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

VERSION 8.4E

QDESIGN REFERENCE

COGNOS®

THE NEXT LEVEL OF PERFORMANCE™

Product Information

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

Copyright

Table of Contents

About this Book

Overview

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

Chapter 1, "Introducing QDESIGN and QUICK", introduces QDESIGN, QUICK, and the other PowerHouse components and utilities.

Chapter 2, "QUICK User Interface", provides information about the QUICK user interface, including Panel input mode, partial-index retrieval in QUICK, and multiple command processing. All QUICK function keys are discussed.

Chapter 3, "QDESIGN Statements", provides concise summaries and detailed information about QDESIGN statements. Syntax summaries, detailed syntax discussions, and examples are provided for each QDESIGN statement, where applicable.

Chapter 4, "QDESIGN Procedures Overview", introduces you to QDESIGN's default and supplementary procedures.

Chapter 5, "QUICK's Processing Modes", describes how QDESIGN's procedures are called by QUICK screen user actions. QUICK's three basic modes of operation (Entry, Find, and Select) are discussed.

Chapter 6, "Customizing QUICK with QKGO", describes the QKGO utility. With QKGO, you can customize QUICK's operating characteristics.

Chapter 7, "QDESIGN Procedures", provides concise summaries and detailed information about QDESIGN procedures. Syntax summaries, detailed syntax discussions, and examples are included, where applicable.

Chapter 8, "QDESIGN Verbs and Control Structures", discusses verbs and control structures that you can use to control processing in QDESIGN procedures.

Chapter 9, "Debugger", provides information about using the QUICK Interactive Debugger to analyze and control QUICK screens as they run.

Chapter 10, "Debugger Commands", provides concise summaries and detailed information about Debugger commands. Syntax summaries, detailed syntax discussions, and examples are provided for each Debugger command, where applicable.

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, refer to the *Release and Install Notes*.

Chapter 1: Introducing QDESIGN and QUICK

Overview

This chapter introduces QDESIGN and QUICK and provides an overview of the other PowerHouse components and utilities.

About PowerHouse

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

Components

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

PowerHouse Dictionary

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

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

For more information about the PHD dictionary, see the *PHD Reference* and *PowerHouse Rules books*. See also the section, "PowerHouse Dictionary on OpenVMS", in Chapter 1, "Introducing the PowerHouse Dictionary", in the *PDL Reference*.

PDL

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

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

PDL Compiler

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

PHDPDL Compiler (OpenVMS)

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

PHD Screen System (OpenVMS)

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

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

QDESIGN and QUICK

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

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

QUIZ

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

QTP

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

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

Utilities

PowerHouse also contains the following data dictionary utilities:

QSHOW

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

QUTIL

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

ITOP (MPE/iX)

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

QCOBLIB (MPE/iX)

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

PHDMAINTENANCE (OpenVMS)

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

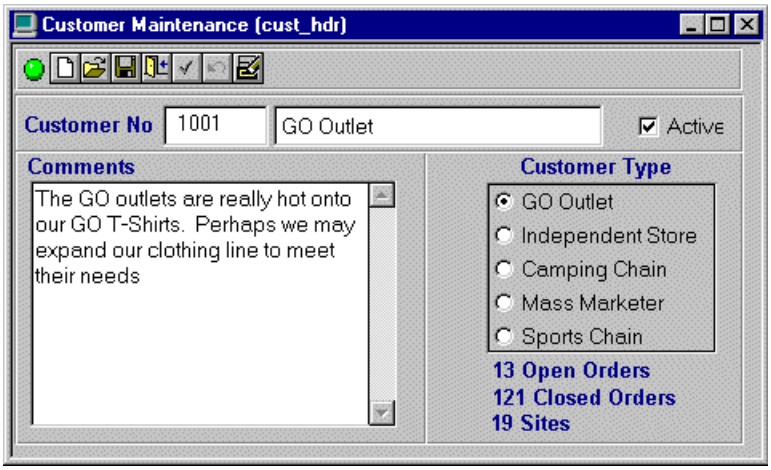
PHDADMIN (OpenVMS)

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

PowerHouse-Related Products

Axiant 4GL

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



PowerHouse Web

Introducing QDESIGN

QDESIGN is the PowerHouse screen development component. Screen designers use QDESIGN statements and procedures to create QUICK screens.

Like the other PowerHouse components, QDESIGN is tied closely to the PowerHouse dictionary. QDESIGN uses the data definitions stored in the PowerHouse dictionary when generating screens for use in QUICK.

Unlike third generation languages, QDESIGN handles a large amount of background procedural processing automatically. In effect, QDESIGN can be split into two distinct yet closely tied layers:

- the Design layer is made up of QDESIGN statements. Statements in the Design layer are entered primarily by the screen designer, although many can be generated automatically by QDESIGN itself. The statements in the Design layer influence the procedures that QDESIGN generates in the Procedural layer.
- the Procedural layer is made up of QDESIGN procedures. Procedures in the Procedural layer are either generated automatically (based on the statements in the Design layer) or are explicitly coded by the screen designer.

All QUICK screens incorporate both the Design and the Procedural layers. Whenever possible, you should restrict your coding efforts to the Design layer, and allow QDESIGN to handle the procedural layer for you automatically. You should code procedures only when a desired result cannot be achieved with QDESIGN statements.

The Sections of a QDESIGN Screen Design

The statements that you use in the Design layer fall into three categories:

The Screen Section

Controls how QUICK actions or menus are presented in a menu bar that extends across the terminal screen.

Note: On Windows, QUICK runs in a Console window, whereas on other platforms, QUICK runs on a terminal or terminal emulator. For simplicity, the word 'terminal' will be used throughout this book.

The Data Section

Defines and controls how the QUICK screen processes data, including which files and record-structures are accessed, as well as how they are accessed. In addition, the data section controls how items are initialized.

The Layout Section

Controls how screen entities such as fields, string literals, and lines appear on QUICK screens.

The following table lists the statements that fall into QDESIGN screen, data, and layout sections:

Screen Section	Data Section	Layout Section
ACTIONMENU	ACCESS	ALIGN
DESCRIPTION	CURSOR	CLUSTER
KEY	[SQL]DECLARE CURSOR (query specification)[1]	COMMAND
MENUITEM	[SQL]DECLARE CURSOR (stored procedure)[1]	DRAW
SCREEN	DEFINE	FIELD
	FILE	GENERATE
	ITEM	HILITE
	query-specification (SELECT)	SKIP
	SELECT	SUBSCREEN
	TARGET	THREAD
	TEMPORARY	TITLE
	TRANSACTION	

[1] *This statement can precede the SCREEN statement.*

The section statements must be entered in the following order:

1. screen
2. data
3. layout

The remaining QDESIGN statements fall into none of the three sections. They control how QDESIGN itself works rather than operational characteristics of the QUICK screens that are being developed. These statements include BUILD, CANCEL, EXIT, GO, QSHOW, QUIT, REVISE, SAVE, SET, SHOW, and USE.

For details about QDESIGN statements and how they work, see Chapter 3, "QDESIGN Statements".

Introducing QUICK

Screens that have been created with QDESIGN are run with QUICK, the PowerHouse screen processing component. Screen designers can use QUICK and the Interactive Debugger to test the screens and prototypes that they've developed in QDESIGN.

For information about QUICK screen use, see Chapter 2, "QUICK User Interface".

For information about QUICK processing, see Chapter 5, "QUICK's Processing Modes".

For information about the Interactive Debugger, see Chapter 9, "Debugger" and Chapter 10, "Debugger Commands".

Chapter 2: QUICK User Interface

Overview

This chapter discusses how the QUICK user interface works. It includes information about

- using QUICK screens
- QKView (Windows)
- input modes
- multiple command processing
- partial-index retrieval
- scrolling primary and detail records
- screen threads

Using QUICK Screens

This section describes how to use the standard QUICK screen commands in Field mode. For details on how to use Panel mode and Compatible Block mode, see "Input Modes" (p. 37). For details on screen threads, see "Screen Threads" (p. 60).

For lists of QUICK commands, see "QUICK Commands" (p. 285) and "QUICK Screen Commands" in Chapter 5, *PowerHouse Rules*.

In any particular application, you can add, change, or remove commands. As well, there may be alternative commands that are specific to the terminal you are using. You can get a list of these alternative commands using the QKGO system or by looking at the TIC files. For more information, see "The Terminal Interface Configuration Screen" (p. 278) and "Modifying TIC Files" (p. 283).

Most input is completed using the Enter or Return key, depending on the terminal or terminal emulator you're using. In this section, where Enter is used, it stands for either the Enter or Return key as is required in your specific case. Using the Enter key is also the default for the Accept and Input Completion commands.

Menu Screens and Data Screens

There are two types of QUICK screens: menu screens and data screens. Most applications begin with a menu screen that lists a number of screens, commands, and programs. Complex applications may have several menu screens, each acting as an introduction to a new part of the application.

Data screens are used to enter and retrieve the data in your application. Entering data on a data screen is like filling out a form. The fields on the data screen correspond to the blanks on the form. Most data fields are identified by an ID-number and a label.

Data screens can also provide access to other screens, commands, and programs.

QUICK Screen Commands

When you first access a screen, you are usually prompted in the Action field, which often appears in the top left-hand corner of a screen. You must enter an Action command to process the data on your screen or to move to another screen. The default command mnemonic is shown in parentheses after the command name. You can use the QKGO system to change the default mnemonic.

You can move, remove, or change the name of the Action or Mode field.

When you first invoke a data screen, you must choose a mode, unless the startup mode has been set by the calling screen. There are three modes you can use in QUICK: Entry mode, Find mode, and Select mode. To choose a mode, enter the appropriate mode command: Entry (E), Find (F), or Select (S).

When you specify a mode, it is displayed in the Mode field as E, F, or S. The Mode field normally appears to the left of the Action field. However, you can move or remove the Mode field.

Action Commands

Action commands are commands that apply to a whole screen. These commands

- set screen processing modes to Enter, Find, or Select
- process data
- move you through a system of screens
- show you help information about a screen
- invoke application specific "designer" commands
- select items on the screen by their ID-number
- perform screen actions such as starting data entry or updating data

Data Commands

Data commands are commands that control data entry. You enter data commands in data fields. These commands let you

- move between data entry fields
- complete or back out of data entry
- request information and help

Using Function Keys to Enter Commands

PowerHouse supports function keys on some terminals and the equivalent emulators as well as in QUICK for Windows.

Action and data commands are assigned or "mapped" to one or more function keys by default. Simply press the appropriate function key to enter the command you want to use. See "The Terminal Interface Configuration Screen" (p. 278) for more information on using the QKGO system to list the function key mappings. You can also use the QKGO system to change the configuration or map other commands to keys and key sequences.

Moving from Screen to Screen

You can move from one screen to another by entering the ID-number of the screen you want in the Action field. To return to the previous screen, use the Return command (^). If you're at the first screen in your application, this command takes you out of the application.

Stopscreens

Some screens, usually menus, are stopscreens. A screen system designed with stopscreens allows you to move up several levels in a screen hierarchy with the Return to Stop command (^^). This command moves you up the screen hierarchy until you hit a stopscreen. If there are no stopscreens above you in the system, the command takes you out of the application.

Moving to the First Screen

You may also make the Return to Start command (^^^) available. This command allows you to return directly to the first screen in the system from any other screen in the system. Any stopscreens are bypassed.

Entering Data

To add new data to your files and tables, move to the correct screen and use the Enter command (E). QUICK automatically starts prompting you to enter data in the first data field on your screen.

To enter a value into a data field, type in the data then press the Enter key. Enter is the default Input Completion or Accept command. Any time you make an entry QUICK can't accept, a message is displayed and you're prompted for another entry. When QUICK accepts your entry, you are prompted to enter data in the next field.

Moving From Field to Field During Data Entry

When you enter data in a data entry screen, it's easy to move from field to field. When you press the Enter key in a field, QUICK takes you to the next field. In most cases, QUICK lets you skip to the next field even if you don't enter a value.

You can add default values that appear in the data fields that you skip.

To skip a field, press the Enter key without entering any data in the data field. However, some fields require you to enter data in them before you can continue with the rest of the screen.

You can also enter the following commands to move from field to field when you're in Entry mode. These commands produce the same effect as entering data in each field along the way. PowerHouse fills in defaults for the fields that you skip, and won't let you skip required fields.

Skip All (??)

Prompts you for an action after you have made entries in all required fields. Otherwise, this command takes you to every required field on the current screen.

Skip Cluster (/)

Takes you forward to the first field of the next group.

Skip to a field ID-number (/n)

Takes you forward to the specified field. If there are any required fields before the field, it takes you to the required field first. You can't use this command to back up.

Backup (\)

Backs you up to the previous field.

Full Field Processing

You can use a QKGO option so that you are notified audibly if you enter data that exceeds the field size. You can then edit or delete the data in the field.

You can use a FIELD statement option to specify that you should be prompted in the next field automatically once you enter data that fills the field.

Entering Numbers

QUICK automatically displays numbers according to the specifications you build into your application. You can enter numbers without worrying about the format, so

- don't enter dollar signs and commas
- don't enter leading or trailing zeros
- do enter negative numbers with a minus sign before or after the number
- do enter fractional values (if allowed) with a decimal character (usually a period)
- enter one zero for a value of zero

For example, if you enter

```
3112.7
```

QUICK might redisplay the number as

```
$3,112.70
```

Entering Dates

The format you use to enter dates is determined by the specifications you build into your application. The month format is either three characters, i.e. JAN, or two digits, i.e. 01. You can enter dates with or without a separator character. The separator character usually used to separate the parts of a date is a slash (/). You may or may not be required to enter dates with the century, and based on the application specifications, the date may be displayed with or without a century. For example, if you enter

```
070214
```

the date might be displayed as

```
2007/02/14
```

Your specifications may require another character or a space as a separator character. If you enter the Help command (?) in the field, QUICK displays a help message with information on the format.

In many applications, entering a single zero in a date field is used to signify that no date exists. Do not enter 6 or 8 zeros as that is an invalid date.

Entering Null Values

To enter a null value into a relational column, use the Enter Null Value command (~). In most cases, columns where null values are allowed are initialized to NULL and a specific entry is not required. Simply skip the field. The Enter Null Value command is used to change a non-null value to a null value.

Rapid-Fire Entry

You can make more than one entry in a single field by separating each entry with the Separator command (;). When you press the Enter key to complete the concatenated entry, QUICK accepts the entries as if each was entered separately. When you use rapid-fire entry to skip screens, the intervening screens aren't displayed.

The Duplicate Command

Entering the Duplicate command (_) duplicates the last entry in the field. This command works even if the last entry in the field was made on the previous screenload of entries.

Completing the Entry Process

The entry process leads you through all the fields on a screen. When you complete the last field, or use the Skip commands, you are prompted for the next action. You can now make corrections, if necessary.

Changing Data During Data Entry

When you complete the entry process, QUICK prompts you in the Action field to either save the data on file with an Update command or make any corrections before you update.

To correct a mistake, enter the field ID-number of the field you want to correct and reenter the correct data when you're prompted in the field. Once you've entered a value, QUICK again prompts you in the Action field.

If many changes are required, you may be able to correct several fields at once by entering a range of ID-numbers, for example 5/9. This lets you move through fields 5, 6, 7, 8, and 9 to make the necessary corrections before QUICK prompts you for your next Action command.

