVAL(string-expression1
[,LOGICAL [,string-expression 2]])

For DCL symbols:

GETSYSTEMVAL(string-expression1, SYMBOL)

string-expression1

Specifies the name of the OpenVMS logical name or DCL symbol to be retrieved.

string-expression2

Optionally specifies the logical name table to be searched.

Default: The standard OpenVMS logical name table search order, set by LNM$FILE_DEV is used. Typically, the process table (LNM$PROCESS) is searched first, followed by the job table (LNM$JOB), the group table (LNM$GROUP), and finally the system table (LNM$SYSTEM).

LOGICAL

Specifies retrieval of an OpenVMS logical name. If the LOGICAL or SYMBOL keyword is not explicitly used, a logical name is assumed.

SYMBOL

Specifies retrieval of a DCL symbol. Local symbols are retrieved before global symbols.

Discussion

The DELETESYSTEMVAL, GETSYSTEMVAL, and SETSYSTEMVAL functions support the deletion, retrieval, and assignment of operating system values. This provides execution-time control of values through the definitions made at the operating system level and allows values to be passed between PowerHouse components.

GETSYSTEMVAL returns a string equal to the value of the symbol or logical. If the system value does not exist or cannot be retrieved for any reason, a string of zero-length is returned.

Examples

For examples showing the use of the DELETESYSTEMVAL, GETSYSTEMVAL, and SETSYSTEMVAL functions, see (p. 398).

INDEX

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	Numeric	✔	✔	✔

Gives the starting position of a substring.

Syntax

INDEX(string-expression1,string-expression2)

string-expression1

Specifies the string to be searched.

string-expression2

Specifies the shorter string to be found in string-expression1.

Discussion

String-expression1 must be longer than string-expression2 or else the result will always be 0.

If string-expression2 does not exist in string-expression1, the INDEX function returns a result of 0.

If string-expression2 starts in the first position of string-expression1, the INDEX function returns a 1.

Examples

```
Input: INDEX("ABCD","BC")
Result: 2

Input: INDEX ("ABC", "XY")
Result: 0

Input: > DEFINE ITEMX CHAR*10 = "ABCDEFGHIJ"
        > DEFINE ITEMY CHAR*1 = "H"
        > DEFINE ITEMZ = INDEX (ITEMX,ITEMY)
Result: 8
```

A new string may be defined that will contain all of the characters from the original string up to and including the first comma:

```
> DEFINE LOCOMMA NUM*2 = INDEX(LASTNAME, ",")
> DEFINE NEWITEM CHAR*20 = TRUNC(LASTNAME[1:LOCOMMA])
```

HEXDECODE Function

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔		

Converts a hexadecimal string to an ASCII string.

Syntax

HEXDECODE(string-expression)

string-expression

Specifies a string expression containing an even number of hexadecimal characters.

Discussion

The HEXDECODE function takes a string of hexadecimal characters and converts it to the equivalent ASCII string. Each pair of hexadecimal characters is converted to one ASCII character.

Examples

Encrypted strings can contain non-printing characters. Non-printing characters cannot be passed between PowerHouse Web pages. The HEXENCODE and HEXDECODE functions can be used to overcome this restriction. Use HEXENCODE to encode the encrypted string. Use HEXDECODE to decode the hexadecimal encoded string before decrypting it.

```
> TEMPORARY ENCODED_ID CHARACTER*20
> TEMPORARY INPUT_ID CHARACTER*10
> TEMPORARY OUTPUT_ID CHARACTER*10
> TEMPORARY T_KEY CHARACTER*10
...
> PROCEDURE DESIGNER ENC
>   BEGIN
>     LET ENCODED_ID = &
>       HEXENCODE(ENCRYPT(TRUNCATE(INPUT_ID),T_KEY))
>     END
> PROCEDURE DESIGNER DEC
>   BEGIN
>     LET OUTPUT_ID = DECRYPT(HEXDECODE(ENCODED_ID),T_KEY)
>     END
```

HEXENCODE Function

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔		

Converts an ASCII string to a hexadecimal string.

Syntax

HEXENCODE(string-expression)

string-expression

Specifies a string expression containing ASCII characters to be encoded.

Discussion

The HEXENCODE function takes a string of ASCII characters and converts it to the equivalent hexadecimal string. Each ASCII character is converted to two hexadecimal characters.

Examples

Encrypted strings can contain non-printing characters. Non-printing characters cannot be passed between PowerHouse Web pages. The HEXENCODE and HEXDECODE functions can be used to overcome this restriction. Use HEXENCODE to encode the encrypted string. Use HEXDECODE to decode the hexadecimal encoded string before decrypting it.

```
> TEMPORARY ENCODED_ID CHARACTER*20
> TEMPORARY INPUT_ID CHARACTER*10
> TEMPORARY OUTPUT_ID CHARACTER*10
> TEMPORARY T_KEY CHARACTER*10
...
> PROCEDURE DESIGNER ENC
>   BEGIN
>     LET ENCODED_ID = &
>       HEXENCODE(ENCRYPT(TRUNCATE(INPUT_ID),T_KEY))
>     END
> PROCEDURE DESIGNER DEC
>   BEGIN
>     LET OUTPUT_ID = DECRYPT(HEXDECODE(ENCODED_ID),T_KEY)
>     END
```

INTERVAL

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Date	Numeric	✔	✔	✔

Returns the days/time value of a string or number as a fractional value.

Syntax

INTERVAL(days-expression)

INTERVAL(days-expression,days-format)

days-expression

A days-expression is a numeric or string expression that yields a date in the form ddddd.hhmmssttt.

days-format

The default days-format is ddddd.hhmmssttt. The days-format can be an expression.

Both forms result in a numeric days/time value in the form ddddd.frac.

Example

In the following example, the INTERVAL function converts 10 days, 12 hours, and 15 minutes to 10.5097 days.

```
Input: INTERVAL( 10.1215, "dddd.hhmm")
Result: 10.5097
```

JCW (MPE/iX)

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	String	Numeric	✔	✔	✔

Returns an integer indicating the value of the specified JCW.

Syntax

JCW(string)

Discussion

The JCW function returns an unsigned integer indicating the value of the JCW specified by the string expression. The range of the integer is from 0 to 65535. If the specified JCW doesn't exist, the JCW function returns a value of -3.

Example

If you do not have the JCW QTP set, then the following returns a value of -3:

```
> DEFINE J-VALUE NUMERIC * 4 = JCW("QTP")
```

LASTDAY

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Date	Date	✔	✔	✔

Sets a date to the last day of the month.

Syntax

LASTDAY(date-expression)

date-expression

Specifies the input date. It can be a 6-digit date with the format YYMMDD or an 8-digit date with the format YYYYMMDD.

Example

```
Input: LASTDAY(900525)
Result: 90/05/31
```

To calculate the number of days left in the current month:

```
> DEFINE DAYSLEFT = LASTDAY(SYSDATE) - SYSDATE
```

LEFT JUSTIFY | LJ

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔	✔	✔

Left-justifies characters in a string.

Syntax

LEFT JUSTIFY(string-expression)

LJ(string-expression)

string-expression

Specifies the string to be left-justified.

Example

```
Input: LJ("   J SMITH")
Result: J SMITH
```

FIELDA and FIELDB both contain data with blanks at the beginning and end. The fields may be concatenated and the blanks removed. Left justifying eliminates the starting blanks, and truncating eliminates the trailing blanks.

```
> DEFINE ITEMX = TRUNC(LJ(FIELDA)) + TRUNC(LJ(FIELDB))
```

LINKVALUE

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Date, Numeric, String	✔	✔	✔

LINKVALUE returns the highest, lowest or equal value of a linkitem in the VIA option of an ACCESS statement in QDESIGN or the CHOOSE statement in QUIZ and QTP.

Syntax

LINKVALUE (linkitem[{,LOWEST}|{,HIGHEST}|{,EQUAL}])

Discussion

LINKVALUE is used by QDESIGN when generating SQL procedural code and referring to expressions or user input. It is also visible in the SQL displayed in QUIZ and QTP when SET LIST SQL is in effect.

Option	Returns
LOWEST	the lowest possible matching value of a USING expression when the expression ends with a generic character
HIGHEST	the highest possible matching value of a USING expression when the expression ends with a generic character
EQUAL	the exact value that was specified. If the expression ends with a generic character, the EQUAL option will return the lowest possible matching value.

Example

```
>   DECLARE PATIENT CURSOR FOR &
>   SELECT * FROM PATIENTS
>
>   ACCESS PATIENT
>   CHOOSE PATIENTID PARM &
>      PROMPT "Enter first patient ID: " &
>   RANGE TOPROMPT "Enter last patient ID: "
```

The following three queries are generated from the above cursor definition. At runtime, the correct query is used based on the values entered for the prompts.

```
>   SELECT * FROM PATIENTS
>   SELECT * FROM PATIENTS &
>      WHERE PATIENTID BETWEEN &
>      :LINKVALUE(PATIENTID,LOWEST) AND &
>      :LINKVALUE(PATIENTID,HIGHEST)
>   SELECT * FROM PATIENTS &
>      WHERE PATIENTID = &
>      :LINKVALUE(PATIENTID,EQUAL)
```

LOGONID

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔	✔	✔

MPE/iX:	Returns a string of up to 17 characters that contains the username and account name used by the current user to log onto the system.
OpenVMS:	Returns a string of up to 12 characters that contains the username used by the current user to log onto the system.
UNIX:	Returns a string of up to 8 characters that contains the logonid of the current user.
Windows:	Returns a string of up to 20 characters that contains the logonid of the current user.

Syntax

LOGONID

Discussion

In QDESIGN, LOGONID cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

Examples

MPE/iX

If the current user is logged on as USER1.ACCOUNT1, then the following returns USER1.ACCOUNT1

```
> DEFINE D-LOGONID CHARACTER*17 = LOGONID
```

OpenVMS

If the current user is logged on as USER1, the following DEFINE statement returns USER1:

```
> DEFINE DLOGONID CHARACTER*12 = LOGONID
```

UNIX

If the current user is logged on as USER1, the following DEFINE statement returns USER1:

```
>   DEFINE DLOGONID CHARACTER*8 = LOGONID
```

Windows

If the current user is logged on as USER1, the following DEFINE statement returns USER1:

```
>   DEFINE DLOGONID CHARACTER*20 = LOGONID
```

LOWER

Type	Input	Result	QDESIGN	QUIZ	QTP
SQL-DMF	String	String	✔	✔	✔

Downshifts a string expression.

Syntax

LOWER(string-expression)
Limit: Valid only in SQL.

string-expression

Specifies the string to be downshifted.

MATCHPATTERN

Type	Input	Result	QDESIGN	QUIZ	QTP
LF	String	Boolean	✔	✔	✔

Compares a string to a pattern.

Syntax

MATCHPATTERN(string-expression, pattern-string)

string-expression

Specifies the string to be checked.

pattern-string

Specifies the pattern to compare the string to.

Discussion

The MATCHPATTERN function returns a boolean TRUE value if the string expression exactly matches the specified pattern string. For information on pattern matching, see (p. 351).

Example

The matchpattern function is used to apply pattern matching procedurally, and is best used with other functions to remove leading and trailing spaces, as in

```
> IF NOT MATCHPATTERN &
>   (TRUNCATE(LJ(LASTNAME)), &
>   (TRUNCATE(TESTPATTERN))
>   THEN ERROR "PATTERN DID NOT MATCH."
```

The following example selects all records that do not meet standard North American postal code formats:

```
> SELECT IF NOT MATCHPATTERN &
>   (POSTAL-CODE,"(^#^#^#)|(^^^^^)")
```

MATCHUSER

Type	Input	Result	QDESIGN	QUIZ	QTP
LF	String	Boolean	✔	✔	✔

Determines whether or not a user belongs to a given application security class (ASC).

Syntax

MATCHUSER(asc)

asc

Specifies the application security class to be checked.

For more information about application security classes, see Chapter 2, "PDL Statements", in the *PDL and Utilities Reference*.

Discussion

The MATCHUSER function returns a Boolean TRUE value if the asc is an application security class associated with the current user as defined in the data dictionary.

Example

The following example returns the TRUE value if the current user belongs to the application security class CLERK, or FALSE if the current user belongs to an application security class other than CLERK:

```
>  DEFINE CLERKSTAT CHARACTER*20 = "TRUE" &
>     IF MATCHUSER("CLERK") ELSE "FALSE"
```

MISSING

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	NULL	✔	✔	✔

Returns NULL.

Syntax

MISSING

Discussion

The MISSING function is used to assign a NULL to an item or column. The MISSING function is identical to the NULL function. Also see "Testing for Null Values" (p. 348).

Examples

```
ITEM BRANCH FINAL MISSING
```

MOD

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Numeric	Numeric	✔	✔	✔

Returns a remainder after division.

Syntax

MOD(numeric-expression1,numeric-expression2)

numeric-expression1

Specifies the number to be divided.

numeric-expression2

Specifies the divisor.

Discussion

The MOD function returns the remainder when the numeric-expression1 is divided by numeric-expression2.

Example

```
Input: MOD(126,10)
Result: 6
```

For example, to calculate the day of the week

```
> DEFINE DAYOFWEEK NUM*5 = MOD(DAYS(19920101),7)
Result: 3
```

As the base date for the DAYS function is Sunday, December 31, 1899 (day 0), the resulting day of the week is a Wednesday.

NCONVERT

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	Numeric	✔	✔	✔

Converts a character string to a number.

Syntax

NCONVERT(string-expression)

string-expression

Specifies the input string. A negative sign can be included. Numbers to the right of a decimal point are ignored for variables defined as INTEGER, but used for rounding for variables defined as NUMERIC.

Examples

The following examples use a NUMERIC variable:

```
Input: NCONVERT("-12236")
Result: -12236

Input: NCONVERT("12.5")
Result: 13

Input: NCONVERT("-12.5")
Result: -13
```

A value for last year may be obtained by extracting the year from today's date and subtracting one:

```
> DEFINE CURRYR NUM*4 = &
>    NCONVERT(ASCII(SYSDATE)[1:4])
> DEFINE LASTYR NUM*4 = CURRYR - 1
```

NULL

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	NULL	✔	✔	✔

Returns NULL.

Syntax

NULL

Discussion

The NULL function is used to assign a NULL to an item or column. The NULL function is identical to the MISSING function. Also see "Testing for Null Values" (p. 348).

Examples

```
ITEM BRANCH FINAL NULL
```

OCCURRENCE

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Numeric	✔		

Returns the number of times a loop has been executed.

Syntax

OCCURRENCE [OF record-structure|item]

record-structure

The name of a record-structure defined in the dictionary or a table in a relational database.

item

A location where PowerHouse can store data. An item is a record item declared in the data dictionary, a defined item, a temporary item, a global temporary, or a predefined defined item. The general form of a record item is:

item[OF record-name]

Discussion

If there is no active loop, the value 1 is returned.

OCCURRENCE cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

Setting the OCCURRENCE System Function

The FOR control structure sets the system function, OCCURRENCE. This function controls which occurrence of repeating records, items, or clusters is addressed by other verbs. The current setting of OCCURRENCE can be addressed procedurally, although subscripting is not allowed. Outside the range of a FOR control structure, the value of OCCURRENCE is 1. Field procedures invoked by field verbs from within a FOR control structure are considered to be within the range of the FOR control structure, and the current setting of OCCURRENCE is unchanged.

A higher-level screen can invoke a lower-level screen by passing one occurrence of a file or item. The lower-level screen can have an independent FOR control structure. Although the indicated occurrence of the passed file or item is passed to lower-level screens, OCCURRENCE itself is not. The value of OCCURRENCE on the lower-level screen is always 1 unless a FOR loop is active there.

Only one FOR control structure can be active at one time; FOR loops nesting is not allowed within a procedure. However, when an INTERNAL procedure is performed from within a FOR loop, it can itself have a FOR loop.

The setting of occurrence in such situations is described by the following example:

```
> FILE A DESIGNER OCCURS 10
.
.
.
> PROCEDURE INTERNAL SHOWLOOP
>      BEGIN
```

```
>          LET X OF A = OCCURRENCE
>          FOR A
>              BEGIN
>                  LET X OF A = OCCURRENCE
>              END
>              LET X OF A = OCCURRENCE
>          END
> PROCEDURE ENTRY
>      BEGIN
.
.
.
>          FOR A
>              DO INTERNAL SHOWLOOP
.
.
.
>      END
```

In this example, the INTERNAL procedure, SHOWLOOP, is performed ten times. On the fifth time, the INTERNAL procedure sets the value of X to the following values:

```
5,1,2,3,4,5,6,7,8,9,10,5
```

At any time, there is only one setting of the function, OCCURRENCE.

You cannot address all the occurrences of a repeating item within a repeating file on the same screen. Instead, you must pass each occurrence of the file, in turn, to a subscreen and process the repeating item there.

OCTET_LENGTH

Type	Input	Result	QDESIGN	QUIZ	QTP
SQL-DMF	String	Numeric	✔	✔	✔

Returns the size of a string expression in bytes.

Syntax

OCTET_LENGTH(string-expression)
Limit: Valid only in SQL.

string-expression

Specifies the input string.

OLDVALUE

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Date, Numeric, String	✔		

Returns the existing value of the item during editing.

Syntax

OLDVALUE(item)

item

Refers to a record item (an item declared in the data dictionary), a predefined item, a temporary item, or a defined item. The general form of a record item in QUIZ, and QTP is:

item [(subscript)] [OF file]

The general form of a record item in QDESIGN is:

item [OF file]

Discussion

OLDVALUE cannot be

- used after the edit procedure has been completed
- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

Within an EDIT procedure, there are two values associated with the named field: the old (or existing) value in the record or temporary item buffer, and the new value entered by the QUICK user. The new value can be referenced in an expression using either the predefined items FIELDTEXT (for character data) or FIELDVALUE (for numeric and date data) or the item name. The old value of the item can be referenced in an expression by using the OLDVALUE function. The new value can be changed by using FIELDTEXT or FIELDVALUE (depending on the data type) as the target of a LET verb. Do not target the item directly in the EDIT procedure. For more information, see Chapter 7, "QDESIGN Procedures," in the *QDESIGN Reference* book.

Limit: OLDVALUE cannot be used after the EDIT procedure has been completed.

PACK

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔	✔	✔

Packs characters in a string.

Syntax

PACK(string-expression)

string-expression

Specifies the string to be packed.

Discussion

The PACK function eliminates leading spaces and all but one intervening space between words. It removes leading and trailing commas and semicolons, and removes leading spaces from words that begin with a comma, period, colon, or semicolon. The resulting size is unchanged.

Example

```
Input: PACK("J     SMITH")
Result: J SMITH
```

The following example will format MICHAEL SMITH to Smith, M:

```
> ACCESS EMPLOYEES
> DEFINE NAME CHAR*22 = &
>   PACK(LASTNAME[1:1] + &
>   DOWNSHIFT(LASTNAME[2:19]) + &
>   " , " + FIRSTNAME[1:1])
```

PORTID

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔	✔	✔

Identifies the terminal device.

Syntax

PORTID

Discussion

The PORTID function returns a string identifying the terminal device used in the current session. In a batch job, the PORTID function returns a string of length 0. On Windows, PORTID returns a blank.

In QDESIGN, PORTID cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

Example

The following returns the terminal device currently in use:

```
> DEFINE PRT CHARACTER*12 = PORTID
```

The results might be:

```
MPE/iX: 14
OpenVMS:LTA1
UNIX: \dev\ttyp4
```

POSITION

Type	Input	Result	QDESIGN	QUIZ	QTP
SQL-DMF	String	Numeric	✔	✔	✔

Gives the starting position of the first string in the second string.

Syntax

POSITION(string-expression1 IN string-expression2)

Limit: Valid only in SQL.

string-expression1

Specifies the string to be found in string-expression2.

string-expression2

string-expression2 must be longer than or equal to string-expression1 or the result will always be 0.

Example

The following example returns a value of 4:

```
POSITION('ing' IN 'string')
```

PROCESSLOCATION

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔		

Returns the value of the **procloc** program parameter.

Syntax

PROCESSLOCATION

Discussion

PROCESSLOCATION cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

Example

If QUICK is initiated as follows:

MPE/iX:	`QUICK PROCLOC=.DEVELOP.DOC`
OpenVMS:	`QUICK PROCLOC=path$disk:[dev.beta]`
UNIX:	`quick procloc=/develop`
Windows:	`QUICK PROCLOC=C:\DEVELOP`

The following DEFINE statement returns the process location specified in the **procloc** program parameter:

```
DEFINE D-PROCLOC CHARACTER*20 = PROCESSLOCATION
```

RANDOM

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	Numeric	Numeric	✔		✔

Returns a random number.

Syntax

RANDOM(seed)

seed

The seed must be a temporary item of datatype FLOAT SIZE 4 or INTEGER SIZE 4, or all resulting values will be zero. The value of the seed is modified by the function.

Discussion

The RANDOM function returns a random number of datatype FLOAT SIZE 8 greater than or equal to zero and less than one.

If the same seed is used, then the same random number is returned.

Example

This example uses the system time as a seed to generate a series of random numbers. If the same seed is used twice, the generated numbers are identical.

```
>  SCREEN RANDOM
>  FILE EMPLOY1
>  TEMP X FLOAT SIZE 4 INIT MOD(SYSTIME,10000)
>  DEFINE Y FLOAT SIZE 8 = RANDOM(X) * 1000000
>  SKIP 2
>  FIELD X LABEL "Seed" HELP &
>  "Enter a random seed or press Return to use &
>     System Time"
>  TITLE "Random Numbers:"
>  ALIGN (,,18)
>  CLUSTER OCCURS 10 AT 6,18
>  FIELD Y
```

RECORDLOCATION

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Numeric	✔		

Returns the physical record number of the current occurrence of a data record.

Syntax

RECORDLOCATION [OF record-structure]

Limit: This function is useful only with relative or direct files; the function result is meaningless with relational tables.

record-structure

The name of a record-structure defined in the dictionary or table in a relational database.

Discussion

The first data record is record number 0 (**MPE/iX, UNIX, Windows**) or 1 (**OpenVMS**). If the OF record-structure qualifier is omitted, the value is that of the assumed record-structure. If there are no records, RECORDLOCATION returns -1.

REMOVECENTURY

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Date	Date	✔	✔	✔

Converts an 8-digit date to a 6-digit date.

Syntax

REMOVECENTURY(date-expression)

date-expression

Specifies the input date.

Discussion

When you mix 6-digit and 8-digit dates in an expression or compare dissimilar date types in a condition, use either the REMOVECENTURY function or the ADDCENTURY function. For example, if the system default is century included, SYSDATE is an 8-digit date. You can use REMOVECENTURY to define a 6-digit version of this system date, as in

```
> DEFINE SHORTDATE DATE CENTURY EXCLUDED = REMOVECENTURY(SYSDATE)
```

or to make comparisons, as in

```
> SELECT IF LATEDATE = REMOVECENTURY(SYSDATE)
```

If a 6-digit date item is to derive its value from an 8-digit date, use the REMOVECENTURY function. For example, the following DEFINE statements assign the value 850630 to the 6-digit date SHORTDATE:

```
> DEFINE LONGDATE DATE CENTURY INCLUDED = 19850630
> DEFINE SHORTDATE DATE CENTURY EXCLUDED  = &
> REMOVECENTURY (LONGDATE)
```

Example

```
Input: REMOVECENTURY(19900525)
Result: 90/05/25
```

REVERSE

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔	✔	✔

Reverses the characters in a string.

Syntax

REVERSE(string-expression)

string-expression

Specifies the string to be reversed.

Discussion

Characters entered left to right appear right to left. Blanks appear to the left.

Example

```
Input: REVERSE("J Smith ")
Result: htimS J
```

RIGHT JUSTIFY | RJ

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔	✔	✔

Right-justifies characters in a string.

Syntax

RIGHT JUSTIFY(string-expression)

RJ(string-expression)

string-expression

Specifies the string to be right-justified.

Example

```
Input: RJ("J SMITH   ")
Result: ...J SMITH
> DEFINE ITEMX CHAR*20 = RJ(LASTNAME[1:20])
```

ROUND

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Numeric	Numeric	✔	✔	✔

Returns a rounded number.

Syntax

ROUND(numeric-expression1[,[numeric-expression2]
[,round-option]])

numeric-expression1

Specifies the number to be rounded.

numeric-expression2

Specifies the precision with which numeric-expression1 is rounded.

If numeric-expression2 is positive, it establishes the precision of numeric-expression1 by allowing you to declare the number of decimal places to keep in the result.

If numeric-expression2 is negative, numeric-expression1 is rounded by the specified factor of 10. For example,

- if -1 is entered, numeric-expression1 is rounded to the nearest 10
- if -2 is entered, numeric-expression1 is rounded to the nearest 100
- if -3 is entered, numeric-expression1 is rounded to the nearest 1000

Limit: If numeric-expression2 is not entered as an integer, PowerHouse truncates it to an integer.

Default: The default precision is zero (0). Numeric-expression1 is rounded to an integer.

round-option

You can specify the following options for the ROUND function:

Option	Rounds
NEAR	to the nearest number
UP	toward positive infinity
DOWN	toward negative infinity
ZERO	toward zero

Default: NEAR

Examples

```
Input: ROUND(4.5)
Result: 5
```

Using the Default Precision

If you use the round function without specifying a round-option or a precision, the default round-option, NEAR, is used with the default precision of zero. The results are as follows:

```
ROUND (4.4) = 4              ROUND (-4.4) = -4
ROUND (4.5) = 5              ROUND (-4.5) = -5
ROUND (4.6) = 5              ROUND (-4.6) = -5
```

If you select the up round-option without specifying a precision number, the default of zero is assumed. Note, however, that an extra comma is required in the syntax. The results are as follows:

```
ROUND (4.4, ,UP) = 5         ROUND (-4.4, ,UP) = -4
ROUND (4.5, ,UP) = 5         ROUND (-4.5, ,UP) = -4
ROUND (4.6, ,UP) = 5         ROUND (-4.6, ,UP) = -4
```

If you select the down round-option with the default precision of zero, the results are as follows:

```
ROUND (4.4, ,DOWN) = 4       ROUND (-4.4, ,DOWN) = -5
ROUND (4.5, ,DOWN) = 4       ROUND (-4.5, ,DOWN) = -5
ROUND (4.6, ,DOWN) = 4       ROUND (-4.6, ,DOWN) = -5
```

If you select the zero round-option with a default precision of zero, the results are as follows:

```
ROUND (4.4, ,ZERO) = 4       ROUND (-4.4, ,ZERO) = -4
ROUND (4.5, ,ZERO) = 4       ROUND (-4.5, ,ZERO) = -4
ROUND (4.6, ,ZERO) = 4       ROUND (-4.6, ,ZERO) = -4
```

Specifying the Precision

The following examples demonstrate the effect that specifying both a precision number (numeric-expression2) and a round-option has on the result of the ROUND function. This set of examples shows the results when numeric-expression2 is equal to

- 1 (meaning round to a precision of one decimal place)
- -3 (meaning round to the nearest thousand or 10^3)
- -2 (meaning round to the nearest hundred or 10^2)

If you select the near round-option, the results are as follows:

```
ROUND (123.44, 1, NEAR) = 123.4
ROUND (-123.4, 1, NEAR) = -123.4
ROUND (12334, -3, NEAR) = 12000
ROUND (-12334, -3, NEAR) = -12000
ROUND (12654, -3, NEAR) = 13000
ROUND (-12654, -3, NEAR) = -13000
ROUND (126.54, -2, NEAR) = 100
ROUND (-126.54, -2, NEAR) = -100
```

If you select the up round-option, the results are as follows:

```
ROUND (123.44, 1, UP) = 123.5
ROUND (-123.44, 1, UP) = -123.4
ROUND (12334, -3, UP) = 13000
ROUND (-12334, -3, UP) = -12000
ROUND (12654, -3, UP) = 13000
ROUND (-12654, -3, UP) = -12000
ROUND (126.54, -2,UP) = 200
ROUND (-126.54, -2, UP) = -100
```

If you select the down round-option, the results are as follows:

```
ROUND (123.44, 1, DOWN) = 123.4
ROUND (-123.44, 1, DOWN) = -123.5
```

```
ROUND (12334, -3, DOWN) = 12000
ROUND (-12334, -3, DOWN) = -13000
ROUND (12654, -3, DOWN) = 12000
ROUND (-12654, -3, DOWN) = -13000
ROUND (126.54, -2, DOWN) = 100
ROUND (-126.54, -2, DOWN) = -200
```

If you select the zero round-option, the results are as follows:

```
ROUND (123.44, 1, ZERO) = 123.4
ROUND (-123.44, 1, ZERO) = -123.4
ROUND (12334, -3, ZERO) = 12000
ROUND (-12334, -3, ZERO) = -12000
ROUND (12654, -3, ZERO) = 12000
ROUND (-12654, -3, ZERO) = -12000
ROUND (126.54, -2, ZERO) = 100
ROUND (-126.54, -2, ZERO) = -100
```

SCREENLEVEL

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Numeric	✔		

Returns the level number of the current screen.

Syntax

SCREENLEVEL

Discussion

Level 1 is the highest level screen. For a new thread, the value of SCREENLEVEL starts at 1. The number of screen levels allowed in an application is controlled by the QKGO execution-time parameters file. The maximum level is 15.

SCREENLEVEL cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

Example

The following DEFINE statement returns the current screen level:

```
>   DEFINE DLEVEL NUMERIC*2 = SCREENLEVEL
```

SETSYSTEMVAL (MPE/iX, UNIX, and Windows)

For SETSYSTEMVAL (**OpenVMS**), see (p. 449).

Type	Input	Result	QDESIGN	QUIZ	QTP
LF	String	Boolean	✔	✔	✔

Assigns operating system values.

Syntax

SETSYSTEMVAL (string-expression1, string-expression2)

string-expression1

Specifies the name of the system variable (**MPE/iX**) or environment variable (**UNIX, Windows**) to be assigned.

string-expression2

Specifies the value to be assigned to the system variable (**MPE/iX**) or environment variable (**UNIX, Windows**).

Discussion

The DELETESYSTEMVAL, GETSYSTEMVAL, and SETSYSTEMVAL functions support the deletion, retrieval, and assignment of operating system values. This provides execution-time control of values through the definitions made at the operating system level and allows values to be passed between PowerHouse components.

SETSYSTEMVAL returns a logical result of True if it successfully assigns the environment variable. Otherwise, it returns False. As this is a logical function, it must be used in a conditional expression.

At parse time, use of the SETSYSTEMVAL function will cause a syntax error if either of the following conditions exist:

- the **noaccess** program parameter is specified
- the OSACCESS resource file option equals OFF

Because the function accesses operating system values, its use is not permitted when operating system access is restricted.

There is no effect on the function in compiled screens, reports and runs if the **noaccess** program parameter or OSACCESS OFF resource file statement is used at runtime.

Example

The following QUICK example attempts to set a variable called "QTP_PARMS" to the value "RUN" plus the run number entered by the screen user. If the variable can't be set, an error message is displayed, otherwise the QTP run is started.

MPE/iX

```
TEMP RUN_REQ CHAR*1
.
.
.
> PROCEDURE DESIGNER 1
>  BEGIN
> ACCEPT RUN_REQ
> IF NOT SETSYSTEMVAL("QTP_PARMS", ("RUN" +  RUN_REQ ))
>      THEN ERROR "Could not set system variable"
>    ELSE RUN COMMAND "qtp auto=!QTP_PARMS"
```

```
.
.
.
```

UNIX, Windows

```
TEMP RUN_REQ CHAR*1
.
.
.
> PROCEDURE DESIGNER 1
>  BEGIN
> ACCEPT RUN_REQ
> IF NOT SETSYSTEMVAL("QTP_PARMS", ("RUN" +  RUN_REQ ))
>      THEN ERROR "Could not set environment variable"
>    ELSE RUN COMMAND "qtp auto=$QTP_PARMS"
.
.
.
```

For more examples showing the use of the DELETESYSTEMVAL, GETSYSTEMVAL, and SETSYSTEMVAL functions, see (p. 394).

SETSYSTEMVAL (OpenVMS)

For SETSYSTEMVAL (**MPE/iX**, **UNIX**, and **Windows**), see (p. 447).

Type	Input	Result	QDESIGN	QUIZ	QTP
LF	String	Boolean	✔	✔	✔

Assigns values at the operating-system level.

Syntax

For OpenVMS logical names:

SETSYSTEMVAL (string-expression1, string-expression2, [,LOGICAL[,string-expression3]])

For DCL global symbols:

SETSYSTEMVAL (string-expression1, string-expression2, SYMBOL)

string-expression1

Specifies the name of the OpenVMS logical name or DCL symbol to be assigned.

string-expression2

Specifies the value to be assigned to the system variable.

string-expression3

Optionally specifies the logical name table to be searched, such as LNM$JOB or LNM$GROUP. LNM$JOB should be used for logicals passed between processes.

Default: LNM$PROCESS

LOGICAL

Specifies that an OpenVMS logical name be defined. This is the default. If the keyword LOGICAL or SYMBOL is not explicitly used, a LOGICAL name is assumed.

SYMBOL

Specifies that a DCL global symbol be created.

Discussion

The DELETESYSTEMVAL, GETSYSTEMVAL, and SETSYSTEMVAL functions support the deletion, retrieval, and assignment of operating system values. This provides execution-time control of values through the definitions made at the operating system level and allows values to be passed between PowerHouse components.

SETSYSTEMVAL returns a logical result of True if it successfully assigns the logical name or global symbol. Otherwise, it returns False. As this is a logical function, it must be used in a conditional expression.

Example

The following QUICK example attempts to set a logical named QTP_PARMS to the value of "REPORT" plus the value of the REPORT_REQ item in the LNM$JOB logical name table. If the logical name can't be set, an error message is displayed.

```
.
.
.
```

```
> PROCEDURE DESIGNER 1
>  BEGIN
>      ACCEPT REPORT_REQ
>      IF NOT SETSYSTEMVAL ("QTP_PARMS",&
>          ("REPORT" + REPORT_REQ )&
>          , LOGICAL, "LNM$JOB")
>      THEN BEGIN
>        ERROR "Could not set logical name."
>      END
>      ELSE RUN COMMAND "QTP AUTO=QTP_PARMS"
.
.
.
```

For more examples showing the use of the DELETESYSTEMVAL, GETSYSTEMVAL, and SETSYSTEMVAL functions, see (p. 398).

SHIFTLEVEL

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Numeric	✔		

Returns the current shift level of the function keys.

Syntax

SHIFTLEVEL

Discussion

SHIFTLEVEL determines what the current function key shift level is. You can use this function to control an application's behavior based on shiftlevel, as shown in the following example.

Example

```
> SCREEN EMPLOYEES
> KEY 1 LEVEL 1 ACTION AND DATA SHIFT
> KEY 1 LEVEL 2 ACTION AND DATA SHIFT TO 1
> KEY 2 LEVEL 1 ACTION AND DATA HELP
> KEY 2 LEVEL 2 ACTION AND DATA EXTENDED HELP
> KEY 3 LEVEL 1 ACTION UPDATE
> KEY 3 LEVEL 2 ACTION UPDATE RETURN
.
.
> KEY 8 LEVEL 1 ACTION DESIGNER KEYHELP
> KEY 8 LEVEL 2 ACTION DESIGNER KEYHELP
.
.
> PROCEDURE DESIGNER KEYHELP
>  BEGIN
>     IF SHIFTLEVEL = 1
>        THEN
>           INFO = "F1=Shift to 2, F2=Help, F3=U, " + &
>           "... F8=Key Help" NOW RESPONSE
>        ELSE
>           INFO = "F1=Shift to 1, F2=Ext Help, F3=UR, " + &
>           "... F8=Key Help" NOW RESPONSE
>  END
.
.
.
```

SIGNONACCOUNT (MPE/iX)

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔	✔	✔

Returns a string of up to eight characters that contains the user's logon account.

Syntax

SIGNONACCOUNT

Example

If the current user logged on as USER1.ACCOUNT1, then SGNACCT contains the user's logon account, ACCOUNT1.

```
> DEFINE SGNACCT CHARACTER SIZE 8 = SIGNONACCOUNT
```

SIGNONGROUP (MPE/iX)

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔	✔	✔

Returns a string of up to eight characters that contains the user's logon group.

Syntax

SIGNONGROUP

Example

If the current user logged on as USER1.ACCOUNT1 in the group GRP1, then GR contains GRP1.

```
> DEFINE GR CHARACTER SIZE 8 = SIGNONGROUP
```

SIGNONUSER

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔	✔	✔

MPE/iX:	Returns a string of up to 8 characters that contains the user's username.
OpenVMS:	Returns a string of up to 12 characters that contains the username used by the current user to log onto the system.
UNIX:	Returns a string of up to 8 characters that contains the logonid of the current user.
Windows:	Returns a string of up to 20 characters that contains the logonid of the current user.

Syntax

SIGNONUSER

Discussion

In QDESIGN, SIGNONUSER cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

Example

MPE/iX

If the current user is logged on as USER1.ACCOUNT1, then D-USER contains USER1.

```
> DEFINE D-USER CHARACTER*8 = SIGNONUSER
```

OpenVMS

If the current user is logged on as USER1, the following DEFINE statement returns USER1:

```
> DEFINE DUSER CHARACTER*12 = SIGNONUSER
```

UNIX

If the current user is logged on as USER1, the following DEFINE statement returns USER1:

```
> DEFINE DUSER CHARACTER*8 = SIGNONUSER
```

Windows

If the current user is logged on as USER1, the following DEFINE statement returns USER1:

```
> DEFINE DUSER CHARACTER*20 = SIGNONUSER
```

SIZE

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	Numeric	✔	✔	✔

Returns the size of a string.

Syntax

SIZE(string-expression)

string-expression

Specifies the input string.

Discussion

If an item is used as the argument to SIZE, then the function returns the size of the item, not the size of the contents of the item. If a string expression is used, then the length of the result of the string expression is returned.

Example

When the string-expression is a character item, this function sometimes returns the length of the item rather than the length of string-expression's value, as demonstrated by the following example:

```
> DEFINE X CHARACTER*5 = "abc"

Input: SIZE("ABC  ")
Result: 5

Input: SIZE(X)
Result: 5

Input: SIZE("abc")
Result: 3

Input: SIZE(X + "de")
Result: 7

Input: SIZE(TRUNCATE(X))
Result: 3
```

The following example finds the length of last names:

```
> DEFINE ITEMX NUM*5 = SIZE(TRUNCATE(LASTNAME))
```

SOUNDEX

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String, Numeric	String	✔	✔	✔

Creates a phonetic code from a string.

Syntax

SOUNDEX(string-expression [,numeric-expression])

string-expression

Specifies the word to be coded.

numeric-expression

Controls the number of characters in the resulting code string. This parameter is optional.