Some fields don't have their own ID-numbers, but share an ID-number with a previous field. To reach any one of these fields, enter the shared ID-number. QUICK prompts you through each of the fields that share that ID-number before prompting you for your next Action command.

If you don't enter a value when you are prompted in a field, QUICK leaves the current value unchanged.

You can also delete an entry, or all the entries on the screen, while making corrections. For more information on the Delete command, see "Deleting Data" (p. 26).

Recalling Data

To edit the data in a field, enter the Recall command (<Ctrl-B>) to return the last value you entered. When you recall the data, the cursor remains at the end of the data.

Saving or Updating Data

When you enter or change data in a screen, you are actually modifying a copy of the original values. The information in the original file or table is not changed until you enter an update command.

If you try to leave a screen or change modes without updating, QUICK prompts you for confirmation. Repeating the command tells QUICK that you do not want to save any data entered since the last update.

The Update Commands

All of the update commands store all the data that is on the screen. The update commands differ in what they do after the data has been stored.

You can specify that an update command run additional checks on the data. This ensures that the data is accurate and complete. If there are errors in your data, the application will not allow the update process to continue. You must correct the errors in your data and try to update again.

Update (U)

Stores the data, then clears the fields for more entries (in Entry mode), or retrieves the next screenload of data (in Find mode).

Update Next (UN)

Stores the data, then clears the fields for more entries (in Entry mode), or retrieves the next screenload of data (in Find mode). The Update Next command is used on screens with repeating fields.

Update Stay (US)

Stores the data, then keeps the current screenload of data on the screen.

Update Return (UR)

Stores the data, then returns you to the preceding screen.

Updating with Repeating Fields

The Update and Update Next commands function differently with screens that have repeating groups of fields. These screens usually have two types of data. For example, a screen may have two different sections: the first section applies to an employee; the second allows you to enter data about a number of expenses for that employee. The second section is organized around repeating groups of fields. For example, each group might consist of the fields for an expense, an amount, and a date.

On this type of screen, you enter repeating groups while the first section of the screen stays the same. With a screen designed this way, the Update command saves all the data on the screen, but clears only the repeating groups. In Entry mode, you are then prompted to enter another set of the repeating groups. In Find mode, you are shown the next set of repeating groups.

If you have not entered data in all of the repeating groups, Update will save the data in all of the fields and prompt you for data in the first, non-repeating, section, for example, for a new employee. If you've filled all of the repeating groups and you want to start with new data in the first section, use the Update Next command. This command saves all the data and always prompts you to enter data in the first section, whether you have filled in all of the repeating groups or not.

Finding Data

Find mode gives you fast retrieval using key and index values. To choose Find mode, enter the Find command (F) in the Action field, and QUICK will prompt you for a value in the first index field. If you enter a value in the field, PowerHouse retrieves all data for that value. If you press the Enter key without entering a value, PowerHouse prompts you for a value in the next index field, if there is one.

Instead of finding a specific data record, you may want to browse through all of them. If you don't enter a value for any key or index, QUICK displays the first screenload of data it finds and prompts you for an action.

The Next Data command, pressing the Enter key, lets you see the next screenload of data. After the last screenload of data has been displayed, PowerHouse clears the fields and prompts you for an action. You can stop the process at any time by entering another Action command (such as Enter or Find) instead of the Next Data command.

Selecting Data

When you're finding data, PowerHouse prompts you for values at index fields. However, you may want to be more selective and retrieve data using different non-index values.

To select data:

1. Enter the Select command (S) in the Action field.
2. Enter key or index values as you do when finding data. When prompting for key or index values is completed, QUICK prompts you for an action.
3. Enter a field ID-number to establish your selection criteria. QUICK prompts you for a value in that field.
4. Enter the value.
5. Enter other selection values by choosing field ID-numbers and entering values.
6. When you've finished entering values, enter the Next Data command in the Action field to begin retrieving data based on your selection criteria.

Retrieving Data by Partial Index

When you select data, you can also retrieve information based on a pattern that describes certain values.

When you're finding or selecting data, you can use the generic retrieval character (@) to retrieve data based on part of a value. This is called partial-index retrieval. For example, if you want to see data records for just those employees whose surnames begin with M, enter M@ in the LastName field. If you enter M@@, PowerHouse finds data records for all surnames that start with the letter M and continues through the alphabet to find all surnames up to, and including, those that start with the letter Z.

The generic retrieval character must always be the last character in partial-index retrievals.

Retrieval using the generic retrieval character is only available for character fields. This feature does not work with numeric or date type items.

Pattern Matching

When you're selecting data, you can use the generic retrieval character (@) and several other characters to create patterns that indicate selection values. Patterns are made up of the following special characters, called metacharacters:

- ^ matches Any single letter.
- # matches Any single digit.
- ? matches Any single letter, digit, or other character.
- @ matches Zero or more characters (letters, digits, or anything else).
- !0 matches A null entry (nothing).

To indicate that the value is a pattern, precede the value with a percent sign (%). For example, if you enter

```
%^^#
```

in a field, QUICK retrieves all data for which the values in this field match the pattern

```
^^#
```

such as Ab2, pd3, and cy4.

The question mark (?) represents any single letter, digit or other character. For the entry

```
%^^#?
```

QUICK finds values such as Fk83, So2b, CY7K, etc. All ordinary characters (for example, the alphabet) and numbers match themselves in patterns.

You can use the at-sign (@) in pattern matching anywhere in your pattern. For example, entering

```
%@TH
```

retrieves all values ending in "th", "TH", "Th", or "tH".

You can specify that uppercase and lowercase letters match letters in the same case as those in the pattern. Case-sensitive pattern matching is accomplished by entering %% to start the pattern. The pattern

```
%%@th
```

matches all values ending in "th", but not values ending in "TH", "Th", or "tH".

For more information on pattern matching, see "Pattern Matching in PowerHouse" in Chapter 5, *PowerHouse Rules*.

Getting More Data

Next Data (by default the Enter key)

QUICK may retrieve more than one screenload of data when you use Find or Select mode. Each time you press the Enter key a new screenload of data is retrieved. You can see all the screenloads by pressing the Enter key until all the data has been retrieved.

Next (N)

On screens with repeating groups of fields, pressing the Return key retrieves a new screenload of repeating groups, before going on to the next screenload of all new data. The Next command lets you skip directly to the next screenload of all new data.

Previous Data (\)

Depending on the file system used, you may be able to move backwards to the previous screenload of data using the Previous Data command.

Scrolling Records

If you have a screen with repeating groups of fields, set up as scrolling records, you may be able to move the data display back and forth using the First Record (FR), Last record (LR), Next Record (NR), and Previous Record (PR) commands. For more information, see "Scrolling Primary and Detail Records" (p. 57).

Interrupting Retrieval

In Find and Select modes, you can stop the retrieval process while you're waiting for a screenload of data to appear. Type the User Break command to stop the retrieval process. The User Break command is <Ctrl-Y> on MPE/iX and <Ctrl-C> on OpenVMS, UNIX, and Windows.

Changing Data in Find or Select Mode

You can change data only when it's on the screen. If the data isn't on your screen you must first use Find or Select mode to retrieve it.

You can change data in a single field by following this procedure:

1. Enter a field ID-number in the Action field. QUICK prompts you in the field.
2. Enter a new value, or edit the existing value.
3. Press the Enter key to complete the correction. QUICK returns you to the Action field.

You can correct more than one field at a time by entering a range of ID-numbers in the Action field, as in

```
5/9
```

When you press the Enter key, QUICK takes you to the first field of the specified range. In this example it would be field 5. Every time you press the Enter key, QUICK takes you to the next field in the range. When the last field specified in the range is reached and you have finished altering your data, QUICK returns you to the Action field for your next command.

Once you've made all your changes to the data on the screen, save the changes on file with one of the update commands (Update, Update Stay, Update Next, or Update Return).

Adding Data in Find or Select Mode

If your screen is designed with repeating groups of fields, and you are in Find or Select mode, you can add more data to a screen by entering the Append command (A) in the Action field. First, you must use Find or Select mode to retrieve the existing data. Then enter the Append command. QUICK prompts you in the first empty field in the first empty repeating group. If no repeating groups are empty, QUICK clears all of the repeating groups and prompts you in the first field of the first group.

Deleting Data

To delete data from your files or tables, first retrieve the data that you want to delete using the Find or Select command. When PowerHouse prompts you for an action, enter one of the following Delete commands:

Delete (D)

Removes all entries from the screen. The data is actually deleted when the screen is updated.

Delete Occurrence (D-n)

Removes one of a repeating group of fields by specifying the ID-number of the group after the hyphen. The group is actually deleted when the screen is updated.

Delete Range (D-n/n)

Removes a range of repeating groups by specifying a range of ID-numbers after the hyphen. The groups are actually deleted when the screen is updated.

Delete Safeguards

QUICK safeguards against accidental deletions by retaining the deleted data until you actually update. To take the final step, enter one of the Update commands. The data is now permanently deleted.

If you change your mind about deleting an entry after entering a Delete command, enter the Find command. PowerHouse warns you that the data has been changed but not updated. Enter the Find command again to confirm your entry, and to find your data. Once you find it, you can change it.

Keep in mind that if you're entering data in Entry mode, and have not yet recorded your data by updating, a Delete command takes immediate effect and all the data on your screen is lost.

Running Reports and Volume Updates

You can add instructions to your application that allow you to run reports or make volume changes using QUIZ or QTP. When you enter the option's ID-number in the Action mode, you may see a system message telling you that the report is running or that the update is taking place.

Responding to Prompts

When you select an option that initiates a report or a volume update, you may see a prompt asking you to specify which data to include. A prompt allows you to specify particular types of information to include in your report or volume update.

For example, if a report includes a prompt to specify patient numbers, you may see:

```
Enter Patient Number:
```

At this point, enter one value for Patient Number. PowerHouse may prompt you for another value. The prompting continues until you press the Enter key without typing a value. Enter the values the same way you would on a data screen.

Miscellaneous Commands

Refreshing Your Screen

Sometimes your QUICK screen can become cluttered with system messages. The Refresh Screen (<Ctrl-G>) and Refresh All (<Ctrl-W>) commands will refresh your screen. These commands can be entered in the Action field or in a data field.

Getting Information

Entering the Information command (I) in the Action field displays the name and creation date of the screen.

Printing

To print a copy of your current screen, enter the List command (L) in the Action field. To print a copy of your current screen and all screens that lead to it, enter the List All command (L@) in the Action field. These commands send copies of the screens to the designated file QKLIST.

Advanced User Interface Features

QDESIGN lets you create screens with optional features that make them easier to use. These features include:

- overlayed screens
- multiple-line messages and pop-up message windows
- full screen help and pop-up help windows
- command processing using the Action field, Action bars, and field marking
- fixed and scrolling data fields
- pop-up data entry windows
- scrolling Selection boxes

Overlayed Screens

In QUICK, you can only work with one screen at a time. However, more than one screen can be visible if you have created screens that are smaller than the terminal window. Pop-up screens are often like this.

When a new screen is called, it overlays the display from preceding screens. The called screen is the "active" screen, and this screen maintains control until you exit from it.

When you exit from a screen, it is removed from display and all previously overlayed information is restored.

Messages

You may see messages on a QUICK screen in either the message line or a pop-up message window.

Every screen has a one-line area for a message line that extends across the terminal window. You don't have to respond to messages presented here.

For multiple-line messages, the screen above the message line is cleared to present the multiple-line message, along with the prompt "Press Enter to continue". When you respond, the underlying information is restored.

Messages can also be displayed in a pop-up window.

After you read the displayed message, press the Enter key. QUICK closes the pop-up window and restores the display.

If the pop-up window is smaller than the length of the message text, the window scrolls. The bottom border of the window appears as a dashed line to indicate that there is more text below. To scroll within the window, enter the Scroll Up, Scroll Down, Page Up, and Page Down commands. By default, these commands correspond to the Cursor Up, Cursor Down, Page Up, and Page Down keys. Once text has scrolled, the top border of the window appears as a dashed line, indicating there is more text above.

Help

QUICK supports a Help system that presents information about the screen and data fields when you enter a help request.

If you request help for a screen, the help message you get relates to the entire screen. If you request help in a field, the help message relates to that particular field.

Two levels of help are available to you: Help (?) and Extended Help (??). Help is a one-line description presented in the standard message area for information messages. Extended Help is more detailed, multiple-line information.

You can present Extended Help information as a full screen display or in a pop-up help window.

If you have implemented Extended Help as a full screen display, a request for Extended Help clears the screen and displays help information. You can move through the displayed information one page at a time.

If you have implemented Extended Help as a pop-up window, a request for Extended Help calls up a pop-up help window. You scroll through Help information in a pop-up help window in the same way as in a pop-up message window. Press the Enter key to close the window.

Entering Action Commands

You can enter Action commands

- in the Action field
- from an Action bar
- using field marking
- using a function key

Using the Action Field

The Action field is an optional prompt area on the screen in which you enter Action commands.

Using an Action Bar

The Action bar is an optional way of presenting and selecting available Action commands. An Action bar is an inverse line that displays a list of commands or menus from left to right across the screen like a menu.

To move across the Action bar, use the Next and Previous Option screen commands. By default, these commands correspond to the Cursor Right and Cursor Left keys. As you move, the current selection is highlighted.

When you've selected the Action bar option you want, press the Enter key. If you select an Action command, PowerHouse processes the Action and returns to the Action bar. If you select an Action menu, the menu opens with the first menu item highlighted.

If all the menu options can't be displayed at one time, the bottom border of the menu appears as a dashed line to indicate there are more menu options.

You can scroll through the Action menu using the Move Up, Move Down, Page Up, and Page Down commands. By default, these commands correspond to the Cursor Up, Cursor Down, Page Up, and Page Down keys. When you reach the bottom of the Action menu, any additional menu options appear one line at a time as you scroll downward. At the same time, the top-most items disappear, and the top border appears as a dashed line to indicate there are more menu options above.

If you open an Action menu and want to return to the Action bar without selecting any of the displayed menu items, enter the Cancel command. The default Cancel command varies by terminal. It is often function key 8.

After you select an Action menu item, press the Enter key. QUICK closes the Action menu, processes the action, then returns to the Action bar.

Using Field Marking

Field marking lets you choose a screen item for processing by pointing directly at the item ID-number. Field marking is also referred to as Full Screen Selection.

With field marking, you use the cursor keys to move through the IDs and labels one at a time. An inverse highlight marks the current ID-number or label. Press the Enter key to choose the item(s) with the highlighted ID-number.

If a data field is marked, QUICK prompts you for data. If a subscreen is marked, that subscreen is invoked. If a command is marked, that command executes.

The Next and Previous Option commands move you to either the next or previous field and changes the mark highlight. By default, these commands correspond to the Cursor Right and Cursor Left keys.

Using Function Keys

You may be able to use function keys to enter Action commands at any time. See "Using Function Keys to Enter Commands" (p. 20).

Changing Action Modes

You can create QUICK screens that let you use a combination of Action field, Action bar, and Field marking to enter Action commands. To select the Action mode you want, use the Action Field, Action Bar, and Field Mark commands. Typically, function keys are mapped to these commands.

When a command is selected in any Action mode, QUICK executes the command, then reprompts using the current Action mode. If the current Action mode is

Action field (ACT)

QUICK reprompts at the Action field prompt.

Action bar (BAR)

QUICK reprompts at the Action bar.