Default: 4 characters

Discussion

The optional numeric expression entered after the string expression determines the size of the phonetic code string. The size of the number controls the number of names that the string matches: the larger the number, the fewer names are retrieved. The default of 4 is adequate for most searches, but increasing the number can be useful in some circumstances. For example, if most of the file names in your data dictionary begin with STOCK, searching for sound-alike files with a key of size 5, 6, or 7 will probably produce better results.

SOUNDEX Function Rules

The SOUNDEX function uses the following rules to generate soundex codes:

1. If adjacent letters are identical, only the first occurrence of the letter is kept.
2. The first character is always retained.
3. The vowels A, E, I, O, U, and Y and consonants W and H are dropped (except when they are the first character in the string).
4. All special characters, including spaces and apostrophes, are dropped unless they are the first character. If they are the first character, they are left as is in the final SOUNDEX code.
5. For each of the remaining letters, except the first, a numeric value is assigned, as follows:

Number assigned	For these letters
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R

6. If adjacent assigned numeric values are equal, then only the first occurrence is kept.

7. If there are insufficient letters to produce a result with the number of characters that are specified by the size parameter (numeric-expression), then the remainder is filled with zeros.

The rules and operation of the SOUNDEX function are unaffected by the CHARACTER SET option specified in the data dictionary. The rules work for most words or names that conform to English spelling conventions. However, the SOUNDEX function may not produce satisfactory results for data that contains many non-English words or names.

Using the SOUNDEX Function with Indexed Retrieval of Data Records

You can use the soundex function to generate alternate indexes for a record-structure. For example, if it's necessary to do phonetic inquiry on a name field, a soundex index field can be included in the record-structure design. The following QDESIGN statements maintain a soundex index field in the data records of the customers record-structure, and permit the retrieval of one or more customer names based on a phonetic match to a name entered by the user:

```
> SCREEN CUSTSCR
> FILE CUSTOMERS PRIMARY OCCURS 10
> ACCESS VIA CUSTINDEX USING SOUNDEX(CUSTNAME) &
>   REQUEST CUSTNAME
> ITEM CUSTINDEX FINAL SOUNDEX(CUSTNAME)
.
.
.
> TITLE "CUSTOMER NAME" AT 3,1
> SKIP TO LINE 5
> CLUSTER OCCURS WITH CUSTOMERS
> FIELD CUSTNAME
> CLUSTER
> BUILD
```

In the screen created by these statements, data records are retrieved by means of CUSTINDEX, using the soundex value of the item CUSTNAME. The ACCESS statement modifies the normal PATH and FIND procedures generated by QDESIGN, so that the QUICK-screen user is prompted for a value in the CUSTNAME field, and retrieval is by means of the soundex key. The FINAL option of the ITEM statement ensures that an appropriate value for the item CUSTINDEX is generated when a new customer is placed in the file and when a customer name is changed.

Example

```
Input: SOUNDEX("BURTON")
Result: B635
```

This example results in finding names such as Bertin, Burtin, and Burdon.

SPREAD

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔	✔	✔

Adds an additional space between each character of a string expression.

Syntax

SPREAD(string-expression)

string-expression

Specifies the input string.

Example

```
Input: SPREAD("TEST")
Result: T E S T
Input: > DEFINE ITEMA CHAR*25 = "FUTURE'S INVENTORY"
       > DEFINE ITEMX CHAR*60 = SPREAD(ITEMA) + "!"
Result: F U T U R E ' S   I N V E N T O R Y !
```

SQLCODE

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Numeric	✔		

Returns the status code of the last SQL statement.

Syntax

SQLCODE

Discussion

An SQLCODE of 250 or above indicates an informational and warning code. The following table contains some of the more common informational and warning codes and messages that users may encounter while using SQL.

SQLCODE	SQLMESSAGE
262	'^1' not supported; raised to '^2'.
263	The operation is pending.
265	The uniqueness of the dbkey value cannot be guaranteed.
267	The blob data has been truncated to the size specified at creation.
268	The target database has performed truncation on data.
269	The target database has returned information concerning the '^1' operation.
270	An attempt to cancel the '^1' operation has been rejected by the target database.
271	The '^1' database does not support transactions.

An SQLCODE less than 0 indicates an error. The following table contains some of the more common error codes and messages that users may encounter while using SQL.

SQLCODE	SQLMESSAGE
-3	Corrupt or obsolete metadata has been encountered during operation '^1'.
-4	A connection has been rejected by the remote interface during operation '^1'.
-5	An internal system failure has occurred during operation '^1'.
-6	An illegal data conversion has been attempted during operation ' ^1'.
-7	Corrupt database data structures have been encountered during operation '^1'.
-8	A lock conflict with another process has been encountered during operation '^1'.
-9	A general exception has occurred during operation '^1'.
-10	An internal implementation limit has been exceeded during operation '^1'.

SQLCODE	SQLMESSAGE
-11	An invalid database has been encountered during operation '^1'.
-12	A low-level input/output error has occurred during operation '^1'.
-14	A non-recoverable transport layer error has occurred during operation '^1'.
-15	Memory allocation failure.
-16	An error was detected during processing of the SQL request.
-17	The server on '^1' for user '^2' has timed out.
-18	An operation was attempted without appropriate permissions during operation '^1'.
-19	The operating system returned an error during operation '^1'.
-27	Feature '^1' in function '^2' is not supported.
-26	Function '^1' is not supported.
-39	Invalid database name: ^1.
-40	The size of the ^1 record (^2) exceeds the maximum supported size (^3).
-41	Null parameter values are not supported.
-45	The database type '^1' associated with the database name '^2' is not supported.
-49	It was not possible to access the '^1' database.
-50	It was not possible to establish an arithmetic trap.
-51	An arithmetic exception was detected.
-53	Unable to load the '^1' gateway.
-60	The operation '^1' has been cancelled since you did not provide the requested security information.
-69	The underlying database detected an error during processing of the SQL request.
-71	A CALL request can only be executed in a read-write transaction.
-72	The database specific native language is not supported.
-73	Insert/Update/Delete requests are not permitted in read-only transactions.
-74	The current transaction has been rolled back by the underlying database during operation '^1'.

SQLMESSAGE

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔		

Returns the error message that explains the status code of the last SQL statement executed.

Syntax

SQLMESSAGE

Discussion

For a detailed description of the status codes and their descriptions, see (p. 459).

Example

```
>   SQL CALL STORENAME_PROC &
>       IN ASSETSDB(STORENAME) &
>       RETURNING STORECOUNT &
>       ON ERROR CONTINUE
.
.
.
>   IF SQLOK
.
.
.
>   ELSE
>       BEGIN
>           LET ERRORMESSAGE = SQLMESSAGE
>           IF SQLCODE EQ ... &
>               THEN INFO=ERRORMESSAGE NOW
```

SUBSTITUTE

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String, Numeric	String	✔	✔	✔

Replaces substitution characters (if any) in a string or in a message retrieved from a designer message file.

Syntax

SUBSTITUTE (string-expression1|numeric-expression [,string-expression2]...)

Limit: Total length of a string after substitution cannot exceed 132 characters.

string-expression1

A string expression containing substitution characters. The default substitution character is a caret (^).

numeric-expression

A number corresponding to a message in a designer message file.

string-expression2

String expression(s) to replace a substitution character in string-expression1 or the designer message.

Discussion

The first parameter to this function is a designer message. The designer message is either determined from a string expression or from the record number in a designer message file. The designer message may contain one or more substitution characters.

Using SUBSTITUTE and string-expression1

The first substitution character in string-expression1 is replaced by string-expression2. Similarly, the second substitution character in string-expression1 is replaced by string-expression3, and so on. A substitution character is not replaced if you do not supply a corresponding string expression.

Using SUBSTITUTE with Designer Messages and numeric-expression1

The SUBSTITUTE function is typically used with designer messages where numeric-expression1 identifies a message in the designer message file.

The following is the processing order:

1. The designer message file is opened if it has not been already.
2. The corresponding message is retrieved from the designer message file. Message number 1 corresponds to the first record in the designer message file since there is no message number zero.
3. A standard error message appears if the designer message file cannot be opened, or if the message record cannot be retrieved. The message appears as "Designer Message Number n.", where n is the designer message number.
4. If the message is blank then it is displayed as a blank message.

5. The first substitution character in the message is replaced by string-expression2. Similarly the second substitution character in the message is replaced by string-expression3, and so on. A substitution character is not replaced if you do not supply a corresponding string expression.

The name of the designer message file cannot be changed while the products are running. Therefore, if you intend to use a designer message file, you must point the component to the appropriate file or files before you start it. For more information about designer message files, see (p. 270).

When creating designer messages, you may want to use the text order number feature. This allows you to specify the order in which substitution strings replace substitution characters in a message by appending an order number to the substitution characters in a string. For more information, see (p. 271).

Example

In the following QUIZ report, the SUBSTITUTE function is used to define the item, NOTIFY_EMPLOYEE. In this case, the item values for FIRST_INITIAL and LASTNAME are substituted into the string "NOTIFY ^. ^, BILLINGS ARE LESS THAN $300." This string either exists in the syntax (string-expression1) or in a message file (numeric-expression). The resulting report lists employees who should be notified that their billings for a specific month and project are too low.

Using string-expression1:

```
> SET REPORT DEVICE PRINTER
> ACCESS EMPLOYEES LINK TO BILLINGS
> DEFINE FIRSTINITIAL CHAR*1 = FIRSTNAME [1:1]
> DEFINE NOTIFYEMPLOYEE CHAR*50 = SUBSTITUTE &
>   ("NOTIFY ^. ^, BILLINGS ARE LESS THAN $300", &
>   FIRSTINITIAL, LASTNAME) IF BILLING LT 30000
> REPORT PROJECT MONTH BILLING NOTIFYEMPLOYEE
> GO
```

Using numeric-expression where message number 14 contains "NOTIFY ^. ^, BILLINGS ARE LESS THAN $300":

```
> SET REPORT DEVICE PRINTER
> ACCESS EMPLOYEES LINK TO BILLINGS
> DEFINE FIRSTINITIAL CHAR*1 = FIRSTNAME [1:1]
> DEFINE NOTIFYEMPLOYEE CHAR*50 = SUBSTITUTE &
>   (14, FIRSTINITIAL, LASTNAME) IF BILLING LT 30000
> REPORT PROJECT MONTH BILLING NOTIFYEMPLOYEE
> GO
```

Both methods produce the same result:

PROJECT	MONTH	BILLING	NOTIFY EMPLOYEE
P000001	01/93	100.00	NOTIFY M. SMITH, BILLINGS ARE LESS THAN $300
P000001	04/93	420.00	
P000003	03/93	315.00	
P000006	06/93	120.00	NOTIFY R. HALLADAY, BILLINGS ARE LESS THAN $300
P000006	03/93	152.00	NOTIFY R. HALLADAY, BILLINGS ARE LESS THAN $300
P000002	05/93	470.00	
P000001	02/93	945.00	

SUBSTRING

Type	Input	Result	QDESIGN	QUIZ	QTP
SQL-DMF	String, Numeric	String	✔	✔	✔

Extracts a portion of a string expression.

Syntax

SUBSTRING(string-expression FROM start [FOR length])
Limit: Valid only in SQL.

string-expression

Specifies the string from which the substring is to be extracted.

start

A numeric expression specifying the starting position of the extract. The first position is 1.

length

A numeric expression specifying the length of the extract.
Default: 1

Substring Extract

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String, Numeric	String	✔	✔	✔

Extracts a substring from a string.

Syntax

string-expression[start:length]

The square brackets are required syntax in this function.

string-expression

Specifies the string from which the substring is to be extracted.

start

A numeric expression specifying the starting position of the extract. The first position is 1.

length

A numeric expression specifying the length of the extract.

Discussion

The Substring Extract function can be used on numeric or date values if these values are first converted to character values. The square brackets are required syntax in this function.

Byte Ordering

Take byte ordering into account when extracting values from numeric or date values. Byte ordering depends on whether the computer is little endian or big endian. For example, this is a common technique used to put the escape character into a single character byte:

```
> DEFINE ESC INTEGER SIZE 2 = 27
> DEFINE ESC_CHAR CHARACTER SIZE 1 = CHARACTERS(ESC)[2:1]
```

The substring extract takes the value from the second byte. This works as expected on a big endian computer. However, on a little endian computer, the byte order is reversed and the substring extract must be:

```
> DEFINE ESC_CHAR CHARACTER SIZE 1 = CHARACTERS(ESC)[1:1]
```

MPE/iX, HP-UX, AIX, and Solaris are big endian platforms. OpenVMS and Windows are little endian platforms.

Examples

```
Input: "ABCD"[2:2]
Result: BC
```

This example extracts the month portion of an 8-digit date:

```
> DEFINE MM CHARACTER*2=(ASCII(SYSDATE))[5:2]
```

SUM

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	Numeric	Numeric		✔	✔

Sums items in an array.

Syntax

SUM(array[,numeric-expression1,numeric-expression2 [,numeric-expression3]])

array

Specifies the array to be summed.

numeric-expression1,numeric-expression2

Used to specify a subset of the occurrences of an array. Numeric-expression1 specifies the beginning subscript; numeric-expression2, the ending subscript. The first occurrence in an array is 1.

numeric-expression3

Controls the increment for the subset of occurrences used in the summing process.

Discussion

The SUM function sums the individual items in an array (a repeating item) from numeric-expression1 to numeric-expression2. The increment is numeric-expression3.

It sums all non-null values in a series of temporary items and returns the value null if all the temporary items used in the calculation are null.

Assumptions Made by the SUM Function

The sum function makes the following assumptions if you specify inconsistent information:

- The parameters numeric-expression1, numeric-expression2 and numeric-expression3 are floored (changed to the largest integer not larger than the value). For example, if numeric-expression1 is 3.4, the value 3 is used. Likewise, if numeric-expression3 is -1.5, the value -2 is used.
- If numeric-expression3 is zero, the result is zero.
- If numeric-expression1, numeric-expression2, and numeric-expression3 define a null range (such as sum(amount,12,1) or sum(amount,1,12,-1)), a calculation error occurs. If the on calculation errors report option is specified in the request statement, the result is zero.
- An out-of-range subscript causes a calculation error. If the on calculation errors report option is specified in the request statement, the result is zero.

Examples

Summing Entire Arrays

The sum function eliminates the need to reference individual array occurrences when you want to add them together. For example, the statement

```
> DEFINE TOTALAMOUNT = SUM(AMOUNT)
```

stores the sum of all occurrences of the array amount in the item totalamount. If the array amount has 12 occurrences, then the item sum(amount) is equivalent to

```
AMOUNT(1) + AMOUNT(2) + AMOUNT(3) + . . . + AMOUNT(12)
```

Summing Subsets of Array Occurrences

If you want to sum a subset of the occurrences of an array, you can use numeric-expression1 and numeric-expression2 to specify the beginning and ending subscripts in the sum function, as in

```
> DEFINE CURRENTMONTH NUMERIC = PARM
> EDIT CURRENTMONTH VALUES 1 TO 12
> DEFINE YTDTOTAL = SUM(AMOUNT,1,CURRENTMONTH)
```

Here, the SUM function performs a year-to-date sum of the occurrences in the array, AMOUNT.

You can use the optional beginning and ending subscripts to overcome such things as inconsistencies between calendar years and fiscal years. For example, if your fiscal year starts in June instead of January and the array amount is still organized with January as the item amount(1), you might require an algorithm that calculates a year-to-date total. Assuming the item, currentmonth, is still the current month, you can define a fiscal total as

```
> DEFINE FISCAL = RUN(AMOUNT,6,CURRENTMONTH) &
>   IF CURRENTMONTH >= 6 &
>      ELSE SUM(AMOUNT,6,12) + &
>      SUM(AMOUNT,1,CURRENTMONTH)
```

Summing Intermittent Occurrences of an Array

The final optional parameter for the sum function (numeric-expression3) specifies an index increment for the summing operation. If you want to sum only the odd occurrences of the array amount, use numeric-expression3, as in

```
> DEFINE ODDAMOUNT = SUM(AMOUNT,1,11,2)
```

This example sums the values of every second occurrence of the array amount. The item sum(amount,1,11,2) is equivalent to

```
AMOUNT(1)+AMOUNT(3)+AMOUNT(5)+AMOUNT(7)+AMOUNT(9)+AMOUNT(11)
```

For more information about arrays and subscripting see (p. 284).

SYSDATE

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Date	✔	✔	✔

Returns the current system date.

Syntax

SYSDATE

Discussion

The SYSDATE function returns the current system date in the format YYMMDD (if dates are defined in the dictionary as not including a century prefix) or in the format YYYYMMDD (if dates include a century prefix).

In QDESIGN, SYSDATE cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

QTP gets the system date every time this function is executed (as opposed to QUIZ which gets it once at the start of the report).

Example

If dates are defined in the dictionary as century excluded, the following DEFINE statement returns the value 97/11/24 if it is November 24, 1997:

```
> DEFINE DSYSDATE DATE = SYSDATE
```

If dates are defined as century included, the same DEFINE statement returns 1997/11/24.

SYSDATETIME

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	D	✔	✔	✔

Returns the current system date and time.

Syntax

SYSDATETIME

Description

The SYSDATETIME function returns the current system date and time in the DATETIME format. The general form of SYSDATETIME is:

yyyymmdd.hhmmssth

t represents tenths and h represents hundreths of a second. Hundredths may not be supported on all platforms because of FLOAT limitations.

In QDESIGN, SYSDATETIME cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

QTP gets the system date every time this function is executed (as opposed to QUIZ which gets it once at the start of the report).

SYSNAME

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔	✔	✔

Returns the system title specified in the dictionary.

Syntax

SYSNAME

Discussion

The SYSNAME function returns the dictionary title specified in the data dictionary as a 40-character string.

In QDESIGN, SYSNAME cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

Example

Programmers can use the SYSNAME function to retrieve a standard title to be used on all screens. The application title retrieved by the following DEFINE statement is "Future Industries Staff System":

```
> DEFINE DSYSNAME CHARACTER*40 = SYSNAME
```

SYSPAGE

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Numeric		✔	

Returns the current page number of a report.

Syntax

SYSPAGE

Discussion

The SYSPAGE function is only valid as a report-item within statements that use a report-group. The value returned is ten digits long with leading zeros suppressed. The maximum value is 2,147,483,647.

SYSTIME

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Numeric	✔	✔	✔

Returns the current system time.

Syntax

SYSTIME

Discussion

In QDESIGN and QTP, the SYSTIME function returns the system time as a number in the form HHMMSSTH (for example, 11080008).

In QUIZ, the SYSTIME function returns the system time as a number in the form HHMM (for example, 1108).

QTP gets the system date every time this function is executed (as opposed to QUIZ which gets it once at the start of the report).

In QDESIGN, SYSTIME cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

Limit (MPE/iX): Since the MPE/iX internal clock does not return hundredths of a second, the last digit will be 0 (zero) if it is displayed or used.

Example

The following example returns the current system time:

```
> DEFINE SYSTEMTIME NUMERIC*8 = SYSTIME
```

TERMTYPE

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔		

Returns a string expression containing the terminal characteristics specified in either the program parameter or the terminal prompt.

Syntax

TERMTYPE

Limit: The string can be up to 35 characters in length. For example,

```
HP2392-48 or vt100
```

Discussion

TERMTYPE cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

TRUNCATE

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔	✔	✔

Removes trailing blanks from a string.

Syntax

TRUNCATE(string-expression)

string-expression

Specifies the string from which trailing blanks are to be removed. The length of the resulting string is reduced by the number of blanks removed.

Discussion

The truncate function can be used to facilitate pattern matching. Removing trailing blanks from a string makes pattern matching less error-prone, since trailing blanks can often prevent strings from matching specified patterns.

```
> DEFINE TESTPATTERN CHARACTER*8=PARM &
>    PROMPT="ENTER SURNAME SELECTION PATTERN:"
> SELECT IF MATCHPATTERN (LASTNAME, &
>    TRUNCATE(TESTPATTERN))
```

It can also be used when concatenating strings to create a formatted result:

```
> DEFINE FULLNAME CHARACTER*60 = &
>    TRUNCATE(LASTNAME) + ", " + &
>    TRUNCATE(FIRSTNAME)
```

If the TRUNCATE function is not used with MATCHPATTERN, and "A@" is entered at the prompt (indicating all last names beginning with the letter A are selected), then the defined item, TEST-PATTERN, contains the letter A, followed by the ampersand metacharacter (@), which is in turn followed by six blanks, "A@ ". With this pattern, only last names beginning with the letter A and ending in six blanks are selected.

Removing trailing blanks from the defined item, TEST-PATTERN, ensures that your SELECT statement selects all names beginning with the letter A, as intended.

Example

```
Input: TRUNCATE("text    ")
Result: text
```

UIC (OpenVMS, UNIX)

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔	✔	✔

OpenVMS:	Returns a user identification code of the current user by group-ID and member-ID.
UNIX:	Returns a string that identifies the current user by group-ID and user-ID.

Syntax

UIC

Discussion

In QDESIGN, UIC cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

OpenVMS

The UIC function returns the numeric user identification code as a 14-character string in the form

```
[GGGGG,MMMMMM]
```

where GGGGG represents the group-ID number and MMMMMM represents the member-ID. By default, the brackets are included in the result.For compatibility with UNIX and Windows, you can use the **nouicbrackets** program parameter to obtain results without brackets

UNIX

The results of the function is a string of up to 11 characters in the form

```
gid,uid
```

where gid represents group-ID number and uid represents the user-ID used by the current user to log on to the system.

UPPER

Type	Input	Result	QDESIGN	QUIZ	QTP
SQL-DMF	String	String	✔	✔	✔

Upshifts a string expression.

Syntax

UPPER(string-expression)
Limit: Valid only in SQL.

string-expression

Specifies the string to be upshifted.

UPSHIFT

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔	✔	✔

Shifts characters to uppercase.

Syntax

UPSHIFT(string-expression)

string-expression

Specifies the string to be converted to uppercase.

Discussion

The function uses the upshift/downshift tables defined in the data dictionary to determine the appropriate shift characters.

Example

```
Input: UPSHIFT("text")
Result: TEXT
```

The following example will display the employee names with the first letter of the firstname in uppercase, the first letter of the lastname in uppercase, and the remaining letters of the lastname in lowercase (as in M Smith)

```
> ACCESS EMPLOYEES
> DEFINE NAME CHAR*30 = &
>   UPSHIFT(FIRSTNAME[1:1]) + " " +&
>   UPSHIFT(LASTNAME[1:1]) + &
>   DOWNSHIFT(LASTNAME[2:19])
```

VALIDPATTERN

Type	Input	Result	QDESIGN	QUIZ	QTP
LF	String	Boolean	✔	✔	✔

Checks a pattern string.

Syntax

VALIDPATTERN(string-expression)

string-expression

Specifies the string to be checked.

Discussion

The VALIDPATTERN function returns a boolean value, TRUE, if the pattern specified by the string expression is valid.

Example

Before using the MATCHPATTERN function, you can use VALIDPATTERN to determine whether a user-entered pattern or a calculated pattern is syntactically valid, as in

```
> DEFINE USERPATTERN char*10 = PARM
> DEFINE FINALPATTERN = USERPATTERN &
>   IF VALIDPATTERN(USERPATTERN)
```

or

```
> IF NOT VALIDPATTERN(USER-PATTERN)
> THEN ERROR "Invalid Pattern"
```

Invalid patterns occur, for example, when pattern-matching metacharacters clash or when reserved metacharacters are used without the escape metacharacter.

For more information about patterns and pattern matching, see (p. 351).

VMSTIMESTAMP (OpenVMS)

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	Date	✔	✔	✔

Returns the system date and time in OpenVMS binary-time format.

Syntax

VMSTIMESTAMP

Discussion

In QDESIGN, VMSTIMESTAMP cannot be

- passed to other screens
- represented as input fields
- represented as display fields
- passed to external subroutines
- changed by a LET verb

WEBLOGONID

Type	Input	Result	QDESIGN	QUIZ	QTP
SF	System generated	String	✔		

Returns the authenticated username.

Syntax

WEBLOGONID

Discussion

WEBLOGONID returns the authenticated username used by the user in response to an authentication prompt. If Web authentication is not used, WEBLOGONID is blank.

Examples

If the current user entered User1 to the authentication prompt, then the following returns User1:

```
> DEFINE D-WEBLOGONID CHARACTER*20 = WEBLOGONID
```

ZEROFILL

Type	Input	Result	QDESIGN	QUIZ	QTP
DMF	String	String	✔	✔	✔

Replaces leading spaces with zeros.

Syntax

ZEROFILL(string-expression)

string-expression

Specifies the string to be filled with zeros.

Example

```
Input: ZEROFILL("  562")
Result: 00562
```

The following example shows how an integer size 4 field may be changed to a field with leading zeros. For example, change 1234 to 01234

```
> DEFINE INITITEM INT*4 = 1234
> DEFINE ITEMX CHAR*5 = ASCII(INTITEM)
> DEFINE ITEMY CHAR*5 = ZEROFILL(RJ(ITEMX))
```

Glossary

This table compares PowerHouse, SQL, and Relational Entities.

PowerHouse Entity	SQL Entity	Relational Entity
file	schema	database
record-structure	tablelview	relationlview
element	domainlabstract datatype	global field
item	column	local field
index	index	index
record	row/tuple	row
segment	index column	index field

access list Refers to all record-structures declared in an ACCESS statement in QUIZ or QTP.

Action bar An alternative to the traditional Action field entry mechanisms.

Action field A special field in the top left-hand corner of every PowerHouse screen created with default user interface options. Commands entered here control what the QUICK-screen user can do on the screen.

alias An alternative name that can be given to a file or item.

allowed syntax The syntax PowerHouse allows you to enter.

alternate index An index on a file defined in the PowerHouse dictionary that's associated with the physical file. It contains the same fields, in the same order, as the physical file.

application A set of programs designed to solve a specific problem or meet a specific need.

application line The general term that specifies the location of your application on the terminal screen.

application manager A dictionary user who has access to the PHD Screen System through the Application Management Menu. This menu provides options used to supervise application development and maintain existing applications.

application security — Specifies which PowerHouse users have access (through PowerHouse) to the data used in the application. Application security doesn't override the operating system's security. (See also security)

application security class — A group of PowerHouse users who share the same access (through PowerHouse) to the data used in the application.

argument — See general term.

array — An item that's defined with multiple occurrences in the data dictionary. Also called a repeating item. For example, if the array item month as 12 occurrences, one for each month, then it's possible to identify any one of the occurrences by using a subscript with the array item name, as in month(6). QDESIGN doesn't allow subscripting. (See also occurrence and subscript.)

ASCII — American Standard Code for Information Interchange. Determines the character set and default sorting sequence used in PowerHouse. Also the name of a PowerHouse data manipulation function.

attach — Connects PowerHouse to a relational database.

attribute — A characteristic (such as size) that you assign to a particular entity.

automatic definition — PHD Screen System process whereby entities are automatically defined as related entities are defined. For example, as a record-structure is defined, an automatic definition is generated for its file if the file has not yet been defined.

break-item — An item named in the SORT or SORTED statement that declares when a control break occurs; it tells PowerHouse which item you want to use to organize your report.

buffer — An area of computer memory used to temporarily store information.

character — Any letter, number, or special character supported by your computer system (that is, any valid ASCII character).

checksum — A mechanism to detect data corruption. An algorithm that returns a value (checksum) for the input string. This value is held with the data, and when the data is later read from storage, the checksum is recomputed from the data. If the checksum has changed, the data has changed.

class — See application security class.

cluster — A group of fields that repeats on one screen.

coded record-structure — A record-structure identified by a specific value of an item. The value is declared using the SELECT option of the ITEM statement in PDL.

column — The smallest data entity that you can reference (also referred to as an item or field).

commit transaction — Saves any changes made by the transaction permanently to the database, and ends the transaction.

compile In PowerHouse, the actions required to transform a source statement file into the compiled file containing the tables that control processing.

compiled file An executable PowerHouse file.

component Refers to one of the PowerHouse programs: PDL, PHDPDL, PHD Screen System, QDESIGN, QUICK, QUIZ, and QTP.

conceptual transaction One or more PowerHouse transactions which span QUICK screen boundaries that the screen designer views as a related group of operations. The screen designer must maintain the integrity of these transactions.

concurrency A measure of the degree to which simultaneous transactions can operate in a database without being involved in excessive (unnatural) waiting caused by other transactions. Generally, strategies employed by databases to increase concurrency result in reduced consistency.

concurrency control The mechanism used by a database to support concurrent database transactions as well as to protect data integrity. Generally, a database preserves data integrity by controlling concurrent access to data, and by detecting and resolving conflicts between transactions. (See also optimistic locking and pessimistic locking.)

condition A logical test that must be satisfied in order for some action to be performed.

conditional expression Contains a series of expressions and conditions that are evaluated in order until a condition is met.

consistency A measure of the degree to which each database transaction appears to be isolated from the effects of other simultaneous transactions. The term "isolation" is also used. Generally, strategies employed by databases to increase consistency result in reduced concurrency.

control break A division that indicates a change in the value of a break-item.

control file A file used in file maintenance continuing the maintenance actions being performed on a group of files.

[Ctrl-Y] (MPE/iX)
[Ctrl-C] (UNIX,OpenVMS) A user break initiated by pressing the control key and the indicated letter key at the same time. It sends a signal that interrupts processing and returns control to the program.

control group The group of record complexes, sharing the same break-item value, that are processed between consecutive control breaks. (See also control break.)

cursor A cursor is the name of a set of data declared on a DECLARE CURSOR statement.

database Refers to the name of a relational database.

database transaction Also known as a physical transaction, it is a unit of work known to the relational database management system. A database transaction can be used to access one or more different databases and, indirectly through gateways, different database types. The relational database management system maintains the integrity of these transactions.

data definition The characteristics of an application's data, which is stored in the data dictionary. The data definitions determine how PowerHouse accesses data for the application.

data dictionary A storehouse of information about the data that you use in your applications.

data file The characteristics of an application's data as stored in the data dictionary. The data definition determines how the data is accessed and formatted by the application.

data item See item.

data manipulation function Used to manipulate and test item values or return values. (See also system function.)

data record See record.

datatype The way an item is stored in a record.

date expression See expression.

deadlock The stalemate that can occur when more than one program attempts to lock two or more files at the same time.

default An automatic response built into a program to ensure that appropriate actions are performed or that acceptable values are provided. Defaults can be overridden by the user if desired.

default-location Refers to a device or directory specification.

defined item An item that you create and use to combine and manipulate information. Although these items are used and referenced in the same way as all the other items in your data dictionary, they exist only for the duration of your screen, report, or run. You can use them as report-items and they can be referenced by other statements for calculations. (See also item.)

definition See data definition.

designated file A file opened by a PowerHouse component and its utilities for their internal use.

detail information The information typically contained in one detail line - usually from a single record or record complex. Detail information is the opposite of summary information. (See also summary information.)

detail lines The lines that constitute the body of a report, as distinguished from heading lines and footing lines. Usually each line corresponds to a single record or to a single record complex.

detail report In QSHOW, produces a listing of all the details about all the elements in the data dictionary.

dictionary See data dictionary.

dictionary definition The data definition, security specification, system-wide standards, and other information stored in a data dictionary.

domain Can be shared by columns of various tables (also referred to as a global field or element).

dictionary manager A dictionary user who has access to the PHD Screen System through the Dictionary Management Menu. A dictionary manager has access to all the functions and capabilities of the PHD Screen System, including dictionary security and maintenance.

dictionary security Specifies who has access to a dictionary and the level of access. Established by the dictionary manager.

dictionary security class A list of dictionary users who share the same access to a dictionary.

driver file In an ACCESS statement, the record-structure to which two or more related record-structures are linked in parallel.

element The smallest category of data described in the data dictionary. Elements are the basic building blocks of a PowerHouse application since they represent many individual values. It is physically represented in a record-structure by a record item. It is physically represented in an index structure by a segment. (See also item and segment.)

element usage See usage.

entity The building blocks of a PowerHouse application. Entities include databases, files, elements, items, records, record-structures and transactions. They are described in the data dictionary.

expected list A list of syntax that PowerHouse expects you to enter.

expression A term or combination of terms that yield a value.

extract The bit extract or substring extract data manipulation function.

field An item declared in a FIELD statement in QDESIGN. It becomes a location on a QUICK screen used for entering, finding, changing, and deleting data. (See also required field.)

file Declares a relation, table, or view on a screen, run, or report.

file-location The physical location of a file as it is known to the operating system.

file specification The name of a file as it is known to the operating system. It can include a location.

fine-grained A lock which affects a small quantity of data, for example, a row or record.

footing line The line displayed or printed at the end of a report, at the end of each page of a report, or at a control break. Footing lines usually contain summary information.

function Used to manipulate and test item values or return values. (See also system function.)

general term A part of a PowerHouse statement that you replace with a value when you enter the statement.

generic retrieval Retrieves the text you specify. You can use the PowerHouse default generic retrieval character (@) as a wildcard to match one or more unspecified character in the retrieval text.

generic retrieval character A character used for partial index retrieval. The at-sign (@) character is the default character.

global dictionary options PDL enables you to use definitions that are always in effect throughout an application. Global dictionary definitions are available for the standard date format, standard date separator, the language of the character set, the release and version numbers, special characters allowed in names, and pattern-matching characters.

global field In a relational database, a global field is similar to a PowerHouse element. In SQL, it corresponds to a domain or abstract datatype.

global temporary item An item that lasts for the duration of a QTP run. (See also item.)

heading line A line containing identifying information. A heading line occurring before a group of detail lines can either identify the data in the detail lines, or can identify a report as a whole. A heading line can be displayed or printed at the beginning of a report, at the beginning of each page of a report, or at the beginning of a section of a report.

hex or hexadecimal Refers to the hexadecimal numbering system.

ID-number A number identifying a field or group of fields on a PowerHouse screen.

index A data structure used by relational systems or indexed file systems to locate records quickly. The index for each record contains an index value and the address for the rest of the information associated with that index value.

indexed file A type of file organization in which all the information in a record is associated with the value of the index or segment of an index in the record. For an indexed file, the operating system automatically creates and maintains an index.

inherited transaction The properties of a transaction have been defined on one screen and are known on another screen only at execution-time.

initialize The process of filling a record buffer with assigned values, default values, or a mixture of assigned and default values for each item.

initial subset Includes the first segment of the index. Optionally, it can include more than one segment of the index if the index is a multiple-segment index.

intermediate file A transaction file used as required by QTP for processing.

isolation See consistency.

isolation levels Isolation levels specify the degree to which each transaction is isolated from the actions of other transactions. It may be one of: PHANTOM PROTECTION, READ COMMITTED, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, or STABLE CURSOR. Different database products support different transaction isolation levels - some offer a choice of isolation levels, some provide just one. Low levels of isolation mean that transactions are not well protected from each other; in other words, simultaneous transactions may get inconsistent results. Higher levels of isolation generally mean that transactions are better protected from each other. At the highest levels, each transaction may be entirely unaware of changes being made by other transactions.

item An entity in a record-structure that holds a value. (See also defined item, temporary item.)

item overlay An item (or item structure) that's defined as occupying the same areas as another item (or item structure) in a record-structure. Item overlays are specified with item substructures and item redefinitions.

keyword A word in a PowerHouse statement that's entered exactly as it appears in syntax.

label A descriptive title for a field on a PowerHouse screen.

library A file provided by OPENVMS in which you can store frequently used modules of code or text.

local field The name of a Oracle Rdb local field. A local field is similar to a PowerHouse item, or an SQL column.

local transaction A transaction that is not inherited.

locally active transaction A transaction is locally active when a local record is updated, added or retrieved. A local record is a record-structure referenced on a given screen but not passed down from a parent screen.

locking The most common method used to prevent conflicts between transactions in a database. Locking is often used to support a pessimistic concurrency control strategy (see pessimistic locking). Depending on the database, a variety of database resources (for example, rows, tables, parts of tables, indexes, pages, even the database itself) are locked to control access to them. Various levels of locking are usually supported, including shared, protected, and exclusive locks, as well as locks for read or write.

Files can be locked with different sharing options and records can be locked to further control access.

lock granularity The amount of information affected by a single lock, for example, an index, a file, a database page, or a relation.

coarse-grained

A lock which affects a large quantity of data, for example, an entire database or file.

fine-grained

A lock which affects a small quantity of data, for example, a row or record.

logical entity An entity describing how the data in your application is entered and displayed. The main logical entities are elements, record-structures, and index structures. (See also physical entity.)

logical function Used to test item values. (See also function.)

logical transaction See PowerHouse transaction.

login file A file containing commands that are automatically executed when you log onto the system. Also called logon file.

menu screen Serves as a table of contents for other screens, programs, or commands.

metacharacter A special character used in a pattern to represent a group of other characters. For example, the crosshatch (#) metacharacter represents any single digit; the caret (^) metacharacter represents any single letter. A metacharacter may appear either singly in a one-character pattern, or as a character in a multi-character pattern.

metadata Metadata, or data about data, refers to the database definitions contained in a relational database. These definitions are held within the database itself, in the same form as application data, and can be accessed in the same way as the application data is.

minidictionary When QUIZ or QTP creates a portable subfile, two files are subsequently created on the host machine: a data file and a minidictionary. The minidictionary describes the contents of the data file. (See also portable subfile.)

missing value See null value.

Mode field A special field found in the top left-hand corner of every PowerHouse screen with default user interface options. The Mode field indicates the current mode: E (Enter), F (Find), or S (Select).

multiple-segment index An index that has more than one segment.

nonsubstitution character Any character other than the substitution character, which is by default a caret (^). It's used with the PICTURE option. (See also substitution character.)

null response Refers to pressing the [Return] key without entering data.

null value A null value means either that the value for the column is unknown at this time; or that there is no value to assign. A null value is not the same as zero, because zero can be a meaningful value.

numeric expression See expression.

occurrence One item of a group of items that constitute an array. For example, an array months can have 12 occurrences, one for each month. (See also array and subscript.)

optimistic locking An approach to concurrency control, based on the assumption that most data that is read is not changed, and that it is therefore not necessary for the database to protect data until a transaction tries to change something. (See also concurrency control.)

option In syntax, refers to features of a particular statement that a user can select.

parallel driver file See driver file.

parameter See program parameter or general term.

pattern A string of characters or metacharacters or both that describe a value or group of values.

pattern matching The process of describing a value by specifying a pattern rather than an exact value.

permanent save file A permanent file that's used to store PowerHouse statements. (See also temporary save file.)

pessimistic locking An approach to concurrency control based on the assumption that anything that is read might be changed, and that the database should therefore protect everything read in case a transaction tries to change something. (See also concurrency control.)

physical entity An entity that describes how and where the data in your application is stored in the computer. The main physical entities are items, records, and files. (See also logical entity.)

picture A combination of substitution and nonsubstitution characters used to format data for display.

portable subfile A self-describing direct file that's stored in standard ASCII format and used to transfer PowerHouse data between different machine architectures.