Field marking (MARK)

QUICK reprompts at the current marked field

Fixed and Scrolling Data Fields

You can create three kinds of data fields in which to enter information:

- fixed
- horizontal scrolling
- multiple-line vertical scrolling

Fixed Fields

Fixed fields are displayed entirely on the screen. You can edit while entering data by using the infield editing commands. For more information, see "The Terminal Interface Configuration Screen" (p. 278) and "Modifying TIC Files" (p. 283).

To complete an entry in a data field, press the Enter key.

Horizontal Scrolling

Horizontal scrolling lets you enter data into fields that are shorter than the data items they represent. Scroll indicators identify horizontal scrolling fields. You can edit within a field while entering data by using the infield editing commands. For more information, see "The Terminal Interface Configuration Screen" (p. 278) and "Modifying TIC Files" (p. 283).

Scrolling horizontal displays are like a moving frame, letting you see a data field a portion at a time. For example, if you have a comment field that's 120 characters long with a data entry field 30 characters long, you can see only 30 characters of the field at a time.

During data entry and editing modes, displayed information scrolls as required as you add or delete information.

When you've finished entering data, press the Enter key.

Multiple Line Vertical Scrolling

Multiple-line scrolling fields are similar to horizontal scrolling fields, except that data is entered on several lines, and the field scrolls vertically a line or page at a time. Scroll indicators identify multiple-line scrolling fields. You can edit within a field while entering data by using the text editing commands. For more information, see "The Terminal Interface Configuration Screen" (p. 278) and "Modifying TIC Files" (p. 283).

You can enter one or more paragraphs of data into a multiple-line field. Each paragraph starts on a new line. Long paragraphs are automatically wrapped onto the next line. To start a new paragraph use the New Paragraph command. The default for the New Paragraph command varies by terminal. When you've finished entering data, press the Enter key.

Pop-up Data Entry Windows

Pop-up data entry windows let you use a multiple-line scrolling pop-up window to enter and change information in a data field.

You enter and edit information in a pop-up data entry window in the same way as you would in a multiple-line scrolling field.

To switch between the pop-up data entry window and the data field, use the Popup Toggle command (+).

Selection Boxes

Selection boxes let you enter and change information in a data field. You choose a value from a predefined list which contains acceptable values for the active field. Values appear in a pop-up list.

To open a Selection box, enter the Select Box command (#) in a data field. The Selection box appears with the first item highlighted. If the box contains more values than those displayed, the bottom border appears as a dashed line to indicate there are more values above.

Use the Move Up, Move Down, Page Up, and Page Down commands to scroll through the Selection box. When you reach the bottom, any hidden values appear one line at a time as you scroll downward. At the same time, the top-most values disappear, and the top border appears as a dashed line to indicate more values above.

To select a value, press the Enter key.

If you don't want to choose a value, enter the Cancel command to return to the prompt.

QKView (Windows)

QKView is a Windows shell client for QUICK on Windows. QKView runs QUICK as a subprocess. It is essentially a terminal emulator that traps the output that would normally go to the Command Prompt window and interprets it as graphic output. User input is passed to QUICK as if it came directly from the Command Prompt window.

To start QKView, run it from the Start menu or from a Command Prompt window in the same manner as you would run QUICK.

Configuration

QKView requires a configuration to identify the location of the QUICK executable. As well, you can specify a working directory and the program parameters that you would normally specify for QUICK.

Creating and Maintaining a Configuration File

When you start QKView you must enter a configuration or load an existing configuration.

To create a configuration and save a configuration file, follow these steps:

1. Click the QUICK menu and select the Configuration entry.
2. The Configuration dialog opens. Enter the information as follows:

   QUICK Program Parameters

   Enter the program parameters that you would normally use on the QUICK command line if you were running QUICK in a Command Prompt window. Do not enter the QUICK executable, only the program parameters. Program parameters are passed to QUICK and are used in the same way as if QUICK was started directly. The **debug** program parameter is ignored. This entry is optional.

   Working Directory

   This is the location that you would normally start QUICK from, not necessarily the QUICK installation location. It may be easier to provide a working directory than to use the **procloc** program parameter. The default is the PowerHouse 4GL installation directory. Environment variables can be used in the location. This entry is optional.

   QUICK location

   This is the location of the QUICK executable which will be the installation location. When you click the Browse button, the default location is the PowerHouse 4GL installation location. Environment variables can be used in the location. This entry is required.

   Application Banner Image

   This is a bitmap image (.bmp) that is displayed just above the screen. By default it occupies the three lines above the screen. The number of lines can be specified in Settings. It is scaled as required. Only one image can be specified per configuration and it remains visible for the duration of the session. This entry is optional.
3. Click OK.
4. To save the configuration as a file (.qfg), click the File menu and select the Save or Save As entry. Browse to the desired location, enter a file name, and click OK.

To load an existing configuration file, follow these steps:

1. Click the File menu and select Open, or click the Open icon on the Toolbar.
2. Browse to the Configuration file location, select the file, and click Open. The last used location is the default starting point when browsing.

To make changes to an existing Configuration file, follow these steps:

1. Load the existing Configuration file.
2. Click the QUICK menu and select the Configuration entry.
3. Make the desired changes and click OK.
4. Click the File menu and select the Save or Save As entry.

Using the Configuration File

After you load a configuration file, you run QUICK by clicking the QUICK menu and selecting Run or by clicking the Run button on the Toolbar. QKView responds by running QUICK as a subprocess passing the program parameters specified in the QUICK Program Parameters entry.

If you specify the **auto** program parameter, and it specifies a screen name (.qkc) or a QKGO file (.qkg or .qki) that specifies a First Screen, that screen is loaded immediately.

If you specify a Configuration file after the QKView executable name in a shortcut, QKView loads the Configuration file immediately and will start any screen specified by the **auto** program parameter. When you start QKView in this way and start a screen automatically, QKView will terminate automatically when you exit the start screen unless you check "Disable auto exit" in the Settings dialog.

This is also how QKView will operate if Configuration files have been associated with QKView and you double click on a Configuration file.

Once QUICK is running under QKView, use QUICK as you would if it were running in a Command Prompt window.

Settings

The Settings dialog is opened when you click the QUICK menu and select Settings. You can also click the Settings button on the Toolbar. Settings are user specific and are saved in the Windows registry. The values are saved from session to session. The options are:

Banner Image Lines

Specifies the number of lines that the application banner image occupies above the screen. The application banner image is identified in the Configuration file. Banner image lines are not part of screen lines. The image specified in the configuration is scaled as required. If no image is specified in the configuration, no space is used. The range is 1 to 5 lines. The default is 3 lines.

Function Key Lines

Specifies the number of lines that function key buttons will occupy below the screen. Function key lines are not part of screen lines. If function keys are not enabled by clicking Function Keys in the View menu, no space is used. The range is 2 to 5 lines. The default is 3 lines.

Show splash screen at startup

Signifies whether the QKView splash screen shows at product start.

Auto exit when QUICK session ends

Indicates whether QKView exits automatically when the top-most screen is exited. This option is only effective if a first screen is specified as described in "Using the Configuration File" (p. 32).

Style

Indicates whether the line drawing style is sunken or raised. The default is raised.

Thickness

Indicates whether the line drawing style is thin, medium, or thick. The default is medium.

Show fixed pitch fonts only

Specifies whether only fixed pitch fonts are shown in the Font dialog box. Even though QKView is a Graphic interface for QUICK, QUICK itself is character based and screen spacing is based on rows and columns. Very few proportional fonts will look pleasing to the user or scale properly if the user resizes the window. Fixed pitch fonts are recommended with QKView.

QKView Considerations

COMMAND and RUN COMMAND

In QUICK, the COMMAND statement and the RUN COMMAND verb start a subprocess within QUICK but any prompting and output appear in the same Command Prompt window as QUICK. Since QKView is running QUICK as a subprocess, any additional subprocess cannot communicate with QKView. Therefore, the COMMAND statement and RUN COMMAND verb in QUICK open a separate Command Prompt window when QUICK is running under QKView.

If you do not want a Command Prompt window to open, specify the NOCONSOLE option on the COMMAND statement or RUN COMMAND verb. You would use this option if you were running a command that did not require any input and you do not want to see any output.

The CLEAR, REFRESH, and RESPONSE options are oriented towards command output appearing in the Command Prompt window where QUICK is running. Since QKView opens a separate window, these options aren't normally needed, however, if they are specified, they will be used.

For reports, the recommended solution is to write the report to a file and then read the file with a text editor. This provides a viewer-like interface to the report along with Windows printing capability.

Function Keys

You can use function keys for QUICK commands in the same way as you would on a terminal or in a terminal emulator. To display or hide the eight function keys, toggle the Function Keys entry under the View menu. You cannot change the number of function keys that are visible.

The default setting for the Function Key Support Mode in QKGO is Fixed Standard. In QKI the FUNCTION_KEY_MODE setting is 1. You can disable function keys in QKGO or by setting FUNCTION_KEY_MODE to 0. Dynamic function keys are also available by using the appropriate setting in QKGO or by setting FUNCTION_KEY_MODE to 2.

The other function key settings in QKGO work in the same manner as they do for terminals on other platforms.

Dynamic function keys use the key definitions from the KEY statement.

Loading Function Keys

You can load QKView's function key contents and labels in a similar manner as you would load a terminal's function keys. When the function key is pressed or the function key image is clicked, the function key contents are transmitted to QUICK as if they were entered by the user. QKView recognizes two escape sequences that you issue using INFORMATION verbs. For details on using the INFORMATION verb, see "INFORMATION" (p. 437). You cannot load QUICK's function keys because QUICK runs in a Command prompt window.

You construct the commands using an expression and issue them with the INFORMATION verb with the NOW option. The NOW option tells QUICK to execute the verb immediately and send the message contents to the display. Without the NOW option, the value is sent to the display buffer to wait for a user prompt. You must also specify the **term** program parameter as

```
term=windows-any
```

The any option tells QUICK to ignore any escape sequences rather than converting them into a question mark (?). This may cause issues if your data contains unprintable characters.

The escape sequence for labels is

esc#key-length-label

- esc is one byte containing the escape character, decimal 27.
- # is the command to set the label.
- key is a two-digit function key number starting at 0. Key 1 is 00, key 2 is 01, and so on. QKView supports a maximum of 8 function keys.
- length is a two-digit label length preceded and followed by a hyphen. The value range is 0 to 16.

- label is the label text. Do not use quotes unless you want quotes to appear in the label. The maximum label size is 16 characters in two lines of 8 characters. In other words, the first 8 characters is the first line and the second 8 characters is the second line. It's best to specify all 16 characters.

The escape sequence for contents is

esc%key-length-contents

- esc is one byte containing the escape character, decimal 27.
- % is the command to set the contents.
- key is a two-digit function key number starting at 0. Key 1 is 00, key 2 is 01, and so on.
- length is a two-digit contents length preceded and followed by a hyphen. The value range is 0 to 80.
- contents is the contents to be loaded into the key. Do not use quotes unless you want the quotes to be included in the value sent to QUICK. The maximum length is 80 characters.

To clear key labels and contents, use a length of zero and no label or contents value.

Since the escape character is a nonprinting character, the sequence is typically put together using expressions. The following example shows how to set a function key and how to clear it. Always set the key label before the key contents. Note that the DEFINE statement to extract the escape character takes into account that the Windows platform is little endian, so the byte order is reversed.

```
> DEFINE ESC_CHAR_NUM INT SIZE 2 = 27
> DEFINE ESC_CHAR CHAR SIZE 1 = CHAR(ESC_CHAR_NUM)[1:1]
> TEMPORARY ESC_MSG VARCHAR*80
> TEMPORARY KEY_NUM INT*2
> TEMPORARY KEY_LBL CHAR*16
> TEMPORARY KEY_CMD CHAR*80
. . .
> PROC DESIGNER SETK NODATA  ; DESIGNER procedure to set a key
>   BEGIN
>     LET KEY_LBL = "  FIND   Boston "
>     LET KEY_NUM = 4 ; This is function key 5
>     LET ESC_MSG = ESC_CHAR + "#" + ASCII(KEY_NUM,2) + "-16-" + KEY_LBL
>     INFO MESSAGE = ESC_MSG NOW
>     LET KEY_CMD = "F;BOS"
>     LET KEY_NUM = 4
>     LET ESC_MSG = ESC_CHAR + "%" + ASCII(KEY_NUM,2) + "-05-" + KEY_CMD
>     INFO MESSAGE = ESC_MSG NOW
>     END
> PROC DESIGNER CLRK NODATA  ; DESIGNER procedure to clear a key
>   BEGIN
>     LET ESC_MSG = ESC_CHAR + "#04-00-"  ; Sets the label to nothing
>     INFO MESSAGE = ESC_MSG NOW
>     LET ESC_MSG  = ESC_CHAR + "%04-00-"  ; Sets the contents to nothing
>     INFO MESSAGE = ESC_MSG NOW
>     END
```

A function key cannot be both loaded and used as a dynamic function key. QUICK resets dynamic function keys automatically when the context (action or data) changes and when the screen is refreshed. To ensure that there is no conflict, when a label value is set for any function key, dynamic or not, QKView also clears that function key's contents. This means that you must set the label for a key before you set the contents.

If you specify a key as disabled using the KEY statement, as in

```
> KEY 5 ACTION AND DATA DISABLE
```

QUICK does not refresh the key and ignores it otherwise, so you can load a disabled key and what you loaded will be transmitted and used by QUICK as if the user entered the value. This is also a convenient way of highlighting in the KEY section that a key is being used procedurally.

Since you use INFORMATION verbs to load the keys, you can't load the keys in data context. The INFORMATION verb execution will always be in action context. As well, do not attempt to use a loaded function key in action context and a dynamic function key in data context by only specifying ACTION DISABLE on the KEY statement, or vice versa because QUICK resets the keys when you switch contexts. This means that you can't use a key as a dynamic key in one context and a loaded key in another.

The QKView Menus

File

New

Clears the current configuration.

Open...

Opens the Open dialog allowing you to browse to a Configuration file to load. This command is also available as an icon on the Toolbar.

Save

Saves the current configuration in the currently loaded Configuration file. This command is also available as an icon on the Toolbar.

Save As...

Opens the Save As dialog allowing you to choose a location and file name to save the current configuration.

Exit

Exits QKView.

Edit

Copy

Copies selected text to the Windows clipboard. This command is also available as an icon on the Toolbar.

Paste

Pastes the contents of the Windows clipboard to the cursor location. This command is also available as an icon on the Toolbar.

View

Toolbar

Toggles the Toolbar.

Status Bar

Toggles the Status Bar.

Function Keys

Toggles the Function Keys.

QUICK

Configuration...

Opens the Configuration dialog. This command is disabled when QUICK is running. See "Configuration" (p. 31).

Run

Starts QUICK using the program parameters in the QUICK Program Parameters entry. This command is also available as an icon on the Toolbar.

Stop

Terminates QUICK. Normally QUICK should be terminated by leaving the start screen. This command is available in case of problems.

Font...

Opens the Font dialog allowing you to choose the desired font. Only the Font and Font Style selections have any effect.

Settings...

Opens the Settings dialog. This command is also available as an icon on the Toolbar. See "Settings" (p. 32).

Help

About QKView...

Shows the QKView About box. This command is also available as an icon on the Toolbar.