PowerHouse transaction PowerHouse handles logical transactions. Existing only inside PowerHouse, a logical transaction coordinates and manages as many physical database transactions as needed as a single unit. PowerHouse maintains the integrity of these transactions.

Any PowerHouse screen, report, or run may involve accessing and manipulating data from several relational databases and/or database products (as well as from other file systems). A single logical PowerHouse "transaction", therefore, may encompass several database transactions.

predefined condition In syntax, refers to a condition that's permanently defined.

prepare phase Refers to the first phase of a two-phase commit. This phase usually involves verifying that all databases are accessible and capable of committing the transaction prior to the commit being issued.

primary file See primary record-structure.

primary index An index on the physical file.

primary record-structure A record-structure whose information is most important to the screen, report, or request. In QUIZ and QTP, the primary record-structure is the first record-structure named in the ACCESS statement. In QDESIGN, it is either the first record-structure named in a screen design (that is not received from a higher-level screen) or the record-structure labeled primary.

procedure In QDESIGN, a procedure is a sequence of instructions directing one of the actions affected by the QUICK screen, such as data entry, data correction, or data retrieval. QDESIGN procedures can be produced automatically by QDESIGN or specified directly by the QUICK screen designer or both.

program parameter Controls attributes such as the dictionary PowerHouse uses and where components look for input.

prompt The character used to indicate that the computer is ready for your next entry. The default PowerHouse prompt is >.

QTP transaction A compound record formed from records in multiple record-structures. It comprises a record from the primary record-structure and related records from the secondary record-structure. (See also transaction set.)

query-specification A query-specification defines a collection of rows that will be accessible when the cursor is opened.

rapid-fire entry A series of entries, separated by the separator character which is by default a semicolon (;), in a field on a PowerHouse screen. Each entry in the series is acted on as if it had been entered separately.

read-chain A set of related records or data.

read-only transaction Default for Query transaction.

read/write transaction Default for all transactions except Query.

record One set of the items in a record-structure and their values. A record is stored in a file. (See also row.)

record buffer The area of computer memory used to temporarily store the values for the items in a record.

record complex In QUIZ, a compound record formed from records in multiple record-structures. It comprises a record from the primary record-structure and related records from the secondary record-structure.

record item See item.

record-structure An ordered collection of items. Each record-structure is associated with exactly one file. For example, the record-structure of a file used to store a mailing list of your customers might consist of several items, including NAME, ADDRESS, POSTALCODE, and PHONENUMBER.

record/tuple A row in a table. It is a meaningful collection of one or more fields.

related record-structure A record-structure declared in an ACCESS statement that's in a parallel relationship to a previous record-structure in the ACCESS statement.

relational database Data in a relational database is organized into tables which are made up of rows and columns. No physical linkages between tables are specified.

relationship A dependency of one entity on another. For example, record-structures are related to files; application security classes can be related to both elements and record-structures; and so on.

repeating index Indicates that the value entered for that field can repeat in more than one data record in the record-structure.

repeating item See array.

report Data selected and displayed on the screen or printed on paper in a readable format specified by the user.

report-group A group of report-items.

report-item An item you include in a report.

report security The means of protecting a compiled QUIZ report from unauthorized use.

request Performs a set of related processes that reads through one or more input files, usually to update one or more output files.

required field A field on a PowerHouse screen in which an entry is required.

rollback Ends a transaction and undoes changes made by the transaction. Data is returned to its state prior to the start of the transaction, or the last committed instance if this was the second phase in a two-phase commit.

row A group of data values (fields, columns) from a file. (See also record.)

run Consists of one or more QTP requests.

save file See temporary save file and permanent save file.

secondary record-structure Any record-structure declared in an ACCESS statement other than the first record-structure (which is the primary record-structure).

security Protects software and information from unauthorized use. Security in PowerHouse is based on application security classes defined in the data dictionary. (See also application security.)

segment An item that's part of an index. Each index is composed of one or more segments.

slave screen A screen that's part of another screen.

sort-item See break-item.

source file See permanent save file and temporary save file.

special character Any character that isn't a letter or a number.

statement A line of instructions to one of the PowerHouse components that can either be entered in response to the PowerHouse prompts or entered into a text editor file.

stopscreen A logical stopping point for a system of screens that the user can return to at any time using a QUICK screen command.

storage option Options governing the datatype used to store the item, how its sign is stored, and the size (in bytes) that the item occupies in each record.

string A series of characters, enclosed in double or single quotation marks.

subfile A self-describing data file; that is, it's a file that contains both data and the information that describes the data. A subfile is not described in a PowerHouse dictionary.

subordinate record-structure See secondary record-structure.

subquery A subquery is identical to a query-specification with two exceptions: the subquery must project a single-column table and the syntax of the subquery includes enclosing brackets.

subscript A number used to refer to a specific item in an array. For example, if array month contains 12 occurrences, the individual items can be identified as month(1), month(2), and so on. QDESIGN doesn't allow subscription. (See also array and occurrence.)

substitution character A character, by default a caret (^), in an element picture. Each substitution character in the picture is replaced by a character from a value for display.

substitution-variable Is a variable-name prefixed by the double colon(::) which is used in SQL to identify the location for substitutions. The text is the default substitution if no other substitution is specified.

::variable-name(text)

summary information Information summarizing a group of records or record complexes. Usually, summary information appears in a footing line, where it summarizes the detail information in the preceding detail lines. (See also detail information.)

summary report In QSHOW, produces a one-line-per-entity summary of important information.

syntax PowerHouse language, statements and commands, containing specific rules and conventions (for example, the use of case, brackets, slashes) that enable communication to take place between the user and the computer.

system function Returns values that can be tested. (See also function.)

system-wide standards The set of attributes governing the defaults used by the PowerHouse components.

table/relation A two-dimensional structure that holds an arbitrary number of records.

temporary item An item you create that doesn't exist in the data dictionary. You can use temporary items in calculations and summary operations. (See also item.)

temporary save file A temporary file that's used to store PowerHouse statements as you enter them. (See also permanent save file.)

term A term is one of the following: string, number, item, expression, system-function, function-result, case-expression-set, parameter specification, USER system variable, column specification, sql-summary-operation.

text file A file of ASCII characters stored on disk. (See also ASCII.)

transaction See also database transaction, PowerHouse transaction, and conceptual transaction.

A group of operations treated as a unit by the database. It has clear beginning and ending points, and may consist of one or more data operations (as determined by the transaction designer). All database activities take place within one or more transactions. Data accesses from PowerHouse are done within database transactions.

A transaction has several important characteristics:

- it starts at a specific point in time
- it can include many individual database retrievals and updates
- it ends by being committed as a unit, or rolled back as a unit.

Transactions are used to transform a database from one consistent state to another and to prevent partial updates from occurring. When a transaction is committed, its changes are made permanent in the database. When a transaction is rolled back, the effects of its changes are undone from the database. The extent of a transaction is determined by the application designer and is called the transaction duration.

transaction commit Saves any changes made by the transaction permanently to the database, and ends the transaction.

transaction rollback Ends a transaction and undoes changes made by the transaction. Data is returned to its state prior to the start of the transaction.

transaction set In QTP, the record-structures that you access with the ACCESS statement in a request. QTP retrieves data from the record-structures to make a transaction. (See also QTP transaction.)

transaction start The point at which a transaction begins. This is an important factor in determining what data can be read for certain isolation levels.

two-phase commit A protocol supported by some databases to commit a transaction that affects more than one database. It usually involves a preliminary verification phase prior to the actual commit phase. Its purpose is to allow a transaction to span multiple independent databases, that is, the complete group of activities is either completed or is undone as a unit. For example, if a transaction involves moving information about an employee from one database to another, the two-phase commit protects against information being removed from one database but not added to the other because of some system failure.

two-phase commit: prepare phase Refers to the first phase of a two-phase commit. This phase usually involves verifying that all databases are accessible and capable of committing the transaction prior to the commit being issued.

unique index Guarantees that the value entered for a field is unique.

usage A template for defining elements.

use file A text file of QDESIGN, QTP, PDL, PHDPDL or QUIZ source statements stored on disk. The contents of the file can be loaded into a PowerHouse component using the USE statement.

user break Sends a signal that interrupts processing and returns control to the program. For UNIX and OpenVMS, the user break is [Ctrl C]. For MPE/iX, the user break is [Ctrl Y].

user mode A group of PowerHouse users who access data in the same location.

value The literal contents assigned to items. In syntax, it can indicate a string or a number, depending on the item type. (See also default.)

versioning Another method used to prevent conflicts between transactions in a database (also called multi-generational record versioning). Versioning is used most frequently to support an optimistic concurrency control strategy (see optimistic locking).

view Relational systems allows users the ability to define views of the tables in their database that consist of a subset of the rows and/or columns of one or more tables. A single-table view consists of a subset of the rows and/or columns from one table. Multiple-table views are views defined as a relational join expression composed from multiple tables.

Index

Symbols

@SETPOWERHOUSE command, 32

A

abbreviating keywords
  syntax, 281
ABSOLUTE function, 367
absolute value
  returning, 367
access
  preventing unauthorized, 402
access list
  definition, 483
ACCESSOK predefined condition, 290
account general term, 275
Action bar
  definition, 483
Action commands
  table, 334
Action field
  definition, 483
ADDCENTURY function, 368
addition
  plus sign (+), 300
alias
  definition, 483
aligning
  decimals, 330
ALLBASE MODULE EXTENSION resource file statement, 180
ALLBASE/SQL
  database modules specifying owners, 224
  DATETIME datatype, 318
allowed syntax
  definition, 483
alpha (^) metacharacter, 351
ALTEREDRECORD predefined condition, 290, 293
alternate index
  definition, 483
  generating SOUNDEX function, 457
alternative message files
  service layer, 269
alternative messages, 261
  displaying, 262
  in PowerHouse, 264
  PowerHouse, 262
  templates, 263
ampersand (&)
  continuation character, 281
  in conditional compile statements, 282
AND logical operator
  compound conditions, 282, 296
any (?) metacharacter, 351
application
  definition, 483
application line
  definition, 483
  general term, 275
application manager
  definition, 483
application security
  definition, 484
application security class (ASC)
  definition, 484
  determining users, 425
argument
  definition, 484
arithmetic operators
  numeric expressions, 300
array
  definition, 484
  general term, 275
arrays, 284, 285, 286-287
  automatic initialization, 287
  calculating, 287
  editing, 286
  in QTP, 286
  in QUIZ, 287
  OCCURRENCE predefined item, 306
  referencing without subscripts, 285, 286
  reporting individual occurrences, 285, 286
  subscripts, 285
  subscripts, values, 285
  summing entire, 466
  summing intermittent occurrences, 467
  summing items, 466
  summing subsets, 467
  writing all occurrences to subfiles, 285
asc general term, 275
ASCII
  converting from hexadecimal, 415
  converting to hexadecimal, 416
  definition, 484
  function, 369
ASSUMED option
  SET statement, 308
asterisk (*)
  metacharacter, 351
  multiplication, 300
at-sign (@)
  as a continuation character, 282
  in conditional compile statements, 282
  metacharacter, 351
attach
  definition, 484

attribute
definition, 484
ATTRIBUTE function, 370
attributes
displaying default assumptions, 328
numeric elements, 326, 326-329
AUDITSTATUS
function, 371
auto program parameter, 75
locating files, 42
autodetach program parameter, 76
AUTODETACH resource file statement, 181
automatic
initialization of arrays, 284
automatic initialization
arrays, 287
AVG
sql-summary-operation, 304
avoiding
conflicting names, 281
Axiant
Terminal Compatible property, 130

B

backslash (\)
in messages in the service layer, 267
metacharacter, 351
base-date general term
DECIMALTIME function, 392
BETWEEN condition
specifying ranges of values, 298
binary numbers
storing, 345
bit extract function, 372
BITEXTRACT function, 373
blank when zero
output conversion, 327
blanks
removing trailing from strings, 474
BLOB datatype, 317
blobs
assigning to a character item, 345
handling contents, 345
non-text, 345
QDESIGN, 345
restrictions when treated as text items, 346
sizes, 345
support, 345-346
working with contents, 387
BLOCK TRANSFER control structure
charmode program parameter, 81
PROMPTOK predefined condition, 292
BLOCKMODE, 291
blockmode program parameter, 77
boolean operations
truth table, 297
BOTTOM UP option
UPDATE resource file statement, 257
bottomup option
update program parameter, 170
braces ({ })
syntax symbol, 274
break-item
definition, 484
BRIEF option
DBAUDIT resource file statement, 195
brief option
dbaudit program parameter, 96
broadcast program parameter, 78
BROADCAST resource file statement, 182
buffer
definition, 484
BUILD statement
locating files, 41
bulkfetch program parameter, 79
BULKFETCH resource file statement, 183

C

calculating
arrays, 287
checksums, 380
compensating for input scaling, 331
floating point numbers, FLOAT datatype, 332
number of days from a base date, 388
numeric items, 331
value of expressions, 301
calculations
multiplication and percentage, 329
calling
operating systems from PowerHouse, 223
caret (^) metacharacter, 351
case
syntax characters, 273
case sensitive names, 113
case-expression-set
CHOOSE statement, 302
general term, 275
case-processing
CHOOSE statement, 302
DEFINE statement, 302
general term, 275
SQL, 303
case-value
DEFINE statement, 302
CATEGORY statement
service layer message file, 267
cc program parameter, 80, 282
CC resource file statement, 184
CEILING function, 374
CENTER|CENTRE function, 375
centering text, 375
CENTURY function, 376-377
CHANGEMODE predefined condition, 290, 291
changing (see also editing, modifying, redefining)
conditions, 296
order of precedence, 296
char general term, 275
CHARACTER
datatype, 317
character
definition, 484

character *(cont'd)*
general term, 275
patterns, 353
CHARACTER option
TERMINAL READ resource file statement, 248
character strings
inverting, 441
items, 415, 416
left-justified, 420
CHARACTER_LENGTH function, 379
CHARACTERMODE, 291
characters
adding space between, 458
centering strings, 375
general term, 275
item assignment to blobs, 345
items addressed as strings, 378
replacing substitution, 462
shifting case, 401, 477
strings, converting numbers, 369
CHARACTERS function, 378
characters strings
converting to numbers, 428
reversing, 441
charmode program parameter, 81
check digit
calculating, 380
checksum
definition, 484
CHECKSUM function, 380-381
CHECKSUM710, 82, 185
checksum710 program parameter, 82-83
backwards compatibility switch for 8.30D, 82
using in PHD dictionaries, 82, 185
CHECKSUM710 resource file statement, 185
checksums
data record security, 381
returning, 380
class
definition, 484
CLOSE DETACH resource file statement, 187
close_detach program parameter, 84
cluster
definition, 484
coarse-grained lock granularity
definition, 489
code
processing based on conditional compile statements, 80, 282
coded record-structure
definition, 484
codes
creating phonetic, 456-457
phonetic, constructing for string, 357
QTP error status, 24, 25
colon-variable general term, 275
column
definition, 484
column general term, 275
column-name general term, 275
columnowner program parameter, 85
COLUMNOWNER resource file statement, 188
columnspec general term, 275
combining functions, 359
COMMANDCODE function, 383
COMMANDMESSAGE function, 384
COMMANDOK predefined condition, 291
commands
action field AMGR, 34
action field DMGR, 34
action field USER, 34
entering shell, 283
PowerHouse (OpenVMS), 31
SETDICTIONARY, 36
SHOWDICTIONARY, 38
SHOWPOWERHOUSE, 39
SHOWQUOTA, 40
COMMANDSEVERITY function, 385
COMMANDSTATUS function, 386
comments
entering, 281
service layer message file, 268
commit transaction
definition, 484
COMMITPOINTS OBSOLETE resource file statement, 190
commitpoints program parameter, 87
committing
transactions, 292
comparisons
relational terminology, summary, 483
COMPATIBLE option
TERMINAL BLOCKMODE resource file option, 244
compatible option
blockmode program parameter, 77
compile
definition, 485
flags, predefined, 282
compile flags
conditional, 80, 184
compiled file
definition, 485
compiled reports
establishing, 75
compiled screens
locating with the GO statement, 44
component
definition, 485
COMPONENT statement
service layer message file, 266
compound
conditions, 282, 296
COMPRESS BUFFERS resource file statement, 191
compress_buffers program parameter, 88
concatenation
in SQL, 303
conceptual transaction
definition, 485
concurrency
definition, 485
concurrency control
definition, 485
condition
definition, 485
EXISTS, SQL, 299

condition *(cont'd)*
general term, 275
conditiona
expressions, 301
conditional
compilation, multiple, 282
compilation, single, 282
compile flags, 184
compile flags, predefined, 282
compile statements, at-sign (@), 282
compile statements, entering, 282
expression, 301
expression, definition, 485
conditional compile flags, 80
conditional error status codes, 22, 23
conditional error status settings
QTP, 24, 25
conditional expression
ELSE, 302
IF, 301
condition-command-list general term, 275
condition-expression general term, 275
conditions, 289-297
compound, 296
creating compound, 282
general form, 289
IS NULL in SQL, 299
logical expression, 289
logical-function, 289
null values, 297
order of precedence, 296
predefined, 290-293, 295
predefined EXISTS, 295
predefined RECORD EXISTS, 295
predefined transactions, 292
simple, 295
SQL, 298-299
testing stored values, 331
confirmer program parameter, 89
conflicting names
avoiding, 281
consistency
definition, 485
consistency checks, 57
CONSOLE KEYS resource file statement, 192
consolekeys program parameter, 90
constructing
phonetic code for a string, 357
CONTENTS function, 387
continuation character
ampersand (&), 281, 282
at-sign (@), 282
continuing
comments, 281
conditional compile statements, 282
statements, 281
control break
definition, 485
control file
definition, 485
control group
definition, 485
converting
ASCII to hexadecimal, 416
character strings to numbers, 428
dates to number of days since January 1 1900, 391
eight-digit dates to six-digit dates, 440
hexadecimal to ASCII, 415
input, 326
numbers to character strings, 369
output, 327
six-digit dates to eight-digit dates, 368
values, storage, 326, 326-329
converting redefinitions, 66
converting substructures
for DISAM, 66
copyright, 2
CORRECTMODE predefined condition, 290, 291
correlation names
COLUMNOWNER resource file statement, 188
COUNT
sql-summary-operation, 304
counting
null values in QUIZ, 348
createall program parameter, 91
createbase program parameter, 92
createfile program parameter, 93
creating
compound conditions, 282
messages in PowerHouse, 264
messages, designer, 271
service layer alternative message file, 269
Ctrl-C
definition, 485
Ctrl-Y
definition, 485
currencies
decimal, specifying, 328
CURRENT option
VMSDATE resource file statement, 259
cursor
definition, 485
general term, 275
cursor-name general term, 275
CURSOROPEN predefined condition, 291
cursorowner program parameter, 94
cursor-reference general term, 275