Input Modes

QUICK provides three modes of input processing:

- Field mode, (previously documented as character mode), where each field is treated as a separate entity. All prompting and editing is done on a field by field basis. The user makes an entry in a field and sends the individual field for editing. To send the field, the user presses [Enter] or [Return] depending on the device.
- Panel mode, where some or all fields are processed as a block under procedural control. The user makes entries in all active fields and sends that block of fields for editing. To send the data, the user presses [Enter] or [Return] depending on the device setting.
- Compatible Block mode (**MPE/iX**), which supports Hewlett Packard terminals' block mode setting. All the fields are considered one block which is unknown to QUICK until the data is sent as a block for editing. There is no prompt or response at the fields while entering data. The user makes entries in all fields and presses [Enter] to send the fields for editing.

Terminology (MPE/iX)

In the following sections, the term "HP Block mode" refers to the Block mode capability of HP Block mode terminals. The term "Compatible Block Mode" refers to the input processing mode used by PowerHouse. In the discussion of Compatible Block mode, the term "Block mode" is used since a distinction is not required. Since HP Block mode is always active when using Compatible Block mode, the term "Block mode" can accurately refer to both the terminal setting and the input processing mode.

The term "Character mode" refers to the Character mode capability of HP Block mode terminals. Field mode refers to the field input processing mode used by PowerHouse. In previous releases, this was documented as "Character mode".

The term "Panel mode" refers to the Panel input processing mode used by PowerHouse. In previous releases, this was referred to as "Panel Block Mode".

The following table explains how each of these input modes functions with various devices, device settings, and read types:

Device	Device Setting	Read Type	Field Mode	Panel Mode	Compatible Block Mode
Terminal	Character Mode	Character	✔	✔	
Terminal	Character Mode	Line	✔	✔	
Terminal	Block Mode	not applicable	✔	✔	✔
PC using Axiant or PowerHouse Web	not applicable	not applicable	✔	✔	

Device Settings (MPE/iX)

Device settings are specific to block mode capable HP terminals and indicate whether the terminal will transmit data by block or not. This has nothing to do with the method of input processing. To use HP Block mode, the SCREEN statement must include BLOCKMODE option.

The device setting is controlled using the following options:

INPUT B|C|SAME option

Puts the terminal in HP block mode or character mode.

the B|C screen command

Toggles a screen between HP block mode or character mode.

the Input Mode Execution Time Parameter in QKGO (B|C)

Sets the default start up device setting. If the initial screen specified in QKGO uses the BLOCKMODE option of the screen statement, it can be started in either HP Block mode or Character mode.

Read Types (MPE/iX)

QUICK can read data from a terminal character by character and react to each character in context as it is read. Alternatively, it can read a block of data at a time using lineread input processing, however that block must be terminated by a termination indicator such as [Return] or [Enter].

QUICK provides the benefits of both of these models by dynamically switching between the two models, based on where input is required in the given screen. QUICK either processes input by way of lineread input processing or by using single character processing depending on the current application context.

The read type, which determines whether QUICK uses single character reads, is specified using the READ program parameter. Valid options for the READ program parameter are:

CHAR

This option forces QUICK to run the entire application with the use of single character processing. Character read signifies that QUICK will request one character from the terminal at a time.

LINE

This option means that the entire application runs without the use of single character processing. Line read indicates that QUICK will take data from the terminal in a character stream that stops when a carriage return is entered. This program parameter is equivalent to LINEREAD.

When no parameter is provided, the default processing is the equivalent of READ=LINE for standard fields and the equivalent of READ=CHAR when a user interface feature requires single character processing.

With a HP Block Mode terminal setting, the number of characters that QUICK receives is whatever is available in the update fields when [Enter] key is pressed, so the type of read is meaningless.

Feature Requiring Single Character Processing

The following is a list of features where single character processing is activated when no READ program parameter is used:

Automatic Next Fields
Reverse Fields
Action Bars
Pull Down Menus
Select Boxes
Field Mark Mode
Horizontal Scrolling Fields
Vertical Scrolling Fields
Pop-up Windows

Line Editing and Lineread Processing

Fields not requiring single character processing are handled using lineread processing. The following is a list of line editing commands and their default key assignments for HP terminals:

Line Editing Command	Default Key
Clear Field	Control K

Line Editing Command	Default Key
Delete Character	Delete Char
Delete Previous Character	Backspace
Delete to Start of Line	Delete_Line
Delete Word	Control_J
Input Completion/Update Display	Return
Insert Toggle	Insert_Char
Move Left One Character	Cursor_Left
Move Right One Character	Cursor_Right
Move to End of Line	Control_E
Move to Start of Line	Control_T
Recall	Cursor_Up
Refresh Screen	Control_G
Refresh All	Control_GG

Whenever the last keyboard character entered before a [Return] is from the previous table, the field display is updated, and the user is prompted for more input in the same field. If the last keyboard character entered before a [Return] is not from the table, it signals end of input, and processing continues to the next field.

The dual meaning of the [Return] key (update display and input complete) when in lineread mode provides the user with the ability to update the field display (after editing has been performed) without causing processing to move to the next field.

Supporting QDESIGN Syntax

Syntax options which affect the use of PowerHouse input modes include:

BLOCKMODE [EXTENDED] option of the SCREEN statement (MPE/iX)

Indicates that QUICK can run this screen using HP block mode on a Block mode terminal.

PANEL|NOPANEL options of the SCREEN statement

Specifies whether or not generated procedure code will contain options and constructs (BLOCK TRANSFER) required to process QUICK screen fields in panel mode.

Program Parameters

To override the default input modes, use the following program parameters:

CHARMODE=FIELD|PANEL

The CHARMODE program parameter applies when the device setting is Character.

The FIELD option indicates BLOCK TRANSFER control structures are ignored. Run QUICK with the CHARMODE=FIELD to use the Field mode input model. You can use CHARMODE=FIELD to run a Panel screen with field input.

The PANEL option indicates BLOCK TRANSFER control structures are recognized. Run QUICK with the CHARMODE=PANEL to use the Panel mode input model.

Default: PANEL

BLOCKMODE=COMPATIBLE|PANEL (MPE/iX)

The BLOCKMODE program parameter applies when the device setting is Block.

To run HP Block mode, the QUICK screen must be compiled with the BLOCKMODE option of the SCREEN statement and Block mode must be specified in QKGO, with the B screen command or the INPUT option.

Run QUICK with the BLOCKMODE= COMPATIBLE program parameter to use the Compatible Block mode input model. Run QUICK with the BLOCKMODE= PANEL program parameter to use the Panel mode input model.

Default: COMPATIBLE

Field Mode

Field mode is the default input mode for QUICK screens.

In Field mode, the user works interactively with QUICK. QUICK displays a prompt and a user responds with an entry. Users can clearly see when data is being entered, changed, corrected, or found. Users know that they are responsible for updating new or changed data (except in Entry mode on screens with the AUTOUPDATE option). You are free to arrange field prompts in any order, supply reprompts on fields, make prompts conditional, and run subscreens and commands at any point.

Panel Mode

Panel mode allows PowerHouse applications to accept and process one or more data fields as a block, rather than individually. Users can move between the fields in a panel at will (using Panel mode commands such as [Tab] or a mouse). Where Field mode processes each field individually, and Compatible Block mode (**MPE/iX**) processes an entire screen of data as one block, Panel mode lets the designer group fields and control access to fields procedurally. A control structure, BLOCK TRANSFER, is used to group fields into panels.

Panel mode may also reduce network traffic in client/server environments, since fewer requests are made of the server.

Data commands are ignored in a panel. They are treated as data.

To move around in a panel, use [Tab]. To send the data for processing, use [Enter] or [Return]. The equivalent of the Backout command (^) is the Fixed Standard function key F8.

If you are using dynamic function keys as opposed to Fixed Standard function keys, data context commands will not work in a panel. You can set the Backout data context command to a specific function key that will function as the Fixed Standard function key regardless of whether dynamic function keys are being used. For example, to specify that F8 is to be used as the Backout command, you would specify that the F8 key is assigned to command 2 in data context as always rather than fixed=0. For more information, see "Modifying TIC Files" (p. 283)

Designing Screens with Panel Input

You can use the following QDESIGN syntax to specify what fields belong to what panels and to procedurally control what happens during Panel input:

Syntax	Description
PANEL option of SET statement	Generates the necessary procedure code to support panels specified in the screen.
PANEL option of SCREEN statement	Generates the necessary procedure code to support panels specified in the screen.

Syntax	Description
AUTOMODIFY option of SCREEN statement	Runs the MODIFY procedure automatically after a FIND or SELECT is done.
BLOCK EACH|ALL option of CLUSTER statement	Use a CLUSTER statement with either the BLOCK EACH or BLOCK ALL option to start the panel, then use FIELD statements to specify all of the fields that belong in the panel, and then mark the end of the panel with a CLUSTER statement with no options.
BLOCK TRANSFER control structure	Defines a panel; without BLOCK TRANSFER control structures, there are no panels in a screen. For each panel on a screen, QDESIGN generates a corresponding BLOCK TRANSFER control structure in the APPEND, ENTRY, MODIFY, PATH, and SELECT procedures. For screens that have modified versions of these procedures, or procedures that accept input (such as DESIGNER procedures), you should add BLOCK TRANSFER control structures as necessary for each panel.

Syntax	Description
MISSING option of FOR control structure	Prompts for occurrences in a panel to ensure that input is accepted into the first free occurrence, rather than overriding an existing occurrence.
MODIFY procedure	Controls how modifications are done in Change mode or Correct mode. When you include the PANEL option on the SCREEN or SET statement, QDESIGN generates this procedure automatically.
SELECT procedure	Controls what fields are available for selection in Select mode. When you include the PANEL option on the SCREEN or SET statement, QDESIGN generates this procedure automatically.
SELECT verb	Used in the SELECT procedure, prompts for selection values.

For more information about this syntax, see the appropriate statement in Chapter 3, "QDESIGN Statements", procedure in Chapter 7, "QDESIGN Procedures", or verb or control structure Chapter 8, "QDESIGN Verbs and Control Structures".

You can use the **charmode=field** program parameter to run a Panel input screen with field input. All BLOCK TRANSFER control structures are ignored when you use this program parameter.

Compatible Block Mode (MPE/iX)

The program parameter BLOCKMODE=COMPATIBLE tells QUICK to use the Compatible Block mode input model. QUICK ignores BLOCK TRANSFER statements and input is handled as a block consisting of the entire screen's data including the Action field.

Compatible Block Mode causes a Block mode terminal to run in standard HP Block mode. For this section, Block mode is used to mean both input mode and terminal setting since a distinction is not necessary.

In Compatible Block mode, unlike in Field mode, data on the screen is unknown to QUICK until the user presses either [Enter] or a dynamic function key (DFK). There is no prompt or immediate response at each field. When all the data for a screen is entered, the user presses [Enter], or a DFK where applicable, to transmit the data to the Block mode buffer (BMB). QUICK then edits the data. If QUICK detects errors, it marks all invalid fields with highlighting, and displays the error message associated with the first invalid field on the message line.

If all the data passes edit processing, QUICK can then update the file and clear the screen. This depends, however, on what initiated the data transmission- the Enter key or a dynamic function key. For more information, see "Processing and Updating Data" (p. 46).

An important distinction between Field mode and Compatible Block mode is the difference in timing. The difference is a result of the full screen block reads that occur in Compatible Block mode. Until the user presses either [Enter], or a DFK, data on the Compatible Block mode screen is unknown to QUICK. Therefore, QUICK provides no opportunity for a dialogue with the user. As a consequence, you must review procedures written for Field mode screens to see if they make sense for Block mode operation.

QUICK has a particular way of processing Block mode input in Entry mode. The screen image is scanned in the reverse order of the ACCEPT verbs (From the ENTRY procedure) to determine where it is possible to internally substitute data field commands, such as Return to Action, for the field value. This internal optimization won't work properly if the execution of the ACCEPT verbs is conditional, because the actual processing sequence can change based on values entered.

In some data-processing environments, Block mode screens are more efficient than Field mode screens. Block mode can reduce data transmission costs and I/O load on a system because reading, editing, and writing are performed on screenloads of data rather than on individual fields of data. In a packet-switched network (for example, a public X.25 network), transmitting a screenload of data in one packet (as in Block mode) is cheaper than transmitting many individual packets (as in field mode). Also, in a high-volume data entry system, editing responses don't delay users at every field.

Implementing Compatible Block Mode Capability

To implement Compatible Block mode capability on your application screen, include the BLOCKMODE option on the SCREEN statement of the design, as in

```
> SCREEN STAFF BLOCKMODE
```

Users now have the option of entering B in the Action field to use the screen in Block mode. QUICK starts the first screen in Block mode if the Input mode QKGO parameter is set to B.

Block mode subscreens called from Character mode screens automatically start in Block mode if the SUBSCREEN statement includes the INPUT B option. The calling screen retains its original input mode when the subscreen exits.

If a user working in Block mode calls a subscreen that does not support Block mode, the called screen automatically appears in Character mode.

Terminal Memory Mapping

Block mode screens are always mapped to lines 1 through 24 of terminal memory. Block mode screens can take full advantage of application lines, but not terminal memory. When a system of screens uses both Character mode and Block mode, you can make full use of terminal memory by using terminal lines

- 1 through 24 Block mode screens
- greater than 24 more Character mode screens

For more information on application lines and terminal memory, see "SCREEN" (p. 182).

Switching from Block mode to Character mode (and vice versa) may cause new screen data to be lost. The same loss can occur if one of several commands is entered at the wrong time. QUICK issues a warning in such cases, and the data will be lost if the action is repeated. For more information, see "Multiple Command Processing" (p. 49). Rapid-fire entry (several commands separated by semicolons) is not valid on Block mode screens. If a rapid-fire entry made on a Character mode screen contains commands that

- switch to Block mode
- move to a screen where Block mode is active

the portion of the command set that follows the switch to Block mode is discarded.

Input modes are considered to be local to the screen. You can switch back and forth while retaining the Input mode for each screen. When a screen that is in Block mode calls a screen without Block mode capability, the terminal switches into Character mode. When control returns to the calling screen, the terminal returns to Block mode.

Implied AUTOUPDATE

When the [Enter] key is pressed, Block mode screens have an implied "autoupdate" feature that updates records automatically after a screen of data has been successfully processed. The AUTOUPDATE option causes a similar effect on Character mode screens in Entry mode. You can use the NOAUTOUPDATE option on the SCREEN statement to suppress the autoupdate feature of Block mode screens.

Extended Fields

When selecting Block mode as a capability for your screen, you should decide whether or not you want extended fields on you Block mode terminal. Terminals issue a beep when a user enters the last character of a field, and the cursor automatically advances to the first character of the next field. If an entry has fewer characters than the field length, a user can tab to advance to the next field.

It is often more convenient to extend the length of fields by one character so that a user must tab to advance to the next field. Beeps then occur only if users enter too much data in a field. If you want extended fields, add EXTENDED to the SCREEN statement, as in

```
> SCREEN STAFF BLOCKMODE EXTENDED
```

Layout Considerations

To mark the end of a QUICK screen, Block mode employs a one-character record separator at the end of the last field in a screen. Any data entered in this special character area is ignored. The last field of a Block mode screen must never end in column 80, since this would leave no room for the record separator.

Highlighting

Highlighting is important in Block mode. Input fields must be distinguishable from the rest of the screen since there is no individual field prompting; invalid fields must be distinguishable from accepted fields.

For terminals that support highlighting, the defaults are as follows:

The following...	are highlighted in...
input fields	INVERSE
invalid fields	INVERSE HALFTONE
prompts for key fields	INVERSE HALFTONE
function key labels	INVERSE HALFTONE

Use the HILITE statement in QDESIGN to assign highlighting to screen entities. QUICK ignores any highlighting features not supported by a terminal.

Entry Processing

When a user is prompted for data in Block mode, the screen is effectively isolated from the computer until the data is transmitted. A user can tab backward or forward to any field, enter data, erase data, or change data without any response from QUICK. Only the final entry- all data currently showing in the fields- gets transmitted. QUICK treats unchanged fields (such as a field with no data entered) as a null response, and assigns default values to the field's associated item during processing.

Don't try to write procedures to interact with a user in Compatible Block mode, since procedures can't recognize any data until it is transmitted.

You shouldn't include fields in a Block mode screen that are processed in a nonsequential order or that are processed under conditional logic. A user can only see a highlighted block into which entries are made. A user has no idea of the order in which you have set field prompting or if a field is to be accepted. Therefore, the user has no way of knowing whether or not conditions are met.

Linking Block Mode Screens

In Entry mode, pressing [Enter] or a DFK with the BLOCKTRANSFER option executes all statements. This includes COMMAND, SUBSCREEN, and THREAD statements with the AUTO option, depending on the order in which they appear in the screen design. If there are no errors, QUICK performs the command, the subscreen or the thread call. If QUICK detects an error in a field before a COMMAND, SUBSCREEN, or THREAD statement, it ignores that statement until the errors are corrected.

Considerations for Subscreens

Placing a SUBSCREEN or THREAD statement before the last FIELD statement can cause problems in Block mode if errors occur in subsequent fields. These fields aren't processed until QUICK returns from the subscreen or thread. If QUICK detects an error in these fields, it won't back out the screen load of data, since the subscreen may have updated records passed from the calling screen as well as related records. Backing out in such cases could corrupt the application data.

To ensure that QUICK does not update data in the MASTER file without editing entries for all fields on the screen, you should position SUBSCREEN, THREAD, and COMMAND statements after all FIELD statements in Compatible Block mode design. If you must call a subscreen in the middle of the entry sequence, you should use procedures to maintain data integrity.

Find and Select Mode Processing

In Find mode, there is one screen read for all requested fields highlighted at the start of the PATH procedure. A user must enter all necessary key information at one time. In Select mode, this same procedure is used, followed by a read for all selection criteria. Because users aren't prompted through a sequence of key fields, they must know when multiple key entries are necessary for retrieval.

The PATH procedure is processed as it is in Character mode. In Find mode, entries in unprocessed key fields or nonkey fields cause an error condition. In Select mode, QUICK treats such unprocessed field entries as selection criteria, and skips the additional selection read.

For each screen load of data retrieved in Find mode, a user is allowed to make changes to fields on display. Any changes cause the associated numbered DESIGNER procedures to be performed for spot changes. Unchanged fields are not processed unless they share the same ID and a null response is assumed. Errors cause error highlighting of invalid fields and additional reads from the terminal until no more errors are found.

In Find mode or Select mode, pressing a dynamic function key or entering an Action field command may affect the prompting sequence, or interrupt the process entirely. Most commands cause QUICK to abandon the record retrieval process. For more information, see "Multiple Command Processing" (p. 49).

If the effect of the command is not detrimental, one of the following occurs in Find mode:

If a key value...	QUICK processes the command, and then...
has already been entered	retrieves records.
has not been entered	reprompts for a key value.

In Select mode, the actions are similar:

If a key value...	QUICK processes the command, and then...
and selection values have already been entered	retrieves records.
has been entered, but no selection values has been entered	reprompts for selection values.
has not been entered	reprompts for a key value and selection values.

Display Fields

A user cannot address fields with the DISPLAY option; consequently, they cannot be used for key requests or selection. During data entry, users do not have to tab over fields with the DISPLAY option. Such fields become part of the screen background and are not accessible to the user.

Transmitting Data

In Block mode, the process of entering new data in Entry mode, or of changing retrieved data in Find mode, is local to the screen. QUICK does not recognize the data until a user presses either the [Enter] key or a dynamic function key with the BLOCKTRANSFER option.

Transmission involves moving data from the terminal into QUICK's Block Mode Buffer (BMB). Each Block mode screen in a screen hierarchy has its own BMB. The ENTRY procedure and the numbered DESIGNER procedures are not executed unless data has been transmitted. If you intend to use DFKs to transmit data, these keys must have an explicit or implied BLOCKTRANSFER option. For more information on dynamic function keys, see Chapter 6, "Customizing QUICK with QKGO", and "KEY" (p. 156).

On most occasions, you will want data transmitted as part of a dynamic function key definition. For example, an UPDATE command option with the NOBLOCKTRANSFER option updates the file using data currently in the record buffer; the current screen data is ignored.

NOBLOCKTRANSFER is not recommended with the EXTENDED HELP command option because EXTENDED HELP clears the current screen of data before displaying the help screen.

In some cases, you won't want data transmitted. For example, using a SHIFT command option with the NOBLOCKTRANSFER option presents unnecessary I/O.

Refreshing the Screen

The standard Refresh [Control-G] or Refresh All [Control-GG] commands for the Field model in Character mode don't refresh the screen in Block mode. If fixed standard function keys are being used, [F4] and [F1_F4] respectively, perform these functions.

If dynamic function keys are assigned, you should specify keys to perform these functions using the REFRESH and REFRESH ALL options and specify the NOBLOCKTRANSFER option to prevent QUICK from transmitting corrupt screen data.

Processing and Updating Data

In standard Block mode operation, a user enters data on the screen, then presses [Enter].

In...	the data...
Entry mode	is moved to the BMB and processed by the ENTRY procedure.
Correct mode or Change mode	in each changed field is processed by the default-numbered DESIGNER procedure(s).
in any mode, where there are no errors	is updated automatically.

In Block mode, no automatic update is provided for data on slave or menu screens. Explicit updates can be specified. The use of dynamic function keys or command combinations (for example, the [Enter] key with a command in the Action field) changes the standard processing and updating.

- A DFK with the NOBLOCKTRANSFER option executes the command or series of commands assigned to it; it does not transmit data and the BMB is ignored. No automatic update is performed. One of the update command options can specify an update.
- A DFK with the BLOCKTRANSFER option first transmits data to the BMB, and then performs whatever processing is associated with the current mode. The DFK then executes the command or series of commands assigned to it. If one or more of the commands need to process data, they use whatever data has been transferred to the BMB. An automatic update is not performed. One of the update command-options can specify an update.
- If a command is in the Action field when you press [Return], data transfer is performed first. Depending on the type of command, processing associated with the current mode may be performed before or after the command is executed. For more information, see "Multiple Command Processing" (p. 49). An automatic update is not performed. One of the update command options can specify an update.

Error Handling

When an edit error or a user-specified procedural error occurs during processing of the ENTRY procedure or a numbered DESIGNER procedure, QUICK flags the current, or most recently processed, field as an error. QUICK then continues processing in order to locate as many errors as possible in a single screen read. To back out of any completed processing once all fields are accepted, QUICK

1. Performs the BACKOUT procedure, if specified (in Entry or Append mode only).
2. Restores all record buffers to their original state (in Entry or Append mode only).
3. Marks any invalid fields with error highlighting.
4. Displays an error message for the first invalid field encountered.
5. Positions the cursor at the field associated with the displayed error message.

Also, any errors cause QUICK to discard all commands in the Pending Screen Input Buffer (PSIB). For more information, see "Multiple Command Processing" (p. 49).

When errors are flagged, the user must either fix the errors or abandon the current data by clearing the screen in Entry mode. The user may also back out of the screen in Correct of Change mode.

The SEVERE, ERROR, and RETURN verbs also interrupt processing. The SEVERE and ERROR verbs have different effects, depending on the type of procedure they're used in. Generally, their action is similar to that for edit errors. RETURN causes QUICK to leave the current screen and move to the next highest screen. In the highest-level screen, RETURN causes QUICK to issue a screen-ID prompt, unless the first screen option of QKGO specifies a screen name.

QUICK handles errors differently when they are detected after the update phase has begun. In this instance, no fields are directly associated with errors, so any message must specify information about the changes that the user must make to correct the errors. The processing performed in the entry and correction phases is not backed out. QUICK now treats any further changes to the fields entered on the screen as corrections to accepted data. The ENTRY procedure is not performed again.

Important Considerations for Block Mode

When working in Block mode, the following should be taken into consideration:

- Set up the procedures to process fields in the same way that they are entered by the user- left-to-right, and to top-to-bottom. Avoid the OMIT, IF, DISPLAY ON, and NOENTRY options on FIELD statements.
- The VERTICAL option of the CLUSTER statement for the Field model in Character mode prompts for entry from top to bottom following a column or columns of fields. The VERTICAL option is not used in Block mode for more than one column of fields since users always tab between fields in Block mode from left to right, and top to bottom.
- QUICK performs an unprocessed field check before starting the update sequence. This implies that any fields on the screen which have had data entered into them, and which had no ACCEPT performed on them, cause an error. The update sequence is not performed until this problem is corrected.
- Character mode users who infrequently switch to Block mode may be inclined to browse through records in Find mode by pressing [Enter], just as they would press [Return] in Character mode. These are not identical actions- the [Enter] key in Block mode transmits data to the BMB, and updates it. This could cause excessive I/O in Find mode.

  ITEM statements with the FINAL option (for example, a time-stamped item) or procedures in the screen design may change values in the record buffers. This would automatically update the record. A designer should provide a NEXT or NEXT DATA command-option if DFKs are being used.
- If the NOAUTOUPDATE screen option is used, data will not be automatically updated by pressing [Enter].
- All input to Block mode screens is isolated from QUICK until [Enter] or a DFK with the BLOCKTRANSFER option is pressed. Therefore, typing data into fields linked by the ID SAME option may not produce the same messages that would result if the data were entered in the Field model in Character mode.

For example, in a Block mode screen where

- a series of fields linked by the ID SAME option, and
- one or more of the fields have the NOCHANGE option

you can change data in a field with the NOCHANGE option. No error message is issued from QUICK, but the change is ignored. To receive the appropriate messages in Block mode, make sure that all fields have ID-numbers. (Depending on the options used, QUICK may not recognize fields without ID-numbers as valid input fields).

- Do not perform updates directly or indirectly from the ENTRY procedure unless they never need to be backed out. In the case of an error in a field, the user is prompted to correct the error. If the user corrects the errors, the update may be repeated; if not, QUICK has no way of forcing a backout.
- Do not use the ACCEPT or PROMPT verbs for the same field more than once within any single screen read. Any subsequent ACCEPT or PROMPT verbs for the same field results in a null response.
- Avoid complicated REQUEST verb sentences for path determination in Find mode. The user is not prompted in a sequence and may enter data in any field on the screen.
- To procedurally test the Input mode, use the predefined conditions BLOCKMODE and CHARACTERMODE. For more information, see "Predefined Conditions in QDESIGN" in Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

In Block mode, processing of field-related verbs differs from Character mode. For Block mode, QUICK follows some or all of the steps detailed in ACCEPT verb, Chapter 12.

Mode	Verb	QUICK follows steps...
Block	ACCEPT	1 to 7
	PROMPT	1 to 3, 6
	REQUEST	1 to 3, 6
	DISPLAY	8 to 10

- In Block mode, FIELDTEXT is the size of the current field entry, without any trailing blanks. In Change mode and Correct mode, however, if a value is changed to spaces, FIELDTEXT has a size of one.

Multiple Command Processing

QUICK can receive commands from a multitude of QDESIGN features, as well as from screen users. Multiple command processing is QUICK's strategy for handling commands it receives from all these sources.

Command Sources

QUICK can receive commands from any of the following sources:

Command Source	For more information, see ...
Screen users (discrete and rapid-fire entries)	"QUICK Screen Commands" in Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book
Function keys	(p. 271), (p. 273), and (p. 275)
ACTIONMENU statements	(p. 70)
MENUITEM statements	(p. 164)
DESIGNER procedures (PRECOMMANDS and POSTCOMMANDS options)	(p. 303)
PUSH verbs	(p. 454)

The above command sources can be used separately or together. For example, you can design a Dynamic Function Key (DFK) that calls a DESIGNER procedure which includes the PRECOMMANDS and POSTCOMMANDS options and several PUSH verbs.

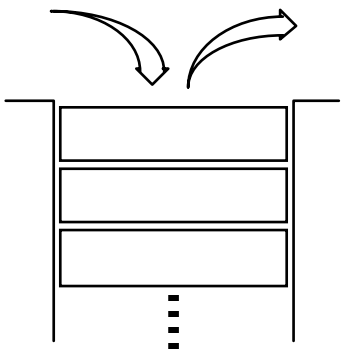
Input Buffers

QUICK has two types of input buffers:

Buffer	Description
Pending Screen Input Buffer (PSIB)	Stores conditional command lists.
Rapid-Fire Buffer (RFB)	Stores user-entered commands and/or data.

Pending Screen Input Buffer (PSIB)

Each screen has its own PSIB. PSIBs are Last In=First Out (LIFO) buffers that establish the order in which conditional command lists are processed, as in:



QUICK does not pass commands between calling screens' and subscreens' PSIBs. When calling a subscreen, QUICK saves the calling screen's PSIB, opens the subscreen's PSIB, and then continues processing on the subscreen. When returning to a calling screen, QUICK clears the subscreen's PSIB, re-opens the calling screen's PSIB, and then continues processing on the calling screen.

Although QUICK doesn't pass commands between calling screens' and subscreens' PSIBs, you can simulate this effect. For more information, see (p. 54).

PSIBs and Conditional Command Lists

You use conditional command lists on KEY, ACTIONMENU, and MENUITEM statements, PUSH verbs, and DESIGNER procedures to send commands to QUICK. The general form of a conditional command list is:

command-list [IF condition
[ELSE command-list IF condition]...
[ELSE command-list]]

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

When QUICK receives a conditional command list, it puts it on the current PSIB. The conditional command list is removed from the PSIB for processing immediately unless QUICK receives other conditional command lists, in which case it is pushed further down in the PSIB.

When a conditional command list is removed from the PSIB for processing, QUICK evaluates the conditions, if any, and then processes the resulting command list. The commands in the command list are executed in the order in which they're specified by the designer.

A Special Note About PUSH Verbs

Unlike the other command sources, you can specify several PUSH verbs in a row for execution under procedural control, as in

```
> PUSH LAST RECORD
> PUSH NEXT DATA
```

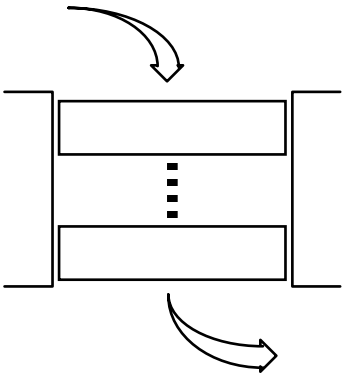
Be aware, however, that this is different from entering

```
> PUSH LAST RECORD, NEXT DATA
```

In the first example, LAST RECORD is put on the PSIB first, followed by NEXT DATA. Due to the PSIB's LIFO configuration, the commands are removed for processing in the order NEXT DATA, LAST RECORD. This is the opposite of the second example, where the commands are processed in the order in which they're specified, that is, LAST RECORD, NEXT DATA.