D

data
detecting invalid, 380
dictionary, definition, 486
encrypting numeric, 378
file, definition, 486
item, definition, 486
manipulation function, definition, 486
record, definition, 486
data definition
definition, 486
data dictionaries
opening, 108, 202
returning system title, 470

data dictionary
  locating, 41
data storage
  DISAM, 65
database
  definition, 485
  general term, 275
DATABASE resource file statement, 193
database transaction
  definition, 486
databases
  providing password and userid, 193, 194
dataset general term, 275
datatype
  definition, 486
  general term, 275
datatype mappings
  DISAM, 65
datatypes
  and items in PowerHouse, 306
  BLOB, 317
  CHARACTER, 317
  DATE, 317
  DATETIME, 318
  default, defined items, 331
  FLOAT, 318, 332, 438
  forms of storage, 309
  FREEFORM, 320
  INTEGER, 320, 330
  INTERVAL, 321
  items, 306
  JDATE, 321
  non-relational, 311
  NUMERIC, 321
  numeric, fractional amounts, 330
  numeric, scaling, 330-333
  ODBC, 317
  PACKED, 321, 330
  PHDATE, 322
  PowerHouse relational, 313-316
  relational specifics, 316
  TIME, 316
  unsupported DB2, 317
  user-defined, 324
  VARCHAR, 322
  VMSDATE, 322
  ZDATE, 323
  ZONED, 323, 330
DATE
  sql-date-literal, 303
date
  expression, definition, 486
DATE datatype, 317
DATE function, 388
DATE general term
  DATEEXTRACT function, 390
date-expression general term, 275
date-expressions
  SQL, 303
DATEEXTRACT function, 390
date-format general term, 275
dates
  calculating number of days from a base date, 388
  converting eight-digit to six-digit, 440
  converting number of days since January 1 1900, 391
  converting six-digit to eight-digit, 368
  expressions, 301
  extracting items, 390
  mixing eight-digit and six-digit, 440
  mixing six-digit with eight-digit, 368
  negative values, 391
  patterns, 354
  returning current, 468, 469
  setting to last day of a month, 419
DATETIME datatype, 318
DAY extract-option
  EXTRACT function, 404
DAY general term
  DATEEXTRACT function, 390
DAYS function, 391
days general term, 275
DB2
  unsupported datatypes, 317
dbaudit program parameter, 96, 96-97
DBAUDIT resource file statement, 195
dbdetach program parameter, 98
DBDETACH resource file statement, 197
dbwait program parameter, 99
DBWAIT resource file statement, 198
dclt program parameter, 100
dd general term, 275
ddd general term, 275
deadlock
  definition, 486
debug (QDESIGN) program parameter, 101
debug (QUICK) program parameter, 102
DEBUG resource file statement, 199
Debugger
  controlling level of capacity, 199
debugger
  controlling level of capacity, 102
decimal
  alignment in display, 330-333
  characters embedded in FREEFORM items, 320
  currencies, specifying, 328
  numbers, PACKED, 321
  numbers, ZONED, 323
DECIMALTIME function, 392
decoding
  encrypted keys, 393
DECRYPT function, 393
default
  assumptions, display attributes, 328
  assumptions, input conversion, 327
  definition, 486
  messages, 262
  PowerHouse files, 48
DEFAULT CURSOR OWNER resource file statement, 200
DEFAULT option
  BROADCAST resource file statement, 182
default option
  broadcast program parameter, 78

default-location
  definition, 486
defaults
  datatype, defined items, 331
  INPUT SCALE, 331
  picture string, 330
  prompt character, 281
  scaling factor, numeric items, 330
DEFERRED option
  BROADCAST resource file statement, 182
deferred option
  broadcast program parameter, 78
DEFINE statement
  testing for null values, 292, 295
defined items, 306
  default datatype, 331
  definition, 486
  missing values, 292
  null values, 292, 293, 295, 347
  testing, 292, 293, 295
defined-item general term, 275
DELAY option
  SUBDICTIONARY resource file statement, 239
delay option
  subdictionary program parameter, 160
deleteall program parameter, 103
deletebase program parameter (MPE/iX), 104
DELETEDRECORD predefined condition, 290, 293
deletefile program parameter, 105
DELETESYSTEMVAL function, 394-401
deleting
  shared memory sections, 57
derived-column general term, 275
designated files, 48
  definition, 486
designer messages, 261, 270-271
  creating, 271
  format, 271
  SUBSTITUTE function, 462
DESIGNER NORETAIN resource file statement, 201
designer_noretain program parameter, 106
detail information
  definition, 486
detail lines
  definition, 486
detail program parameter, 107
detail report
  definition, 486
dictionaries
  opening at startup, 108
dictionary, 487
  accessing installed, 56
  accessing uninstalled, 57
  definition, 486
  displaying current name, 38
  establishing for PowerHouse session, 36
  installing PDL, 56
  opening at startup, 202
  PDC shared, 53
  PDL shared, 56
  PHD shared, 55
  returning system title, 470
dictionary manager
  definition, 487
dictionary program parameter, 108
DICTIONARY resource file statement, 202
dictionary security
  definition, 487
dictionary security class
  definition, 487
dicttype (OpenVMS) program parameter, 109
direct_file_base_zero program parameter, 110
DIRECTORY resource file statement, 204
DISABLE NULLS resource file statement, 205
DISABLE option
  SUBDICTIONARY resource file statement, 239
disable option
  subdictionary program parameter, 160
disable_nulls program parameter, 111
disabling
  support for null values, 347
DISAM
  converting, 66
  data storage on NT/2000/XP, 65
  datatype mappings, 65
  reading and writing, 65
  retrieving, 65
displaying
  current dictionary name, 38
  name of active version of PowerHouse, 39
  negative values, 329
  numeric datatypes, 330-333
  process quotas, 40
  statistics, 237
  statistics in QTP, 25
displaying default messages, 262
division
  by zero, 300
  returning remainders, 427
  slash (/), 300
DO BLOB verb, 345
domain
  definition, 487
domains
  SQL entity, corresponding to PowerHouse entities, 483
dont_store_module program parameter, 112
double
  asterisk (**), exponentiation, 300
  encryption, 402
DOWN option
  ROUND function, 443
  SHIFT resource file statement, 236
DOWNSHIFT function, 401
downshift program parameter, 113
driver file
  definition, 487
dsc general term, 275

E

EDIT statement, 286
editing
  arrays, 286

eight-digit dates
converting to six-digit, 440
mixing with six-digit dates, 368, 440
element
definition, 487
general term, 275
usage, definition, 487
elements
numeric, attributes, 326, 326-329
size, 310
typed, input conversion, 326
types, output conversion, 327
ellipsis (...)
syntax symbol, 274
ELSE
conditional compile statements, 282
conditional expression, 302
ELSEIF
conditional compile statements, 282
ENABLE option
SUBDICTIONARY resource file statement, 239
enable option
subdictionary program parameter, 160
encoding encrypted keys, 402-403
ENCRYPT function, 402-403
CHECKSUM function, 381
encrypted keys
decoding, 393
encoding, 402-403
encrypting
numeric data, 378, 415, 416
encryption
double, 402
security, 402
endian data formats, 65
ENDIF
conditional compile statements, 282
entering
comments, 281
conditional compile statements, 282
general terms, 273
keywords, 273
PDL program parameters, 27
PHDPDL program parameters, 28
program parameter syntax, 67
QDESIGN program parameters, 20
QSHOW program parameters, 29
QTP program parameters, 24
QUICK program parameters, 21
QUIZ program parameters, 22
QUTIL program parameters, 30
shell commands, 283
statements, 281
entity
definition, 487
general term, 275
Entry mode
status of predefined conditions, 290
ENTRY RECALL resource file statement, 206
ENTRYMODE predefined condition, 290, 291
entryrecall program parameter, 114
environment variables
PHEDIT, 138
EQUAL option
LINKVALUE function, 421
equal sign
operator in conditions, 298
equal to (=)
operator in conditions, 289
errlist program parameter, 115
error
messages, alternative, 264
messages, locating, 261-262
error messages
prompting user for verification, 89
returning text, 384
ERROR option
DEBUG resource file statement, 199
NONPORTABLE resource file statement, 218
OBSOLETE resource file statement, 221
error option
debug program parameter, 102
nonportable program parameter, 129
obsolete program parameter, 136
error status settings in QTP, 24, 25
error status settings in QUIZ, 22, 23
errors
entering statements, 281
escape (!) metacharacter, 351, 353
ESCAPE option
SQL, 298, 356
etp general term, 275
evaluating series of expressions
based on conditions, 301
exact-match characters
pattern matching, 351
exceptions
null value rules, summary, 348
exclamation mark (!)
metacharacter, pattern matching, 351
exclamation mark and zero (0!) metacharacter
pattern matching, 351
exclusions
null values, 349
EXECUTE statement
locating files, 42
procloc program parameter, 145
EXISTS
predefined condition, 295
EXISTS predefined condition, 293
EXIT resource file statement, 207
EXIT statement
QUICK, 21
exiting
PDL, 27
PHDPDLPHDPDL, 28
QDESIGN, 20
QSHOW, 29
QTP, 24
QUICK, 21
QUIZ, 22
QUTIL, 30

expected list
definition, 487
exponent
floating point numbers, 318
exponentiation, 333
double asterisk (**), 300
expression
definition, 487
expression general term, 275
expressions, 300-302
calculating value, 301
case-expression-set, 302
case-processing, 303
case-value, 302
conditional, 301
date, 301, 303
logical, 289
numeric, 300, 303
SQL, 303
string, 303
strings, 300
within program variables, 304
extension general term, 275
extract
definition, 487
EXTRACT function, 404
extracting
bits from numbers, 372
substrings from strings, 465
extract-option general term
DATEEXTRACT function, 390
EXTRACT function, 404

F

FALSE results, 297
fastread program parameter, 116
fdl program parameter, 117
field
definition, 487
general term, 275
FIELD option
TERMINAL CHARACTERMODE resource file statement, 245
field option
charmode program parameter, 81
FIELD statement
VALUES option, input scaling, 331
FIELDTEXT
predefined item, 306
predefined item, null values, 347
FIELDVALUE, 347
predefined item, 306
predefined item, null values, 347
file
definition, 487
general term, 275
message, 261
specification, definition, 487
File Definition Language *See* fdl
file names
locating ODS5, 46
FILE option
DBAUDIT resource file statement, 195
file option
dbaudit program parameter, 96
file-location
definition, 487
filelocation general term, 275
filename general term, 275
files
designated, 48
locating, 41-44, 145
locating a data dictionary, 41
message templates, 263
PowerHouse default, 48
filespec general term, 275
fill character
output conversion, 327
Find mode
status of predefined conditions, 290
FIND procedure
PATH predefined item, 307
FINDMODE predefined condition, 290, 291
fine-grained
definition, 487
fine-grained lock granularity
definition, 489
FIRST function, 405
fkc_put_order option
update program parameter, 170
flags
conditional compile, 80, 184
predefined conditional compile, 282
FLOAT
item, storage, 331
float character
output conversion, 327
FLOAT datatype, 318
RANDOM function, 438
floating point
representation, calculations, 332
types, 318
FLOOR function, 406
footing line
definition, 487
FOR control structure, 284
OCCURRENCE predefined item, 306
FOREIGN KEY CONSTRAINT option
UPDATE resource file statement, 257
format
general term, 275
messages, designer, 271
FORMAT option, 161, 240
FORMATNUMBER function, 407
formatting
messages in PowerHouse, 263
fractional
amounts, retaining fractional portion, 330
fractional values
rounding down, 406
rounding up, 374
FREEFORM datatype, 320

FULL option
DBAUDIT resource file statement, 195
full option
dbaudit program parameter, 96
function
definition, 487
SIGNONGROUP, 453
function-result general term, 275
functions
ABSOLUTE, 367
ADDCENTURY, 368
ASCII, 369
ATTRIBUTE, 370
AUDITSTATUS, 371
bit extract, 372
BITEXTRACT, 373
CEILING, 374
CENTER|CENTRE, 375
CENTURY, 376-377
CHARACTER_LENGTH, 379
CHARACTERS, 378
CHECKSUM, 380-381
combining, 359
COMMANDCODE, 383
COMMANDMESSAGE, 384
COMMANDSEVERITY, 385
COMMANDSTATUS, 386
CONTENTS, 387
DATE, 388
DATEEXTRACT, 390
DAYS, 391
DECIMALTIME, 392
DECRYPT, 393
DELETESYSTEMVAL, 394-401
DOWNSHIFT, 401
ENCRYPT, 402-403
EXTRACT, 404
FIRST, 405
FLOOR, 406
FORMATNUMBER, 407
GETSYSTEMVAL, 412-414
HEXDECODE, 415
HEXENCODE, 416
INDEX, 414
INTERVAL, 417
JCW, 418
LASTDAY, 419
LEFT JUSTIFY|LJ, 420
LINKVALUE, 421
LOGONID, 422
LOWER, 423
MATCHPATTERN, 424
MATCHUSER, 425
MISSING, 426
MOD, 427
NCONVERT, 428
nesting, 359
NULL, 429
OCCURRENCE, 430
OCTET_LENGTH, 432
OLDVALUE, 433
PACK, 434
functions *(cont'd)*
PORTID, 435
POSITION, 436
PowerHouse, 359-481
PowerHouse table, 359-365
PROCESSLOCATION, 437
QDESIGN, 359-481
RANDOM, 438
RECORDLOCATION, 439
REMOVECENTURY, 440
REVERSE, 441
RIGHT JUSITFY|RJ, 442
ROUND, 443-445
SCREENLEVEL, 446
SETSYSTEMVAL, 447-450
SHIFTLEVEL, 451
SIGNONACCOUNT, 452
SIGNONGROUP, 453
SIGNONUSER, 454
SIZE, 455
SOUNDEX, 456-457
SPREAD, 458
SUBSTITUTE, 462-463
SUBSTRING, 464
SUBSTRING EXTRACT, 465
SUM, 466
SYSDATE, 468
SYSDATETIME, 469
SYSNAME, 470
SYSPAGE, 471
SYSTIME, 472
TERMTYPE, 473
TRUNCATE, 474
UIC, 475
UPPER, 476
UPSHIFT, 477
VALIDPATTERN, 478
VMSTIMESTAMP, 479
WEBLOGONID, 480
ZEROFILL, 481

G

general term
definition, 487
general terms, 275
GENERATE option
LIST resource file statement, 213
generating
alternate indexes, 457
generating indexes
compressed, 380-381
generic retrieval
definition, 488
generic retrieval character
definition, 488
GETSYSTEMVAL function, 412-414
global dictionary options
definition, 488
global field
definition, 488

global temporary item
definition, 488
global temporary items, 306
GO statements
locating compiled screens, 44
greater than (>)
operator in conditions, 289, 298
greater than or equal to (>=)
operator in conditio, 298
operator in conditions, 289
group general term, 275

H

handling
contents of blobs, 345
heading line
definition, 488
hex or hexadecimal
definition, 488
hexadecimal
converting from ASCII, 416
converting to ASCII, 415
HEXDECODE function, 415
HEXENCODE function, 416
HIGHEST option
LINKVALUE function, 421
highlight general term, 275
HOUR extract-option
EXTRACT function, 404
HOUR option
DATEEXTRACT function, 390
hours general term, 275
HPSLAVE EXTRA LINE resource file statement, 208
HPSLAVE SPLIT LINE resource file statement, 209
hundredths general term, 275

I

ID-number
definition, 488
IF
conditional compile statements, 282
conditional expression, 301
IGNORE option
OBSOLETE resource file statement, 221
ignore option
obsolete program parameter, 136
indented syntax, 274
index
definition, 488
general term, 275
INDEX function, 414
indexed file
definition, 488
indexes
generating alternate, 457
generating compressed, 380-381
indexname general term, 275
informational messages
alternative, 264
confirmation by user, 89
locating, 261-262
inherited transaction
definition, 488
initial subset
definition, 488
initialization
arrays, 284, 287
initialize
definition, 488
INITIALIZE NULLS resource file statement, 210
initializing
non-relational data structure, 347
null values, 347
initnulls program parameter, 118
input
conversion, 326
conversion, default assumptions, 327
scaling, 330
scaling, fractional amounts, 330
installed dictionaries
accessing, 56
INTEGER
datatype, 330
item, storage, 331
INTEGER datatype, 320
fractional amounts, 330
INTEGER SIZE 6 resource file statement, 211
intermediate file
definition, 488
intermittent occurrences of arrays
summing, 467
internal
messages, 261
INTERVAL
sql-date-literal, 303
INTERVAL datatype, 321
INTERVAL function, 417
intsize6 program parameter, 119
invalid data
detecting, 380
invalid patterns
detecting, 478
inverting
character strings, 441
IS NULL condition
SQL, 299
isolation
definition, 488
isolation levels
definition, 489
item
and datatypes in PowerHouse, 306
definition, 489
general term, 275
item EXISTS predefined condition, 293
item MISSING predefined condition, 293
item names
size, 161, 240
item NULL predefined condition, 293
item overlay
definition, 489
ITEM statement
arrays, 286

ITEM statement *(cont'd)*
assigning null values, 348
itemname EXISTS predefined condition, 293
items
addressed as character strings, 378
arrays, subscripts, 285
datatype defaults, 310
datatypes, 306
datatypes, input conversion, 326
datatypes, output conversion, 327
defined, 306
global temporary, 306
numeric alignment and storage, 330
operands in conditions, 289
record, 307
searching, 308
size, 310
subscripting, 285
summing in arrays, 466
temporary, 308
types, 309
items addressed as character strings, 415, 416

J

JCW function, 418
jcwbase program parameter, 120
JCWBASE resource file statement, 212
JDATE datatype, 321
DATEEXTRACT function, 390
Julian date, 321

K

keyword
definition, 489
keyword general term, 275
keywords
abbreviating, 281
entering, 273

L

label
definition, 489
language
compatibility, ZONED datatype, 324
languages in messages, 261
LASTDAY function, 419
LAYOUT option
LIST resource file statement, 213
leading
sign, input conversion, 326
sign, output conversion, 327
spaces, FREEFORM items, 320
spaces, replacing with zeros, 481
leading spaces
eliminating, 434
LEFT JUSTIFY|LJ function, 420
left parenthesis (() metacharacter, 351
left-justification
characters in strings, 420
length general term
bit extract function, 372
Substring Extract function, 465
SUBSTRING function, 464
less than (<)
operator in conditions, 289, 298
less than or equal to (<=)
operator in conditions, 289, 298
LET verb
AUDITSTATUS function, 371
levels
number of current screen, 446
LIKE condition
pattern-matching, 298, 356
LINE option
TERMINAL READ resource file statement, 248
lineread program parameter, 121
lines
preventing splitting, 126
linkage general term, 275
linkitem general term, 275
LINKVALUE function, 421
LIST option
SET resource file statement, 234
USE resource file statement, 258
list program parameter, 122
LIST resource file statement, 213
listing
resource files, 234
source statements, 122, 213
local transaction
definition, 489
locally active transaction
definition, 489
testing, 292
locating
compiled screens with the GO statement, 44
data dictionary, 41
files, 41-44, 145
items, 308
messages, 261-262
ODS5 file names, 46
QKGO files, 43
service layer message files, 265
start screens, 43
subfiles, 44
locating files
auto program parameter, 42
BUILD statement, 41
EXECUTE statement, 42
SAVE statement, 41
USE statement, 42
LOCATION MODULE resource file statement, 214
location option
DIRECTORY resource statement, 204
LOCATION PROCESS resource file statement, 215
lock granularity
definition, 489
locking
definition, 489
lockword general term, 275
lockword program parameter, 123

LOCKWORD resource file statement, 216
log file
  naming, 147
logical entity
  definition, 490
logical expressions
  conditions, 289
logical function
  definition, 490
  general term, 275
logical sizes, 310
logical transaction
  definition, 490
logical-function
  condition, 289
logicals, CHECKSUM710, 82, 185
login file
  definition, 490
logonid
  general term, 275
  returning, 422, 454
LOGONID function, 422
LOWER function, 423
lowercase
  characters, syntax, 273
  shifting to uppercase, 477
lowercase characters, 401
LOWEST option
  LINKVALUE function, 421