Rapid-Fire Buffer (RFB)

The RFB is shared by all screens in a PowerHouse application. The RFB is a First In=First Out (FIFO) buffer that stores all user-entered commands and/or data prior to processing, as in:

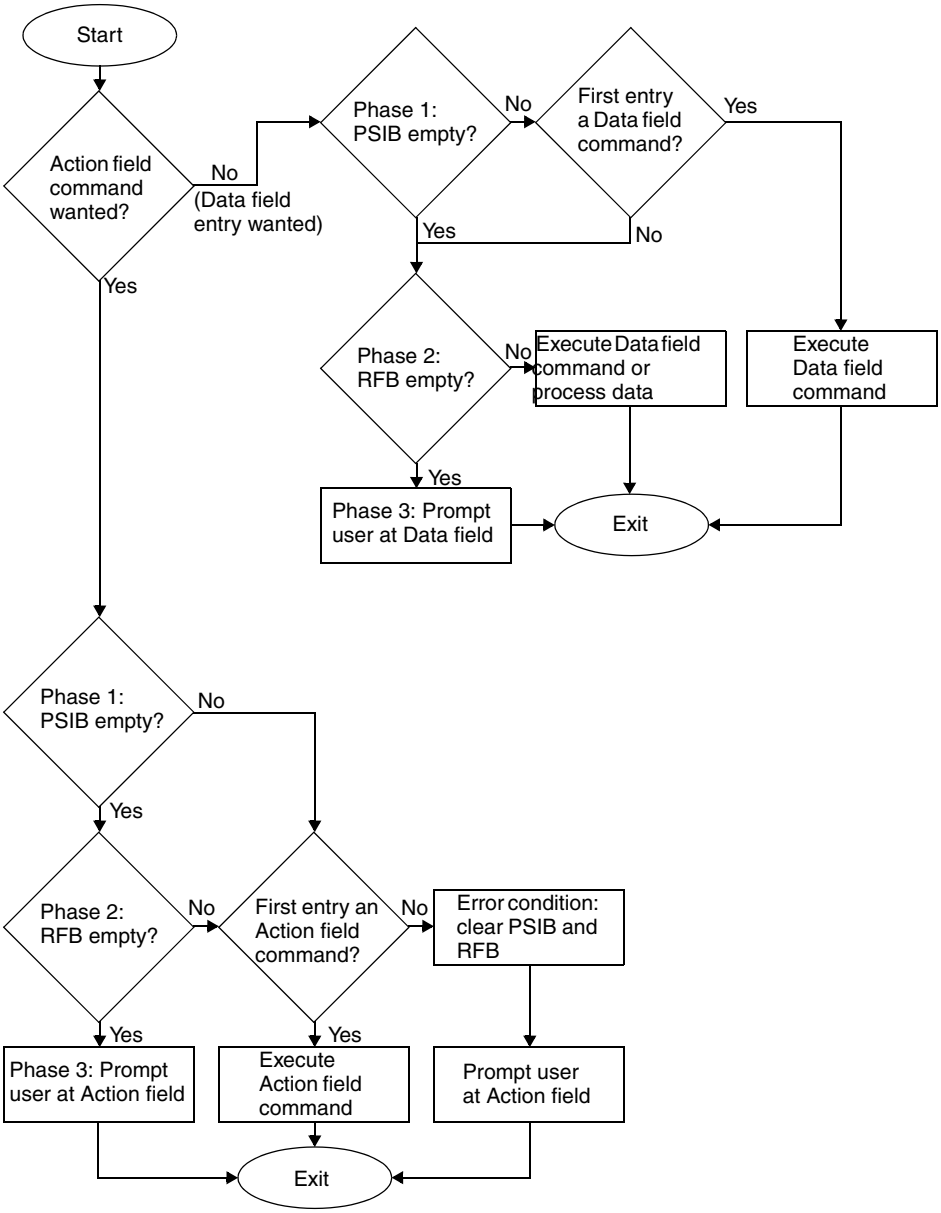


When QUICK receives discrete or rapid-fire user entries, it puts them on the RFB in the order in which they are entered, and removes them for processing in the same order.

Order of Processing

Depending on the current context, QUICK requires either an Action command or a Data field entry. QUICK looks for its next command or entry according to the following phases (these phases are indicated on the processing flowchart on the next page):

Phase	Description
1. Process commands in the current PSIB.	Causes QUICK to give priority to programmatically-entered commands (which are stored in PSIBs). This ensures that designer-controlled features perform as expected, regardless of user entries.
2. Process entries in the RFB.	Causes QUICK to process all discrete and rapid-fire user entries before prompting for new entries.
3. Accept new user input.	Prompts for new entries only when none are available in the PSIB or the RFB. This results in commands being put on PSIBs, entries being put on the RFB, or both, which causes QUICK to return to phase 1.



Error Handling

The way QUICK handles command-related errors depends upon whether it is in Action context or Data context.

Action Context

When QUICK is in Action context, it requires an Action command. Anything other than a valid Action command causes QUICK to clear the PSIB and the RFB, issue an error message, and prompt for new input at the Action field.

Data Context

When QUICK is in Data context, it requires a Data field entry (this could be either a Data command or data for the field). A data field entry that does not comply with the field's type, size, picture, or EDIT procedure conditions causes QUICK to clear the RFB, issue an error message, and prompt for new input at the field in which the error occurred.

A simple rule of thumb about data context is this: if an entry wouldn't work if it were entered as one of a series of discrete entries, it won't work with multiple command processing either.

Examples

Multiple command processing allows you to design very powerful automatic features into your applications. The following examples show some innovative design techniques that take advantage of multiple command processing.

Multiple Retrieval Paths

In the following example, several retrieval paths are predefined as menu items. Users simply select the appropriate retrieval path from the menu, rather than entering retrieval criteria in data fields:

```
> SCREEN XMPL4 ACTIONBAR STARTUP
> FILE EMPLOYEES
> TEMPORARY MY_PATH CHAR*10 RESET AT STARTUP
> ACTIONMENU LABEL "Retrieval Paths"
>      MENUITEM LABEL "Retrieve via Employee No"   &
>         ACTION DESIGNER RTV1
>      MENUITEM LABEL "Retrieve via Last Name"     &
>         ACTION DESIGNER RTV2
.
.
.
>      MENUITEM LABEL "Retrieve All Employees"     &
>         ACTION DESIGNER RTV5
.
.
.
> PROCEDURE DESIGNER RTV1 NODATA
>      BEGIN
>         LET MY_PATH = "EMP#"
>         PUSH FIND
>      END
>
> PROCEDURE DESIGNER RTV2 NODATA
>      BEGIN
>         LET MY_PATH = "LNAME"
>         PUSH FIND
>      END
.
.
.
> PROCEDURE DESIGNER RTV5 NODATA
>      BEGIN
>         LET MY_PATH = "ALL"
>         PUSH FIND
>      END
>
> PROCEDURE PATH
>      BEGIN
>         IF MY_PATH = "EMP#"
>            THEN REQUEST EMPLOYEE
>         ELSE IF MY_PATH = "LNAME"
>            THEN REQUEST LASTNAME
.
.
.
>   END
>
> PROCEDURE FIND
>      BEGIN
>         IF MY_PATH = "EMP#"
>            THEN GET EMPLOYEES VIA EMPLOYEE
>         IF MY_PATH = "LNAME"
>            THEN GET EMPLOYEES VIA LASTNAME
.
.
.
>         IF MY_PATH = "ALL"
```

```
>            THEN GET EMPLOYEES SEQUENTIAL
>      END
```

Passing Parameters Between Screens

Although commands are not passed between calling screens' and subscreens' PSIBs, the following example shows how you can simulate this effect:

```
> SCREEN XMPL5
> FILE EMPLOYEES PRIMARY
> TEMPORARY ACTION_FLAG CHAR*1
.
.
.
> PROCEDURE DESIGNER COPY
>      BEGIN
>         LET ACTION_FLAG = "C"
>         RUN SCREEN XMPL5A PASSING EMPLOYEES, ACTION_FLAG
>      END
> BUILD

> SCREEN XMPL5A RECEIVING EMPLOYEES, ACTION_FLAG
> FILE EMPLOYEES MASTER
> FILE BILLINGS PRIMARY
.
.
.
> TEMPORARY ACTION_FLAG CHAR*1
.
.
.
> PROCEDURE DESIGNER MOVE NODATA  &
>         PRECOMMAND FIND
>      BEGIN
.
.
.
>      END
> PROCEDURE DESIGNER COPY NODATA  &
>         PRECOMMAND FIND
>      BEGIN
.
.
.
>      END
>
> PROCEDURE INITIALIZE
>      BEGIN
>         IF ACTION_FLAG = "C"
>            THEN PUSH DESIGNER COPY
>         IF ACTION_FLAG = "X"
>            THEN PUSH DESIGNER MOVE
>      END
```

Creating Your Own Actions

In addition to the standard Entry, Find, and Select modes, you can design custom actions that assist users with commonly-performed or tedious routines. In the following example, custom actions make short work of changing an employee's address or position.

The MY_PUSH_FLAG condition allows the FIND command unless the ADDR procedure was executed by the XFER procedure (below). If the latter is true, then the employee has already been found by the XFER procedure's FIND command.

The XFER procedure executes the ADDR procedure because it assumes that if the employee has been transferred, then he/she has also had a change of address.

```
> SCREEN XMPL6 ACTIONBAR STARTUP
> FILE EMPLOYEES
> TEMPORARY MY_PUSH_FLAG CHAR*4
```

```
.
.
.
> ACTIONMENU LABEL "Actions"
>   MENUITEM LABEL "Add Employee" ACTION ENTRY
>   MENUITEM LABEL "Find Employee" ACTION FIND
>   MENUITEM LABEL "Change Employee's Address"  &
>         ACTION DESIGNER ADDR
>   MENUITEM LABEL "Transfer Employee"     &
>         ACTION DESIGNER XFER
.
.
.
> FIELD EMPLOYEE
.
.
.
> FIELD STREET      ;ID 4
> FIELD CITY        ;ID 5
> FIELD STATE       ;ID 6
> FIELD ZIP         ;ID 7
.
.
.
> FIELD BRANCH      ;ID 10
> FIELD POSITION    ;ID 11
.
.
.
> PROCEDURE DESIGNER ADDR NODATA    &
>       PRECOMMAND FIND IF MY_PUSH_FLAG <> "XFER" &
>       POSTCOMMAND UPDATE
>   BEGIN
>     PUSH ID 4 TO 7   ;IDs for street, city, state, zip
>   END
>
> PROCEDURE DESIGNER XFER NODATA    &
>       PRECOMMAND FIND
>   BEGIN
>     LET MY_PUSH_FLAG = "XFER"
>     PUSH ID 10 TO 11, DESIGNER ADDR ;IDs for branch
>                                     ;and position
>   END
```

Partial-Index Retrieval in QUICK

Data records from indexed files can be retrieved by index, by partial-index (generically), or by indexed sequential access. To perform partial-index retrieval, the index must be a character-type item. When an index has more than one segment, generic retrieval can be used on any combination of the character segments of the index. Partial-index retrieval is available directly to the user. If the user enters M@ in an index field, all data records will be retrieved where the index begins with the letter M. If the user enters M@@, all data records will be retrieved where the index begins with letter M to the highest value (that is the last segment value). The default generic retrieval character is the at-sign (@).

You can also program retrieval, either by setting the last nonblank character of the search value to @ or @@, or by defining, in the data dictionary, an index that is shorter than the actual file index and starts in the same position. Partial-index retrieval can be disabled with the NOGENERIC option. To perform sequential retrieval on a specific index, the VIA and SEQUENTIAL options are used together.

In Find mode, when QUICK is prompting for index values for an indexed file, @@ can be added to a value used for partial-index retrieval. The @@ may make the entered value too large for the field; if this is the case, the entry is rejected with an error message. For example, an entry of GL@@ should retrieve data records from an indexed file from the first index value starting with GL, and continuing upwards through the alphabet. If the linkitem field was only two characters long, the entire @@ is cropped, leaving the two characters GL. Generally, for the use of @@ to be effective, the linkitem field length must be at least two positions greater than the partial-index value. However, if @@ is concatenated to a value in the expression of a USING option, the retrieval works correctly, and @@ is not truncated or dropped.

Limitation on Retrieval from B-Tree Indexes (MPE/iX)

The following limitation exists on B-Tree indexes.

If the set of records being retrieved is from a wildcard, ranged or partial key search, the "super chain" of records that satisfies the search criteria can be corrupted by a directed read from the dataset. Results of reads from the super chain after a directed read can be unpredictable.

Since PowerHouse uses directed reads as part of its Update process, the following actions are taken to prevent corrupted reads:

- For Master datasets, the Find sequence is stopped if any item in a record is changed.
- For Detail datasets, the Find sequence is stopped if the value of the key that was used for retrieval is changed.
- For Detail datasets, the Find sequence is stopped if, after changing an item other than the retrieval key, PowerHouse finds that another user has changed the key used for retrieval in the last record read from the super chain between the time the record was originally read and the time PowerHouse tries to read the next record from the super chain.

For Detail datasets, PowerHouse will attempt to reestablish its position after a directed read and continue reading.

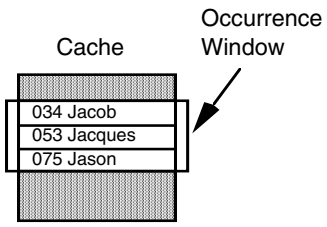
Scrolling Primary and Detail Records

QUICK uses a cache to scroll forwards and backwards through primary and detail record-structures. The cache allows both screen designers and end users to work with more records than can be displayed on the screen.

When a record-structure is cached, QUICK stores in memory all of its record buffers as well as the record buffers for the record-structures associated with the cached record-structure.

You use the CACHE option on the FILE statement to specify how many record buffers for a primary or detail record-structure are to be cached.

You use the OCCURS option on the FILE statement to display multiple occurrences of a record-structure on a screen. This occurrence window serves as a window into the cache. The number of records in the cache can be greater than or equal to the occurrence window. If this option is not used, then the occurrence window size is one. If the CACHE option is not specified, then the cache is set to the number of occurrences. If neither is specified, then both the cache size and the occurrence window size is one.



Screen Designer Options

In addition to the CACHE and OCCURS option on the FILE statement, the PRESCROLL and POSTSCROLL procedures and the FOR control structure are available:

Procedure	Description
PRESCROLL	Provides designer control before QUICK performs any scrolling.
POSTSCROLL	Provides designer control after QUICK performs the scrolling but before the screen is refreshed.
FOR	Accesses all records in the cache. The MISSING option accesses the unused entries in the cache, and the DISPLAY option accesses only those cached records currently displayed on the screen.

For more information about these procedures, see (p. 293).

For more information about the FOR control structure, see (p. 425).

Screen User Commands

Screen users use the following Action commands for scrolling forward and backward through the set of records in the cache:

FIRST RECORD	LAST RECORD
NEXT	NEXT DATA
NEXT RECORD	
PREVIOUS DATA	PREVIOUS RECORD

Cache Contents

When a SECONDARY record-structure occurs with a cached record-structure, the SECONDARY record-structure is cached as well; for every record buffer of the cached record-structure, there is an associated record buffer for the SECONDARY record-structure. The same is true for any SECONDARY record-structure that occurs with the implied cached SECONDARY record-structure. The cached secondary records do not affect the cache limit specified by the CACHE n option.

Any other record-structure that is linked to the cached record-structure is also cached. This includes DELETE, AUDIT, and DESIGNER record-structures as well as temporary items.

Scrolling Commands

The commands for scrolling through the cache are dependent on whether the CACHE option was specified for the PRIMARY or DETAIL record-structure and whether the OCCURS option was used.

Cached PRIMARY Record-Structure

With a screen that has a cached PRIMARY record-structure

```
> FILE ...PRIMARY OCCURS n CACHE
```

or

```
> FILE ...PRIMARY CACHE
```

then the table below describes what QUICK does when the following QUICK commands are issued:

QUICK Command	Action
NEXT DATA	Scroll the occurrence window down one full window length. On the QUICK screen, you will see the next set of n data records. If the OCCURS option has not been specified, then this command is the same as NEXT RECORD.
NEXT	Same as Next Data.
NEXT RECORD	Scroll forward one PRIMARY record.
PREVIOUS DATA	Scroll the occurrence window up one full window length. On the QUICK screen, you will see the previous set of n data records. If the OCCURS option has not been specified, then this command is the same as PREVIOUS RECORD.
PREVIOUS RECORD	Scroll backwards one PRIMARY record.
FIRST RECORD	Move the occurrence window to the top of the cache. On the QUICK screen, you will see the first n data records in the cache.
LAST RECORD	Move the occurrence window to the bottom of the cache. On the QUICK screen, you will see the last n data records in the cache.

Cached DETAIL Record-Structure

With a screen that has a cached DETAIL record-structure

```
> FILE ...PRIMARY
```

or

```
> FILE ...DETAIL OCCURS n CACHE
```

then the table below describes what QUICK does when the following QUICK commands are issued:

QUICK Command	Action
NEXT DATA	Scroll the occurrence window down one full window length. In Find mode, and there are insufficient data records in the cache, read the necessary DETAIL records from the file. If there are no more DETAIL data records, advance to next PRIMARY data record and clear the cache.
NEXT	Clear the cache and advance to the next PRIMARY data record.
NEXT RECORD	Scroll forward one DETAIL record.
PREVIOUS DATA	Scroll the occurrence window up one full window length. On the QUICK screen, you will see the previous set of n detail data records. If at the beginning of the cache, display first records from the cache. If attempt to scroll back past the beginning of the cache, display warning message saying no more records.
PREVIOUS RECORD	Scroll backwards one DETAIL record.
FIRST RECORD	Move the occurrence window to the top of the cache. On the QUICK screen, you will see the first n DETAIL data records in the cache.
LAST RECORD	Move the occurrence window to the bottom of the cache. On the QUICK screen, you will see the last n DETAIL data records in the cache.

Screen Threads

Separate screen threads let an application have more than one screen hierarchy active at the same time. For example, the following code generates a system of screens in which users can browse parts in the Parts screen while entering orders in the Orders screen:

```
> SCREEN MENU
>   THREAD ORDERS
>   THREAD PARTS
.
.
.
```

Screen threads can be used with QUICK in the host environment or with Axiant 4GL in the client/server environment.

You use the THREAD statement and RUN THREAD verb to specify a new screen thread. They specify the screen to be loaded as the root of a new screen thread. They are functionally equivalent to the SUBSCREEN statement and RUN SCREEN verb except that no passing or receiving lists are possible. If the SHARED option is specified and the thread already exists, then you go to that screen without creating a new instance of the screen.

Once a new thread is started, you can move from one thread to another using the Toggle command (T).

For more information about threads, see (p. 221) and (p. 478).

Chapter 3: QDESIGN Statements

Overview

This chapter provides a detailed reference of QDESIGN statements. For each statement you'll find

- formal syntax
- syntax summaries
- detailed syntax descriptions
- detailed discussions
- examples

QDESIGN syntax that is specific to PowerHouse Web or that has differences in the context of PowerHouse Web is described in the Chapter 7 of the PowerHouse Web Developer's Guide.

Summary of QDESIGN Statements

The following table summarizes the purpose of each QDESIGN statement:

Statement	Screen Section	Purpose
ACCESS	Data	Specifies or overrides record-structure access methods.
ACTIONMENU	Screen	Specifies the action taken by an Action bar item.
ALIGN	Layout	Changes the default positioning of objects on a screen.
BUILD	n/a	Compiles the current screen design, optionally listing generated procedures.
CANCEL	n/a	Cancels the screen design specifications.
CLUSTER	Layout	Groups a set of screen entities.
COMMAND	Layout	Executes an operating system command or runs a program.
CURSOR	Data	Identifies and describes how a cursor, table, or view is used by the screen.
[SQL] DECLARE CURSOR (query-specification)	Data	Defines a set of data as a run-time view.
[SQL] DECLARE CURSOR (stored procedure)	Data	Calls a stored procedure.
DEFINE	Data	Assigns a name to an expression.
DESCRIPTION	Screen	Allows screen designers to enter extended help messages for a screen or a field.

Statement	Screen Section	Purpose
DRAW	Layout	Draws lines and boxes on a screen.
EXIT	n/a	Ends a QDESIGN session.
FIELD	Layout	Creates fields on the screen that correspond to items for data entry and display.
FILE	Data	Identifies and describes a record-structure accessed by the screen.
GENERATE	Layout	Generates FIELD statements based on dictionary definitions of record-structures specified in FILE statements.
GO	n/a	Runs QUICK from QDESIGN.
HILITE	Layout	Assigns highlighting features to screen entities.
ITEM	Data	Assigns values to items or performs sums and balances on items.
KEY	Screen	Specifies a dynamic function key (DFK).
MENUITEM	Screen	Specifies the action taken by a drop-down menu item.
QSHOW	n/a	Runs QSHOW from QDESIGN.
query-specification (SELECT)	Data	Defines a collection of rows that will be accessible when the cursor is opened.
QUIT	n/a	Terminates QDESIGN.
REPORT	Layout	Executes QUIZ.
REVISE	n/a	Invokes an editor to edit files from within QDESIGN.
RUN	Layout	Executes QTP.
SAVE	n/a	Saves QDESIGN source statements in a permanent file.
SCREEN	Screen	Names the screen and specifies its characteristics.
SELECT	Data	Applies a selection condition to retrieved data records.
SET	n/a	Changes default settings for a QDESIGN session.
SHOW	n/a	Displays available record-structures and/or items as defined in the data dictionary.
SKIP	Layout	Skips lines to a specific line or to an alignment group.
SUBSCREEN	Layout	Invokes a lower-level screen.
TARGET	Data	Calculates the record number in a direct or relative file for storing a newly-created record.
TEMPORARY	Data	Creates a temporary item that is not defined in the data dictionary.
THREAD	Layout	Specifies a screen thread.

Statement	Screen Section	Purpose
TITLE	Layout	Positions text on the screen.
TRANSACTION	Data	Defines transactions used for relational files.
USE	n/a	Processes QDESIGN source statements contained in a file.

ACCESS

Specifies or overrides record-structure access methods.

Syntax

ACCESS [option]...

Options

ACCESS Options		
BACKWARDS	GENERIC|NOGENERIC	OPTIONAL
ORDERBY	REQUEST	SEQUENTIAL
sql-substitution	UNIQUE	USING
VIA	VIAINDEX	

BACKWARDS

Reverses the sequence in which the data records are normally read.

Limit: Valid only for C-ISAM, DISAM, RMS ISAM, and IMAGE datasets with keyed access.

Limit: The BACKWARDS and SEQUENTIAL options cannot be used together for RMS ISAM files.

GENERIC|NOGENERIC

GENERIC allows partial-index retrieval. NOGENERIC prevents partial-index retrieval.

Limit: Not valid for IMAGE indexes, unless they are B-Tree or OMNIDEX indexes.

Default: GENERIC

OPTIONAL

Continues processing even if the access fails. If no data record is found, QUICK creates a data record containing initial values for each item. These values are taken from the data dictionary and any ITEM statements. If no initial values are specified in the data dictionary or an ITEM statement, character items are initialized to spaces, and numeric and date items are initialized to zero.

Limit: Not valid for PRIMARY files.

ORDERBY item [ASCENDING|DESCENDING] [,item [ASCENDING|DESCENDING]]...

Allows the ordered retrieval of records in a relational table or view by any column (or combination of columns) defined in the table or view.

If the ORDERBY option occurs with the VIAINDEX option, ordering is performed according to the columns of the ORDERBY option and the ordering imposed by the VIAINDEX option is ignored.

If the ACCESS statement is associated with a CURSOR statement, ORDERBY is used to build a substitution value for the ORDERBY substitution-variable. If there is no ORDERBY substitution-variable on the DECLARE CURSOR for the table, then PowerHouse attempts to find the right place for one.

Note: Some relational databases have sorting restrictions for certain datatypes. For sorting restrictions, refer to your relational database reference manual.

Limit: Valid only for relational files.

Default: ASCENDING

REQUEST field [,field]...

Prompts the screen user for a value in each named field. An entered value can be used as an index value, or as a means to calculate such a value. An entered value can also be used as a value in a selection condition.

Limit: Valid only for record-structures in PRIMARY files.

SEQUENTIAL

Accesses the record-structure sequentially.

Limit: Valid only for record-structures in PRIMARY files.

Limit: The USING and SEQUENTIAL options can't be used in the same ACCESS statement. The VIA and SEQUENTIAL options are compatible in the same ACCESS statement for indexed files only.

Limit: The BACKWARDS and SEQUENTIAL options cannot be used together for RMS ISAM files.

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 VIA and USING options are used to build a substitution for the default substitution-variable: WHERE. The ORDERBY and ORDERED options are used to build a substitution for the default substitution-variable: ORDERBY.

The syntax for an sql-substitution is:

substitution-variable (text)

For more information about substitutions and substitution-variables, see Chapter 1, "About PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

Limit: Any sql-substitutions must appear before any other options.

UNIQUE

Forces a re-evaluation of the USING expression and a "get first" access for each data record read. UNIQUE overrides chained-type access to indexed files.

MPE/iX, OpenVMS: For direct files, and relative files, UNIQUE allows calculation of the data record number for each individual data record.

Limits: Valid for PRIMARY, DETAIL, and SECONDARY files. Not valid when the ACCESS statement applies to a CURSOR statement.

USING expression [,expression]...

The USING, VIA, and VIAINDEX options control data retrieval.

The USING option accesses an associated file using the results of a specified expression as

- the value for corresponding segments
- **MPE/iX, OpenVMS:** the data record number for record-structures in direct files or relative files
- the column value in a relational table

For direct files or relative files (**MPE/iX, OpenVMS**), there can be only one value which QUICK interprets as a record number.

For indexed files or IMAGE databases (**MPE/iX**), there can be more than one value (for segmented indexes), but QUICK interprets the values as a single index value in that file.

If the record-structure belongs to a direct file, there can only be one expression specified, which must be numeric. Otherwise, a series of expressions can be specified which correspond one-to-one with the segments established by either the VIA or VIAINDEX options. If neither the VIA nor the VIAINDEX option is specified, and the record-structure has only one associated index, this index is used as if the VIAINDEX option had been specified except that index retrieval order will not be enforced.

If a record-structure is a relational table, there can be several values in the USING option which QDESIGN interprets as the values of the columns in the table. The VIA or VIAINDEX options must be used to indicate which columns the values belong to if more than one index is in use or if no index is used.

If the VIA option is specified, the number of expressions specified must correspond one-to-one with the number of linkitems specified on the VIA option.

If the VIAINDEX option is specified and the VIA option isn't specified, the number of expressions specified may be less than or equal to the number of segments contained within the specified index. There must always be at least one expression.

Due to the logic of NULL value processing, placing NULL in a USING expression for any segment should be avoided, as it will never result in a match.

Limit: 255 expressions.

Limit (MPE/iX): IMAGE does not support retrieval via an initial subset of the segments of a multi-segment index, unless the index is a B-Tree or OMNIDEX index. An expression must be specified for every segment of the index.

VIA linkitem [,linkitem]... [ORDERED[ASCENDING|DESCENDING]]

The USING, VIA, and VIAINDEX options control data retrieval.

Accesses the record-structure via the specified linkitem. A linkitem is an item declared as a segment of an index declared in the data dictionary, or a column in a table of a relational database declared in the data dictionary.

When a VIA list is used in combination with the USING option, there must be a one-to-one match between the USING expressions and VIA linkitems. This option is valid for indexed files, IMAGE databases (**MPE/iX**), and relational tables only.

For indexed files, and IMAGE databases (**MPE/iX**), the series of linkitems declared must define a series of segments contained within the index structure associated with the record-structure. In this case, the first linkitem is the first segment within the index structure, the second linkitem is the second segment, and so on.

For relational tables, a series of linkitems may represent any series of columns in a table as long as the VIAINDEX option is not specified. If VIAINDEX is specified, a series of linkitems must be a series of segments contained within a specific index structure: match the first linkitem to the first segment, the second linkitem to the second segment, and so on.

If the ACCESS statement is associated with a CURSOR statement, the ORDERED option is used to build a substitution value for the ORDERBY substitution-variable and the VIA options are used to build substitution values for the WHERE substitution-variable. If there is no ORDERBY variable on the DECLARE CURSOR for the table, then PowerHouse attempts to find the right place for one.

Limit: 255 segments.

Limit (MPE/iX): IMAGE does not support retrieval via an initial subset of the segments of a multi-segment index, unless the index is a B-Tree or OMNIDEX index. The series of linkitems must include all of the segments in the index.

ORDERED[ASCENDING|DESCENDING]

Allows the ordered retrieval of records in a relational table or view by the columns specified in the VIA option (or combination of columns) defined in the table or view.

The ORDERED option is a convenient method of specifying ORDERBY items when the items to be specified are the same as those in the VIA list.

If the ORDERED option occurs with the VIAINDEX option, which also imposes an ordering, the ordering is done by the columns of the VIA option. The implicit ordering imposed by the VIAINDEX option is ignored.

Limit: 255 segments. ORDERED is valid only when the VIA option is used with one or more linkitems.

Default: ASCENDING

VIAINDEX indexname

The USING, VIA, and VIAINDEX options control data retrieval.

The VIAINDEX option names an index of an indexed file, IMAGE database (**MPE/iX**), or relational table. When VIAINDEX is used with the USING option, there can be as many USING values as there are segments in the index, or fewer values than the index segments. In the latter case, the values are matched to the index segments in order, starting from the first segment; the leftover segments are not used. When using VIAINDEX, the retrieval always follows the order specified by that index.

Use VIA instead of VIAINDEX with relational tables. By explicitly referencing an index with the VIAINDEX option, it becomes harder to change the database definitions. If the index is deleted, then the source code must be modified. If VIA is used instead, the index can be deleted and the screen continues to work properly.

Limit: Not valid when the ACCESS statement applies to a CURSOR statement.

Discussion

The ACCESS statement is part of the data section of your screen design. The ACCESS statement specifies record-structure access and can be used to alter the default PATH and FIND procedures QDESIGN normally generates. Using the ACCESS statement is preferable to modifying the PATH and FIND procedures directly.

Where to Enter the ACCESS Statement

ACCESS statements must follow the FILE or CURSOR statement for the record-structure to which they refer, and must come before any other FILE, CURSOR, DEFINE, or TEMPORARY statements.

Specifying Multiple ACCESS Statements

You can specify multiple ACCESS statements for the PRIMARY file. QDESIGN constructs PATH and FIND procedures based on all specified ACCESS statements, in the order in which they occur. This allows you to create QUICK screens that, in Find mode, prompt first for the segments of one index, then for the segments of a second index, and so on until the screen user enters a non-null response to one of the prompts. QUICK then uses the index that corresponds to the segments for which values were entered.