M

m general term, 275
mantissa
  floating point numbers, 318
matching
  metacharacters, 353
  patterns, 351-356
  patterns in SQL, 356
MATCHPATTERN function, 424
  TRUNCATE, 474
MATCHUSER function, 425
MAX
  sql-summary-operation, 304
menu screen
  definition, 490
message
  definition in message file, 267
message file format
  service layer, 266
message files
  CATEGORY statement, 267
  COMPONENT statement, 266
  customized, 261
  default service layer, 266
  locating service layer, 265
  NAME statement, 266
  PARAMETER statement, 267
  SEVERITY statement, 266
message format
  runtime, 270
message templates, 263
messages, 261-271
  alternative, 261
  alternative in PowerHouse, 264
  alternative PowerHouse, 262
  confirmation by user, 89
  creating in PowerHouse, 264
  default, 262
  designer, 261, 270-271, 462
  designer format, 271
  designer, creating, 271
  displaying alternative, 262
  displaying default, 262
  formatting in PowerHouse, 263
  in PowerHouse, 261
  in service layer, 261, 265
  internal, 261
  locating, 261-262
  modifying in PowerHouse, 264
  nonportable syntax, 129, 218
  obsolete syntax, 136, 221, 239
  PowerHouse, 261
  returning text, 384
  runtime format in service layer, 270
  service layer, 265-270
  substitution characters, 267, 271
  text order numbers, 271
  using nontermcompat to suppress, 130
MESSAGES option
  DBAUDIT resource file statement, 195
messages restructuring, 271
metacharacter, 351
  definition, 490
meta-character general term, 275
metacharacters
  alpha (^), 351
  any (?), 351
  changing reserved, 353
  definition, 351
  escape (!), 351
  left parenthesis ((), 351
  not (\), 351
  null (!0), 351
  optional (<), 351
  optional repeating (*), 351
  or (|), 351
  pattern matching, 351-356
  repeat (>), 351
  reserved, 352
  right parenthesis ()), 351
  wild (@), 351
metadata
  definition, 490
metadata references
  COLUMNOWNER resource file statement, 188
MIN
  sql-summary-operation, 304
minidictionary
  definition, 490
minus sign (-)
  FREEFORM items, 320
  subtraction, 300

MINUTE extract-option
EXTRACT function, 404
MINUTE general term
DATEEXTRACT function, 390
minutes general term, 275
MISSING function, 426
MISSING predefined condition, 295
missing value
definition, 490
mixing
six-digit and eight-digit dates, 368
mm general term, 275
mmm general term, 275
MOD function, 427
Mode field
definition, 490
modifying
messages in PowerHouse, 264
substitution order, 271
moduleext program parameter, 124
moduleloc program parameter, 125
monetary values
rounding down, 406
rounding up, 374
MONTH extract-option
EXTRACT function, 404
MONTH general term
DATEEXTRACT function, 390
MPE/iX
notimezone program parameter, 166
QTP error status settings, 24
QUIZ error status settings, 22
setting up the PowerHouse environment, 18
timezone program parameter, 166
MSGS option
DBAUDIT resource file statement, 195
msgs option
dbaudit program parameter, 96
multiple
conditional compilation, 282
conditions, modifying, 296
multiple-segment index
definition, 490
multiplication
asterisk (*), 300
calculations, 329
mutually exclusive options
syntax symbol, 274

N

n general term, 275
name general term, 275
NAME statement
service layer message file, 266
names
avoiding conflicting, 281
case sensitivity, 113
naming
log file, 147
trace files, 147
NCONVERT function, 428
NEAR option
ROUND function, 443
negative values
dates, 391
displaying, 329
storing FREEFORM, 320
nesting
functions, 359
newlink retainmarkpp, 154
NEWRECORD predefined condition, 290, 293
nls (no line split) program parameter, 126
no extra line*See* nxl
no line split *See* nls
noautodetach program parameter, 76
noblobs program parameter, 127
NOBLOBS resource file statement, 217
nobreakset program parameter, 128
noconsolekeys program parameter, 90
nodbdetach program parameter, 98
nodbwait program parameter, 99
nodcl program parameter, 100
NODELAY option
SUBDICTIONARY resource file statement, 239
nodelay option
subdictionary program parameter, 160
nodetail program parameter, 107
nofdl program parameter, 117
noinitnulls program parameter, 118
nointsize6 program parameter, 119
NOLIST option
SET resource file statement, 234
USE resource file statement, 258
nolist program parameter, 122
NONE option
DBAUDIT resource file statement, 195
SHIFT resource file statement, 236
nonportable program parameter, 129
NONPORTABLE resource file statement, 218
nonportable syntax
messages, 129
non-relational datatypes, 311
nonsubsitution character
definition, 490
nontermcompat program parameter, 130
non-text blobs, 345
noomnidex program parameter, 137
NOOMNIDEX resource file statement, 222
noosaccess program parameter, 138
noowner program parameter, 131
NOOWNER resource file statement, 219
noprefix_ownername program parameter, 132
noretainmark program parameter, 154
noreuse_screen_buffers program parameter, 155
NOSEARCH option
SUBDICTIONARY resource file statement, 239
nosearch option
subdictionary program parameter, 160
NOSET WARN STATUS resource file statement, 220
nosetjobshow program parameter, 158
nosetwarnstatus program parameter, 133
noshift program parameter, 113
nostatistics program parameter, 159

NOT
compound conditions, 282
not (\) metacharacter, 351
not equal to (<>)
operator in conditions, 289, 298
NOT logical operator
compound conditions, 296
notermpoll program parameter, 165
notimezone program parameter, 166
equivalent resource file statement, 166
notpi program parameter, 167
notrusted program parameter, 168
nouicbrackets program parameter, 134
nowarn option
debug program parameter, 102
nonportable program parameter, 129
obsolete program parameter, 136
NOWARNING option
DEBUG resource file statement, 199
NONPORTABLE resource file statement, 218
OBSOLETE resource file statement, 221
NT/2000/XP
DISAM data storage, 65
NULL
condition in SQL, 348
results, 297
null (0!) metacharacter, 351
NULL function, 429
NULL predefined condition, 293, 295
null response
definition, 490
NULL value
returning, 426, 429
null value
definition, 490
NULL VALUE NOT ALLOWED option
FIELD statement, 349
null values
assigning, 348
comparisons, 297
conditions, 297
counting in QUIZ, 348
disabling support, 347
entering, 347
exceptions to rules, 348
initializing, 347
missing, 292
operating, 348
selective record retrieval, 348
support in relational databases, 347-349
testing, 292, 293, 295
numbers
converting to character strings, 369
returning absolute values, 367
returning NULL value, 426, 429
returning random, 438
returning rounded, 443-445
rounding next-highest integer, 374
rounding next-lowest integer, 406
numeric
datatypes, fractional amounts, 330
elements, attributes, 326, 326-329
numeric *(cont'd)*
expressions, 300
expressions, arithmetic operators, 300
expressions, array subscripts, 285
items, alignment, 330
items, calculation, 331
items, storage, 330
patterns, 353
picture, output conversion, 327
numeric data
encrypting, 378, 415, 416
NUMERIC datatype, 321
numeric expression
definition, 490
numeric expressions
evaluating using 8-byte floating point, 300
numeric-expression
SQL, 303
numeric-expression general term, 275
numeric-item general term, 378
nxl (no extra line) program parameter

O

OBSOLETE option
VMSDATE resource file statement, 259
obsolete program parameter, 136
OBSOLETE resource file statement, 221
obsolete syntax
messages, 136, 239
OCCURRENCE
predefined item, 306
system function, 284
occurrence
definition, 490
OCCURRENCE function, 430
occurrences
arrays, referencing in QUICK, 284
OCTET_LENGTH function, 432
ODBC datatype, 317
ODS5 file names
locating, 46
OF record-structure
qualifier, 290
OFF option
DBDETACH resource file statement, 197
DBWAIT resource file statement, 198
DEFAULT CURSOR OWNER resource file statement, 200
HPSLAVE EXTRA LINE resource file statement, 208
HPSLAVE SPLIT LINES resource file statement, 209
LIST resource file statement, 213
OSACCESS resource file statement, 223
PREFIX ORACLE OPEN NAME resource file statement, 225
REUSE SCREEN BUFFERS resource file statement, 231
STATISTICS resource file statement, 237
TERMPOLL resource file statement, 249
TRUNCATE PARM VALUES resource file statement, 253
TRUSTED resource file statement, 254
UIC BRACKETS resource file statement, 256
Off option
SETJOBSHOW resource file statement, 235

OFFoption
  TIME ZONE resource file statement, 251
OLDVALUE function, 433
omnidex program parameter, 137
OMNIDEX resource file statement, 222
ON option
  DBDETACH resource file statement, 197
  DBWAIT resource file statement, 198
  DEFAULT CURSOR OWNER resource file statement, 200
  HPSLAVE EXTRA LINE resource file statement, 208
  HPSLAVE SPLIT LINES resource file statement, 209
  LIST resource file statement, 213
  OSACCESS resource file statement, 223
  PREFIX ORACLE OPEN NAME resource file statement, 225
  REUSE SCREEN BUFFERS resource file statement, 231
  SETJOBSHOW resource file statement, 235
  STATISTICS resource file statement, 237
  TERMPOLL resource file statement, 249
  TIME ZONE resource file statement, 251
  TRUNCATE PARM VALUES resource file statement, 253
  TRUSTED resource file statement, 254
  UIC BRACKETS resource file statement, 256
OPEN option
  DATABASE resource file statement, 193
opening a data dictionary at startup, 202
opening data dictionary at startup, 108
open-name-string general term, 275
OpenVMS
  PowerHouse commands, 31
  QTP error status settings, 25
  QUIZ error status settings, 23
  running PHDPDL, 28
  setting up the PowerHouse environment, 18
OpenVMS program parameters
  checksum710, 82-83
operand
  logical expression, 289
operating system
  calling from PowerHouse, 138
  entering commands, 283
  rules for program parameters syntax, 67
operating systems
  calling from PowerHouse, 223
operators
  logical expressions, 289
  precedence, 296
optimistic locking
  definition, 490
option
  definition, 491
optional (<) metacharacter, 351
optional repeating (*) metacharacter, 351
options
  order entered, 281
  SOUNDEX, 357
OR logical operator
  compound conditions, 282, 296
ORACLE
  database modules specifying owners, 224
  limitations of using synonyms, 325
  using synonyms in PowerHouse, 325
or-bar (|)
  metacharacter, 351
  syntax symbol, 274
OSACCESS predefined condition, 292
osaccess program parameter, 138
OSACCESS resource file statement, 223
output
  conversion, 327
  scaling, 330-333
overflowing statements, 281
overriding
  order of precedence of metacharacters, parentheses, 353
owner option
  OWNER resource file statement, 224
owner program parameter, 139
OWNER resource file statement, 224

P

PACK function, 434
PACKED datatype, 321, 330
  fractional amounts, 330
packing
  strings, 434
page numbers
  returning, 471
PANEL option
  TERMINAL BLOCKMODE resource file statement, 244
  TERMINAL CHARACTERMODE resource file statement, 245
panel option
  blockmode program parameter, 77
  charmode program parameter, 81
parallel driver file
  definition, 491
parameter
  definition, 491
PARAMETER statements
  in message file, 267
parentheses ( )
  changing precedence, 296
parmfile program parameter, 140
parmprompt program parameter, 141
PASSWORD option
  DATABASE resource file statement, 194
patch program parameter, 142
pattern
  definition, 491
  general term, 275
  matching, 351-356, 474
  matching in SQL, 356
  matching, definition, 491
  matching, exact-match characters, 351
  matching, formal syntax, 354
  matching, SQL, 298
patterns
  comparing strings, 424
  detecting invalid, 478
  input conversion, 326
  types, 353
  types, character patterns, 353
  types, date patterns, 354

patterns *(cont'd)*
types, numeric patterns, 353
PDC shared dictionary (OpenVMS), 53
PDL
entering program parameters, 27
exiting, 27
running, 27
PDL shared dictionary
installing, 56
PDL shared dictionary (UNIX), 56
PDL suboption
DICTIONARY resource file statement, 202
percent sign (%)
avoiding conflicting names, 281
percentages
calculations, 329
performance
scaling, 333
period (.)
eliminating, 434
permanent save file
definition, 491
pessimistic locking
definition, 491
PHD dictionaries
using CHECKSUM, 82
PHD dictionaries, using CHECKSUM, 185
PHD shared dictionary (OpenVMS), 55
PHD suboption
DICTIONARY resource file statement, 202
PHDATE datatype, 322
DATEEXTRACT function, 390
PHDPDL, 28
running, 28
PHEDIT environment variable, 138
phonetic code
constructing for string, 357
phonetic codes
creating, 456-457
physical entity
definition, 491
picture
definition, 491
PICTURE option
default scaling factor, 330
plus sign (+)
addition, 300
concatenator, 300
FREEFORM items, 320
pollspeed program parameter, 143
portable subfile
definition, 491
PORTID function, 435
POSITION function, 436
PowerHouse
alternative messages, 262
blob support, 345-346
default files, 48
functions table, 359-365
general terms, 275
non-relational datatypes, 311
prerequisites to running, 17
PowerHouse *(cont'd)*
relational datatypes, 313-316
transaction, definition, 491
using ORACLE synonyms, 325
POWERHOUSE command, 33-34
PowerHouse commands (OpenVMS), 31
PowerHouse functions, 359-481
PowerHouse menu
action field commands, 34
PowerHouse messages, 261
pre_chooseall program parameter, 144
precedence
changing order, 296
conditions, 296
metacharacters in pattern matching, overriding with parentheses, 353
precision
ROUND function, 444-445
Predefined, 282, 293
predefined
conditional compile flags, 282
conditions, 290-293, 295
CURSOROPEN, 291
conditions, transactions, 292
items, 306-307, 347
items, FIELDTEXT, 347
items, SUBPATH, 307
predefined condition
definition, 491
EXISTS, 292
NULL, 292
predefined conditions
EXISTS, 295
IS MISSING, 293
IS NULL, 293
MISSING, 295
NULL, 295
RANGED, 292
RECORD EXISTS, 295
SQLOK, 292
TRANSACTION, 292
predefined conditions, EXISTS, 293
predefined-condition general term, 275
predefined-item general term, 275
predefined-value general term, 275
PREFIX ORACLE OPEN NAME resource file statement, 225
prepare phase
definition, 491, 495
prerequisites
running PowerHouse, 17
primary file
definition, 491
primary index
definition, 491
primary record-structure
definition, 491
printing
preventing extra blank lines, 135
procedural code
SQL using LINKVALUE function, 421
procedural-statement general term, 275

procedure
definition, 492
procedures
FOR loop nesting, 306
PROCEDURES option
LIST statement, 213
process quotas
displaying, 40
processing
blocks of code, 184, 282
operators, precedence, 296
processing code
conditional compile statements, 80
PROCESSLOCATION function, 437
procloc program parameter, 145
returning values, 437
program parameter
definition, 492
program parameters, 67
auto, 75
autodetach, 76
blockmode, 77
broadcast, 78
bulkfetch, 79
cc, 80, 282
charmode, 81
checksum710, 82-83
close_detach, 84
columnowner, 85
commitpoints, 87
compress_buffers, 88
confirmer, 89
consolekeys, 90
createall, 91
createbase, 92
createfile, 93
cursorowner, 94
dbaudit, 96, 96-97
dbdetach, 98
dbwait, 99
dcl, 100
debug (QDESIGN), 101
debug (QUICK), 102
deleteall, 103
deletebase (MPE/iX), 104
deletefile, 105
designer_noretain, 106
detail, 107
dictionary, 108
dicttype (OpenVMS), 109
direct_file_base_zero, 110
disable_nulls, 111
dont_store_module, 112
downshift, 113
entering PDL, 27
entering PHDPDL, 28
entering QDESIGN, 20
entering QSHOW, 29
entering QTP, 24
entering QUICK, 21
entering QUIZ, 22
entering QUTIL, 30
program parameters *(cont'd)*
entering syntax, 67
entryrecall, 114
errlist, 115
fastread, 116
fdl, 117
initnulls, 118
intsize6, 119
jcwbase, 120
lineread, 121
list, 122
lockword, 123
moduleext, 124
moduleloc, 125
no line split (nls), 126
noautodetach, 76
noblobs, 127
nobreakset, 128
noconsolekeys, 90
nodbdetach, 98
nodbwait, 99
nodcl, 100
nodetail, 107
nofdl, 117
noinitnulls, 118
nointsize6, 119
nolist, 122
nonportable, 129
nontermcompat, 130
nontermcompat relationship with Axiant Build Profile setting, 130
noomnidex, 137
noosaccess, 138
noowner, 131
noprefix_ownername, 132
noretainmark, 154
noreuse_screen_buffers, 155
nosetjobshow, 158
nosetwarnstatus, 133
noshift, 113
nostatistics, 159
notermpoll, 165
notimezone, 166
notpi, 167
notrusted, 168
nouicbrackets, 134
nxl (no extra line, 135
obsolete, 136
omnidex, 137
osaccess, 138
owner, 139
parmfile, 140
parmprompt, 141
patch, 142
pollspeed, 143
pre_chooseall, 144
proloc, 145
prompt, 146
qktrace, 147
quotedproccall, 149
read, 150
resetbindvar, 151

program parameters *(cont'd)*
resource, 152
restore, 153
retainmark, 154
reuse_screen_buffers, 155
search, 156
secured, 157
setjobshow, 158
statistics, 159
subdictionary, 160
subformat, 161
term, 162
termpoll, 165
timezone, 166
tpi, 167
trusted, 168
update, 170
upshift, 113
version, 171
vmsdate, 172
project-list general term, 275
prompt
definition, 492
prompt program parameter, 146
PROMPT resource file statement, 226
PROMPTOK predefined condition, 292
punctuation
eliminating, 434

Q

QDESIGN
blobs, 345
entering program parameters, 20
entering statements, 281
exiting, 20
functions, 359-481
running, 20
QKGO files
locating, 43
QKGO files in QUICK
locating, 43
QKGO parameter file
establishing, 75
qktrace program parameter, 147
QSHOW
entering program parameters, 29
exiting, 29
invoking from QDESIGN, QUIZ or QTP, 29
running, 29
QTP
displaying statistics, 25
entering program parameters, 24
error status settings, 24, 25
exiting, 24
reporting update activity, 26
running, 24
testing error status settings, 25
transaction, definition, 492
query-expression general term, 275
query-specification
definition, 492
query-specification general term, 275
question mark (?)
metacharacter, 351
QUICK
displaying null values, 347
entering program parameters, 21
running, 21
screen commands tables, 334
QUICK Debugger
running, 21
QUIT resource file statement, 227
QUIT statement, 20, 21
QUIZ
entering program parameters, 22
error status settings, 22, 23
exiting, 22
running, 22
subformat program parameter, 161
suppressing blank lines in reports, 135
quotedproccall program parameter, 149
QUTIL
entering program parameters, 30
exiting, 30
non-relational datatype mapping, 311
running, 30

R

RANDOM function, 438
seed general term, 438
random numbers
returning, 438
RANGED predefined condition, 292
rapid-fire entry
definition, 492
read program parameter, 150
read/write transaction
definition, 492
read-chain
definition, 492
read-only transaction
definition, 492
record
definition, 492
general term, 275
selective retrieval, 348
testing items for null values, 292
record buffer
definition, 492
returning audit trail status, 371
record complex
definition, 492
RECORD EXISTS predefined condition, 293, 295
record item
definition, 492
record items
testing for missing values, 293, 295
testing for null values, 293, 295
record numbers
zero-based, 110
record/tuple
definition, 492

record-complex general term, 275
record-item general term, 275
RECORDLOCATION function, 439
records
  items, 307
  retrieving with SOUNDEX function, 457
  returning audit trail status, 371
  security, 381
  settings summary status, 293
  status, combinations, 293
record-structure
  definition, 492
  general term, 275
record-structures
  setting assumed, 308
redefining
  null value character, 347
redefinitions
  converting for DISAM, 66
referencing
  arrays, 284
  arrays without subscripts, 285, 286
  occurrences of arrays, 284
related record-structure
  definition, 492
relational database
  definition, 492
  null values support, 347-349
relational datatypes
  PowerHouse, 313-316
  specifics, 316
relationships
  definition, 493
remainders
  returning, 427
REMOVECENTURY function, 440
repeat (>) metacharacter, 351
repeatable options
  syntax symbol, 274
repeating index
  definition, 493
repeating item
  definition, 493
replacing
  leading spaces with zeros, 481
  substitution characters, 462
report
  definition, 493
report security
  definition, 493
report-group
  definition, 493
  general term, 275
report-item
  definition, 493
  general term, 275
report-name general term, 275
reports
  returning current page number, 471
request
  definition, 493
request status
  determining at execution time, 26
required field
  definition, 493
REQUIRED option
  FIELD statement, 349
reserved metacharacters, 352
  changing, 353
RESET BIND VARIABLES resource file statement, 228
resetbindvar program parameter, 151
resource file statements, 173-259
  ALLBASE MODULE EXTENSION, 180
  AUTODETACH, 181
  BROADCAST, 182
  BULKFETCH, 183
  CC, 184
  CHECKSUM710, 185
  CLOSE DETACH, 187
  COLUMNOWNER, 188
  COMMITPOINTS OBSOLETE, 190
  COMPRESS BUFFERS, 191
  CONSOLE KEYS, 192
  DATABASE, 193
  DBAUDIT, 195
  DBDETACH, 197
  DBWAIT, 198
  DEBUG, 199
  DEFAULT CURSOR OWNER, 200
  DESIGNER NORETAIN, 201
  DICTIONARY, 202
  DIRECTORY, 204
  DISABLE NULLS, 205
  ENTRY RECALL, 206
  EXIT, 207
  HPSLAVE EXTRA LINE, 208
  HPSLAVE SPLIT LINES, 209
  INITIALIZE NULLS, 210
  INTEGER SIZE 6, 211
  JCWBASE, 212
  LIST, 213
  LOCATION MODULE, 214
  LOCATION PROCESS, 215
  LOCKWORD, 216
  NOBLOBS, 217
  NONPORTABLE, 218
  NOOMNIDEX, 222
  NOOWNER, 219
  NOSET WARN STATUS, 220
  OBSOLETE, 221
  OMNIDEX, 222
  OSACCESS, 223
  OWNER, 224
  PREFIX ORACLE OPEN NAME, 225
  PROMPT, 226
  QUIT, 227
  RESET BIND VARIABLES, 228
  RESTORE LINES, 229
  RETAIN MARK, 230
  REUSE SCREEN BUFFERS, 231
  RMS FAST READ, 232
  RMS FILE BASE, 233
  SET, 234

resource file statements *(cont'd)*
SETJOBSHOW, 235
SHIFT, 236
STATISTICS, 237
STORE MODULES, 238
SUBDICTIONARY, 239
SUBFORMAT, 240
summary, 173
TERMINAL, 241
TERMINAL BLOCKMODE, 244
TERMINAL CHARACTERMODE, 245
TERMINAL CONFIRMER, 246
TERMINAL POLLING SPEED, 247
TERMINAL READ, 248
TERMPOLL, 249
TIC RESOURCE FILE, 250
TIME ZONE, 251
TPI, 252
TRUNCATE PARM VALUES, 253
TRUSTED, 254
UIC BRACKETS, 256
UPDATE ORDER, 257
USE, 258
VMSDATE, 259
resource files
listing, 234
resource program parameter, 152
RESTORE LINES resource file statement, 229
restore program parameter, 153
restrictions
blobs treated as text items, 346
restructuring
messages, 271
results
comparison with null values, 297
RETAIN MARK resource file statement, 230
retainmark program parameter, 154
retrieving
data records with SOUNDEX function, 457
DISAM, 65
REUSE SCREEN BUFFERS resource file statement, 231
reuse_screen_buffers program parameter, 155
REVERSE function, 441
reversing
character strings, 441
REVISE statement
default list option, 213
list program parameter, 122
nolist program parameter, 122
RIGHT JUSTIFY|RJ function, 442
right parenthesis ()) metacharacter, 351
right-justification
characters in strings, 442
RMS FAST READ resource file statement, 232
RMS FILE BASE resource file statement, 233
rollback
definition, 493
rolling back transactions, 292
ROUND function, 443-445
rounded numbers
returning, 443-445
rounding
integer values downwards, 406
integer values upwards, 374
row
definition, 493
general term, 275
rules
entering program parameters, 67
exceptions, null values, 348
for entering conditional compile statements, 282
run
definition, 493
running
PDL, 27
PHDPDL, 28
QDESIGN, 20
QSHOW, 29
QTP, 24
QUICK, 21
QUICK Debugger, 21
QUIZ, 22
QUTIL, 30
running PowerHouse
prerequisites, 17
runtime message format
service layer, 270

S

save file
definition, 493
SAVE statement
locating files, 41
scaling
decimal alignment, 330-333
efficiency, 333
input, effect of calculations, 331
input, fractional amounts, 330
output, 330-333
stored values, 331
scaling factor
numeric items, default, 330
SCREENLEVEL function, 446
screen-name general term, 275
screens
establishing, 75
level number, 446
SEARCH option
SUBDICTIONARY resource file statement, 239
search option
subdictionary program parameter, 160
search program parameter, 156
searching
items, 308
SECOND extract-option
EXTRACT function, 404
SECOND general term
DATEEXTRACT function, 390
secondary record-structure
definition, 493
seconds general term, 275
secured program parameter, 157

security
CHECKSUM function, 380
data records, 381
definition, 493
encryption, 402
returning logonid, 422
seed general term
RANDOM function, 438
segment
definition, 493
segment general term, 275
SELECTMODE predefined condition, 291
semicolon (;)
eliminating, 434
entering comments, 281
sequence
processing in pattern matching, 353
service layer message file
creating alternative, 269
service layer messages, 261, 265, 265-270
runtime format, 270
service message compiler, 268
SET resource file statement, 234
SET statement
ASSUMED option, 308
SET SUBFILE statement, 161, 240
SETDICTIONARY command session, 36
setjobshow program parameter, 158
SETJOBSHOW resource file statement, 235
SETSYSTEMVAL function, 447-450
setting up the PowerHouse environment
on MPE/iX, 18
on OpenVMS, 18
on UNIX, 18
on Windows, 19
SEVERITY statement
service layer message file, 266
shared memory management, 58
shared memory sections
deleting, 57
shell
accessing from PowerHouse, 138, 223
entering commands, 283
SHIFT resource file statement, 236
SHIFTLEVEL function, 451
SHOWDICTIONARY command, 38
SHOWPOWERHOUSE command, 39
SHOWQUOTA command, 40
SIGNED option
ZONED datatype, 323
significance
output conversion, 327
significant digits
representing floating point numbers, 318
SIGNONACCOUNT function, 452
SIGNONGROUP function, 453
SIGNONUSER function, 454
simple conditions, 295
modifying, 296
single
conditional compilation, 282
sitehook, 63
six-digit dates
converting eight-digit, 368
mixing with eight-digit dates, 368, 440
SIZE function, 455
sizes
blobs, 345
elements, 310
items, 310
logical, 310
returning string-expression, 455
storage, 310
slash (/)
division, 300
slave screen
definition, 493
sort-item
definition, 493
general term, 275
SOUNDEX function, 456-457
SOUNDEX option, 357
source file
definition, 493
SOURCE option
DEBUG resource file statement, 199
source option
debug program parameter, 102
source statements
establishing, 75
listing, 122, 213
spaces
adding between characters, 458
eliminating, 434
statements, 281
special character
definition, 493
special characters
ampersand (&), continuation character, 281, 282
arithmetic operators in numeric expressions, 300
at-sign (@) in conditional compile statements, 282
at-sign (@), continuation character, 282
operator in conditions, 289, 298
parentheses ( ), order of precedence, 296
percent sign (%), avoiding conflicting names, 281
semicolons (;), entering comments, 281
syntax, 273
specifying, 401
items addressed as character strings, 415, 416
specifying owners
ALLBASE/SQL database modules, 224
ORACLE database modules, 224
SYBASE database modules, 224
unqualified table name, 139
unqualified table names, 224
splitting
conditional compile statements, 282
statements, 281
SPREAD function, 458
SQL
conditions, 298-299
expressions, 303
NULL condition, 348
pattern matching, 356

SQL *(cont'd)*
procedural code using LINKVALUE function, 421
SQL option
LIST resource file statement, 213
SQLCODE
system function, 459
sql-conditions
general term, 275
HAVING option, query-specification, 298
WHERE option, query-specification, 298
sql-datatype general term, 275
sql-date-expression general term, 275
sql-date-literals
DATE, 303
INTERVAL, 303
TIME, 303
TIMESTAMP, 303
sql-expression general term, 275
SQLMESSAGE
system function, 461
sql-numeric-expression general term, 275
SQLOK predefined condition, 292
sql-operator general term, 275
sql-substitution general term, 275
sql-substitution-variable general term, 275
sql-summary-operations general term, 275
sql-syntax general term, 275
square brackets ([ ])
syntax symbol, 273
stacked syntax, 274
stacking
mutually exclusive options in syntax, 274
start screens
locating, 43
starting
position of substrings in strings, 414
statement
definition, 493
statements, 282
continuing, 281
EDIT, 286
entering, 281
EXECUTE, 42
EXIT, 21
GO locating compiled screens, 44
overflowing, 281
QUIT, 20, 21
splitting, 281
syntax, help, 17
USE, 42
statistics
displaying, 237
displaying in QTP, 25
statistics program parameter, 159
STATISTICS resource file statement, 237
status
audit trail, 371
of a request, 26
predefined conditions, Entry mode, 290
predefined conditions, Find mode, 290
returning code, 383
stopscreen
definition, 493
storage
conversion of values, 326, 326-329
forms of data, 309
size, 310
storage option
definition, 494
STORE MODULES resource file statement, 238
storing
binary numbers, 345
FREEFORM values, 320
numeric items, 331
string
definition, 494
general term, 275
string expression general term
INDEX function, 414
string expressions, 303
string-expression general term, 275
strings
appending with || in SQL, 303
appending with plus sign (+), 300
assigning to blobs, 345
centering characters, 375
character, converting numbers, 369
comparing to patterns, 424
concatenating, 300, 303
constructing phonetic code, 357
creating phonetic codes, 456-457
eliminating punctuation, 434
expressions, 300
extracting substrings, 465
finding substrings, 414
inverting, 441
left-justified, 420
operands in conditions, 289
packing, 434
removing trailing blanks, 474
returning checksums, 380
returning size, 455
reversing characters, 441
right-justified character strings, 442
subdict,*See*subdictionary
subdictionary program parameter, 160
SUBDICTIONARY resource file statement, 239
subfile, 161, 240
definition, 494
null values, 347
subfile format, 161
subfiles
locating, 44
subfilespec general term, 275
subformat program parameter, 161
SUBFORMAT resource file statement, 240
subordinate record-structure
definition, 494
SUBPATH
predefined item, 307
subquery
definition, 494
general term, 275

subquery *(cont'd)*
general term, SQL, 298, 299
SUBSCREEN statement
procloc program parameter, 145
subscript
definition, 494
general term, 275
subscripting items, 285
SUBSTITUTE function, 462-463
substitution character
definition, 494
in message files, 267, 271
substitution characters
in message files, 264
substitution order
changing, 271
substitutions
replacing characters, 462
substitution-variable
definition, 494
general term, 275
SUBSTRING EXTRACT function, 465
SUBSTRING function, 464
substrings
extracting from strings, 465
locating within strings, 414
subtraction
minus sign (-), 300
SUM
sql-summary-operation, 305
SUM function, 466
summary
floating point types, 318
summary information
definition, 494
summary operations
SQL, 304
summary report
definition, 494
summing
entire arrays, 466
intermittent occurrences of arrays, 467
items in arrays, 466
subsets of arrays, 467
support
blobs, 345-346
suppressing
extra blank lines in QUIZ reports, 135
SYBASE
database modules specifying owners, 224
symbols
syntax, 273-274
syntax
abbreviating keywords, 281
definition, 494
explicit exclusion, 349
help, 17
indented, 274
obsolete messages, 136, 221, 239
program parameters, 67
special characters, 273
stacked, 274
syntax *(cont'd)*
symbols, 273-274
SYSDATE function, 468
SYSDATETIME function, 469
SYSNAME function, 470
SYSPAGE function, 471
system function
definition, 494
OCCURRENCE, 284
operands in conditions, 289
SQLCODE, 459
SQLMESSAGE, 461
system-function general term, 275
system-wide standards
definition, 494
SYSTIME function, 472

T

table/relation
definition, 494
table-name general term, 275
tables
item datatype defaults, 310
PowerHouse functions, 359-365
User commands, 342
tablespec general term, 275
templates
messages, 263
temporary item, 308
definition, 494
missing values, 293
null values, 293, 295, 347
temporary items
missing values, 295
null values, 295
temporary save file
definition, 494
temporary-item general term, 275
term
definition, 495
general term, 275
program parameter, 162
TERMINAL BLOCKMODE resource file statement, 244
TERMINAL CHARACTERMODE resource file statement, 245
Terminal Compatible property, 130
TERMINAL CONFIRMER resource file statemen, 246
TERMINAL POLLING SPEED resource file statement, 247
TERMINAL READ resource file statement, 248
TERMINAL resource file statement, 241
terminal-parameter option
term program parameter, 162
terminals
identifying, 435
returning type, 473
type when running QUICK, 162
types when running QUICK, 241
terminal-type option
term program parameter, 162
terminology, 275
termpoll program parameter, 165

TERMPOLL resource file statement, 249
TERMTYPE function, 473
termtype general term, 275
testing
  error status settings in QTP, 25
  error status settings in QUIZ, 23
  locally active transactions, 292
  missing values, 292
  null values, 292, 293, 295
  status of output, 293
  stored values, 331
  values, predefined conditional compilation, 282
testing for null values, 348
text
  centering, 375
text file
  definition, 495
text order numbers and substitution characters, 271
TIC RESOURCE FILE resource file statement, 250
TIME
  sql-date-literal, 303
TIME datatype, 316
TIME general term
  DATEEXTRACT function, 390
TIME ZONE resource file statement, 251
times
  returning current, 472
TIMESTAMP
  sql-date-literal, 303
timezone program parameter, 166
  equivalent resource file statement, 166
titles
  returning current dictionary, 470
TOP DOWN option
  UPDATE resource file statement, 257
topdown option
  update program parameter, 170
tpi program parameter, 167
TPI resource file statement, 252
  See also OMNIDEX resource file statement
trace files
  naming, 147
trailing
  blanks, 474
  sign, input conversion, 326
  sign, output conversion, 327
  spaces, FREEFORM items, 320
trailing blanks
  removing from strings, 474
TRANSACTION, 292
transaction
  definition, 495
  general term, 275
transaction commit
  definition, 495
TRANSACTION IS ACTIVE predefined condition, 292
TRANSACTION IS INACTIVE
  predefined condition, 292
TRANSACTION IS LOCALLY ACTIVE
  predefined condition, 292
TRANSACTION option
  LIST resource file statement, 213
transaction rollback
  definitions, 495
transaction set
  definition, 495
transaction start
  definition, 495
transactions
  committing, 292
  predefined conditions, 292
  rolling back, 292
TRUE
  results, 297
TRUNCATE function, 474
TRUNCATE PARM VALUES resource file statement, 253
truncating
  numeric items, 331
trusted program parameter, 168
TRUSTED resource file statement, 254
truth tables
  boolean operations, 297
two-phase commit
  definition, 495
type general term, 275
TYPE option
  DICTIONARY resource file statement, 202
type-option general term, 275

U

UIC BRACKETS resource file statement, 256
UIC function, 475
unauthorized access
  preventing, 402
underlining
  examples of functions, 359
uninstalled dictionaries
  accessing, 57
unique index
  definition, 495
UNIX
  QTP error status settings, 24
  QUIZ error status settings, 22
  setting up the PowerHouse environment, 18
UNSIGNED option
  ZONED datatype, 323
UP option
  ROUND function, 443
  SHIFT resource file statement, 236
update activity in QTP, 26
UPDATE ORDER resource file statement, 257
update program parameter, 170
UPPER function, 476
uppercase
  characters in syntax, 273
  specifying, 477
uppercase characters
  shifting to lowercase, 401
UPSHIFT function, 477
upshift program parameter, 113
usage
  definition, 495
  general term, 275

usage general term, 275
use file
  definition, 496
USE option
  LIST resource file statement, 213
USE resource file statement, 258
USE statement
  default list option, 213
  list program parameter, 122
  locating files, 42
  nolist program parameter, 122
  procloc program parameter, 145
User, 324
user
  commands tables, 342
user break
  definition, 496
user mode
  definition, 496
user-defined
  datatypes, SYBASE SQL Server, 324
USERID option
  DATABASE resource file statement, 194
username
  returning, 454
usernames
  returning, 422

V

VALIDPATTERN function, 289, 478
value
  definition, 496
  general term, 275
values
  allowed, input conversion, 326
  monetary, specifying decimal currencies, 328
  negative, displaying, 329
  returning absolute, 367
  returning NULL, 426, 429
  returning using the procloc program parameter, 437
  returning, buffers, 433
  ROUND function, 443-445
  rounding down, 406
  rounding up, 374
VALUES option
  FIELD statement, input scaling, 331
value-set general term, 275
VARCHAR datatype, 322
verbs
  DO BLOB, 345
version of document, 2
version program parameter, 171
versioning
  definition, 496
view
  definition, 496
VMSDATE datatype, 322
vmsdate program parameter, 172
VMSDATE resource file statement, 259
VMSTIMESTAMP function, 479

W

warn option
  debug program parameter, 102
  nonportable program parameter, 129
  obsolete program parameter, 136
warning messages
  alternative, 264
  confirmation by user, 89
  locating, 261-262
  nonportable, 129
  returning text, 384
WARNING option
  DEBUG resource file statement, 199
  NONPORTABLE resource file statement, 218
  OBSOLETE resource file statement, 221
WEBLOGONID function, 480
wild (@) metacharacter, 351
Windows
  QTP error status settings, 24
  QUIZ error status settings, 22
  setting up the PowerHouse environment, 19

Y

YEAR extract-option
  EXTRACT function, 404
YEAR general term
  DATEEXTRACT function, 390
yy general term, 275
yyyy general term, 275

Z

ZDATE datatype, 323
ZERO option
  ROUND function, 443
zero-based
  record numbers, 110
ZEROFILL function, 481
zeros
  dividing by, 300
  replacing leading spaces, 481
ZONED datatype, 323, 330
  fractional amounts, 330