Effects of the ACCESS Statement on Record Retrieval

Once you include an ACCESS statement for a record-structure, QDESIGN leaves all access specifications up to you. For record-structures in primary files, if you don't include an ACCESS statement for a given index, that index isn't available for data record retrieval if you specified an ACCESS statement for another index. In the same way, if you don't include an ACCESS statement for sequential access after you specified an ACCESS statement, sequential retrieval isn't available in Find mode for that QUICK screen.

The USING and SEQUENTIAL options are incompatible in the same ACCESS statement. The VIA and SEQUENTIAL options are compatible in the same ACCESS statement for indexed files only.

Linking Record-structures with the ACCESS Statement

If there is only one segment, QDESIGN tries to match its value to a value of that item. If the record-structure accessed has more than one index, use the VIA or VIAINDEX option to indicate which segment QDESIGN should use to establish the linkage.

For more information on record retrieval, see (p. 137).

Retrieving Data in QUICK

QUICK determines retrieval information for data records of a file according to the following steps:

1. QUICK looks for either procedural retrieval specifications on the GET verb, or a LOOKUP option on the FIELD statement.
2. If Step 1 fails, QUICK looks for retrieval specifications from ACCESS statements associated with the file (for non-PRIMARY record-structures only).
3. For indexed record-structures, QDESIGN assumes indexed retrieval and the minimum required specification is the linkitem name. If the record-structure has only one index, QDESIGN assumes indexed retrieval using that index. Nothing more is required.

   If there is more than one index, QDESIGN looks for generated (by automatic item initialization) or specified ITEM statements with the INITIAL and FIXED options for the linkitems of the record-structure. The first such ITEM statement found determines the index used to retrieve the data records of the file. The index value used for retrieval is the value of the linkitem at the time the retrieval is attempted.

The retrieval specifications referred to in steps 1 and 2 allow you to control the way that QUICK retrieves data records. These retrieval specifications are:

BACKWARDS	GENERIC|NOGENERIC	OPTIONAL
REQUEST	SEQUENTIAL	UNIQUE
USING	VIA	VIAINDEX

Note that the REQUEST option is valid only in the ACCESS statement, and not in any of the other statements specified above in steps 1 and 2. All of the other retrieval specifications are valid in each of the statements specified in steps 1 and 2.

If any of these retrieval specifications are used in step 1, they will override the same option on any ACCESS statement; however, any other options will still be used.

Controlling Data Record Retrieval Explicitly in QUICK

There are four ways in which you can use QDESIGN syntax to override or alter QUICK's assumptions about data record retrieval for certain record-structures. You can use

- a NOSEQUENTIAL option on the SCREEN statement
- ACCESS statements with BACKWARDS, NOGENERIC, OPTIONAL, REQUEST, SEQUENTIAL, UNIQUE, USING, VIA, and VIAINDEX options
- LOOKUP options with BACKWARDS, NOGENERIC, OPTIONAL, SEQUENTIAL, USING, VIA, and VIAINDEX options
- procedures

Ascending/Descending Index Support

You can specify segments of an index in either ascending or descending order. When PowerHouse generates database retrieval requests as the result of an explicit VIAINDEX index, it generates the sorting specification in the request to match the order declared when the index was defined.

Example

ACCESS statements control how the CUSTOMERS file is accessed in Find mode.

The following examples illustrate how to use the ACCESS statements to control how QUICK accesses files and cursors.

QUICK prompts first for the segment CUSTOMERKEY of the CUSTOMERS index, then for the segment CUSTOMERNAME of the CUSTOMERNAME index. If no entries are made in either field, then QUICK retrieves CUSTOMERS data records sequentially.

```
> SCREEN MODCUST
>
> FILE CUSTOMERS PRIMARY
>   ACCESS VIAINDEX CUSTOMERS &
>     USING CUSTOMERKEY OF CUSTOMERS &
>     REQUEST CUSTOMERKEY
>   ACCESS VIAINDEX CUSTOMERNAME &
>     USING CUSTOMERNAME OF CUSTOMERS &
>      REQUEST CUSTOMERNAME
>   ACCESS SEQUENTIAL

> SQL DECLARE EMPLIST CURSOR FOR &
>  SELECT EMPLOYEE, FIRST_NAME, LAST_NAME, &
>  BRANCHES.BRANCH, BRANCH_NAME &
>  FROM EMPLOYEES, BRANCHES &
>  WHERE EMPLOYEES.BRANCH = BRANCHES.BRANCH
>  SCREEN EMPBRANCHC
>  CURSOR EMPLIST PRIMARY KEY EMPLOYEE
>  ACCESS VIA EMPLOYEE REQUEST EMPLOYEE
>  ACCESS VIA LAST_NAME REQUEST LAST_NAME
>  ACCESS SEQUENTIAL
```

Sequential access could have been disabled by omitting the ACCESS SEQUENTIAL statement following the first two ACCESS statements.

For an example of how the ACCESS statement influences the generated PATH procedure, see (p. 333).

When retrieving records, the ORDERED option ensures that if you specify a LAST_NAME value, and a generic FIRST_NAME value, the records will be displayed for a particular LAST_NAME in order of FIRST_NAME.

The following example illustrates how to use the ORDERED option to control the order that QUICK will display data records in. When retrieving records, the ORDERED option ensures that if you specify a LAST_NAME value, and a generic FIRST_NAME value, the records will be displayed for a particular LAST_NAME in order of FIRST_NAME.

```
> SCREEN EMPLOYEE_DISPLAY
>
> FILE EMPLOYEES IN EMPL_DB PRIMARY OCCURS 4
>   ACCESS VIA EMPLOYEE &
>     REQUEST EMPLOYEE
>   ACCESS VIA LAST_NAME, FIRST_NAME ORDERED &
>     REQUEST LAST_NAME, FIRST_NAME
>   ACCESS SEQUENTIAL
> TITLE &
>   "   EMPL. NO     SURNAME    FIRST NAME    START DATE"
> CLUSTER OCCURS WITH EMPLOYEES
> ALIGN (1,,4) (,,16) (,,30) (,,53)
> CLUSTER OCCURS WITH EMPLOYEES
> FIELD EMPLOYEE OF EMPLOYEES REQUIRED NOCHANGE &
>   LOOKUP NOTON EMPLOYEES
> FIELD FIRST_NAME OF EMPLOYEES REQUIRED NOCHANGE
> FIELD LAST_NAME OF EMPLOYEES REQUIRED NOCHANGE
> FIELD DATE_JOINED OF EMPLOYEES REQUIRED NOCHANGE
> BUILD
```

ACTIONMENU

Specifies the action taken by an Action bar item.

Syntax

ACTIONMENU [option]...

Options

The options are ACTION, LABEL, MENUKEY and NOMENUKEY.

ACTION conditional-command-list

Specifies what command(s) the Action bar item executes and, optionally, under what conditions.

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. The general form of a command list is:

command [, command]...

For a list of the available commands, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

condition

A condition is a logical test that has the general form:

[NOT] condition [AND|OR [NOT] condition]...

For more information about conditions or conditional command lists, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

LABEL string

Displays a specified string on the Action bar.

A string is a series of displayable characters (letters, numbers, or special characters) in double or single quotation marks. The string is aligned to the left margin of the Action bar and separated from any previous string by one space.

The Action bar always extends across the entire terminal window. Each ACTIONMENU statement adds a new label that extends across the screen from left to right. One blank precedes and follows each label.

By default, the Action bar is positioned on the same line as the first screen line. You can change the position by using the ON LINE option of the ACTIONBAR screen option.

Default: If no LABEL is specified, QUICK provides a default label of 5 spaces.

{MENUKEY char}|NOMENUKEY

MENUKEY assigns a short-cut menu key that users can press to select an Action bar item when they're in the Action bar. MENUKEY overrides default menu keys applied by the MENUKEYS option of the SET statement.

The menu key character should be unique among all the items in the Action bar. If the menu key character is not a character in the item label, then it is displayed in brackets at the end of the label.

Limit: The character can be an uppercase letter, a lowercase letter, or a number, but not a special character.

NOMENUKEY ensures that no menu key is assigned to an Action bar item, even if the MENUKEYS option of the SET statement is used.

Menu keys are highlighted with an underline by default. You can change the highlighting by using the HILITE statement.

For more information about adding and customizing menu keys for your menu-driven QUICK screens, see (p. 147), (p. 164), and (p. 199).

Discussion

The ACTIONMENU statement is part of the screen section of your screen design. The ACTIONMENU statement, together with the MENUITEM statement and ACTIONBAR option of the SCREEN statement, enable Action bar definition. By including a Return action in the ACTIONBAR, you can leave the screen. For more information, see Chapter 2, "QUICK User Interface".

Action Bars

An Action bar presents a list of QUICK actions or menus in a menu bar that extends across the terminal window. You can invoke an Action command by selecting the appropriate entry in the Action bar or pull down menu as an alternative to entering the action at the Action field prompt.

With an Action bar, you can associate commands with a more descriptive label, and also show all the commands that are available to the user.

Adding an Action Bar to Your Screen Design

To create an Action bar:

1. Specify the ACTIONBAR option of the SCREEN statement.
2. Use the ACTIONMENU statement to define the menus and actions you want to place on the Action bar.
3. Use the MENUITEM statement to define the actions for each menu to be pulled down from the Action bar.

Action Bar Menus and Actions

The Action bar can contain both menus and actions. To specify a menu, enter the ACTIONMENU statement without indicating an action option, as in

```
> ACTIONMENU LABEL "EMPLOYEES"
```

To specify the action that's performed, use the ACTION option, as in

```
> ACTIONMENU LABEL "CREATE" ACTION ENTRY
```

You can specify any QUICK Action command as an ACTION option. If the action requires an ID-number or ID-number range as a parameter, you can specify these explicitly or you can obtain the ID-numbers at run time by using the MARK or PROMPT options, as in

```
> ACTIONMENU LABEL "PROMPT FOR ID" ACTION ID PROMPT
> ACTIONMENU LABEL "MARK" ACTION ID MARK
```

If you specify PROMPT, QUICK prompts you for ID-number values with a prompt box. If you specify MARK, QUICK determines the value from the current FIELDMARK setting. If there is no current MARK and MARK has been specified, QUICK prompts for the ID-numbers.

If you don't want your Action bar to have pull-down menus, exclude MENUITEM statements from your design statements. Instead, you can specify Action commands in the Action bar.

But if you want to specify a pull-down menu, enter MENUITEM statements after the corresponding ACTIONMENU statement, as in:

```
> ACTIONMENU LABEL "EMPLOYEES"
> MENUITEM LABEL "LOCATE" ACTION FIND
> MENUITEM LABEL "NEW" ACTION ENTRY
```

If you specify MENUITEM statements following the ACTIONMENU statement, QDESIGN constructs a menu that appears to "pull down" from the Action bar when the Action menu is selected.

Each MENUITEM statement creates a new entry on the menu. The menu can accommodate any size label and any number of items: the menu border adjusts to accommodate the largest menu label, and the menu scrolls to fit the number of menu items.

Generally, menus pull down from the Action bar; that is, they appear below the corresponding Action menu on the Action bar. However, if you specify that the Action bar appear on the last three lines of the terminal, the menus pop up above the Action bar.

Action Bars and Action Fields

You can specify that both an Action bar and an Action field appear on a screen. When the screen runs, only one mode is active at a time, but you can switch between the two modes. This way, you can make a limited set of commands available through an Action bar for the novice user but still enable an Action field for the more experienced user.

If both the Action bar and Action field are available, the screen initially runs Action field prompting mode by default. To specify that the Action bar should be the initial mode, add the STARTUP option to the ACTIONBAR option of the SCREEN statement.

If the Action field and Action bar appear on the same line, only the current prompting mode is visible. If they are not on the same line, both the Action bar and Action field are visible, but QUICK only prompts you at the active mode.

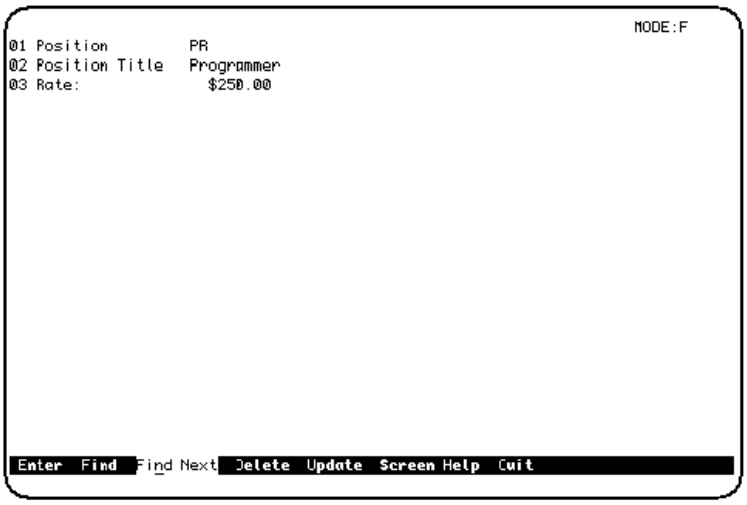
Example

The statements in the following example create an Action bar complete with menu keys for each ACTIONMENU item. When you select an option and enter the Accept command, or press a menu key, QUICK executes the associated action.

The warning message means that since you've positioned the Action bar at the bottom of the screen, and there's no room for drop-down menu items to drop down, they'll drop "up" instead. However, since this screen has no MENUITEM statements, the message can be ignored.

```
> SCREEN POSITIONSMAINTENANCE  &
>    NOACTION      &
>    MODE AT 1,70  &
>    FIELDMARK     &
>    MESSAGE ON LINE 23  &
>    ACTIONBAR ON LINE 24
*W* Menus will appear on screen lines preceding ACTIONBAR line.
> ACTIONMENU LABEL "Enter" ACTION ENTRY   &
>    MENUKEY "E"
> ACTIONMENU LABEL "Find" ACTION FIND   &
>    MENUKEY "F"
> ACTIONMENU LABEL "Find Next" ACTION NEXT DATA  &
>    MENUKEY "N"
> ACTIONMENU LABEL "Delete" ACTION DELETE  &
>    MENUKEY "D"
> ACTIONMENU LABEL "Update" ACTION UPDATE   &
>    MENUKEY "U"
> ACTIONMENU LABEL "Screen Help" ACTION EXTENDED HELP  &
>    MENUKEY "H"
> ACTIONMENU LABEL "Quit" ACTION RETURN   &
>    MENUKEY "Q"
.
.
.
```

The resulting screen looks like this:



In the previous example, the ACTIONBAR option of the SCREEN statement specifies that the POSITIONS_MAINTENANCE screen has an Action bar, and that the Action bar appears on line 24. Each ACTIONMENU statement defines the label that appears on the Action bar and the corresponding Action command.

The statements in the next example create an Action bar with pull-down menus. When you select one of the commands on the Action bar and press [Accept] QUICK presents you with another list of commands. Move the highlight to the command you want and use the Accept command to execute it.

```
> SCREEN ADDRESS_PULLDOWN &
>   NOACTION &
>   MODE AT 23,60 &
>   ACTIONBAR ON LINE 1
>
> ACTIONMENU LABEL "Help" ACTION EXTENDED HELP
>
> ACTIONMENU LABEL "Search"
>   MENUITEM LABEL "Next" ACTION NEXT DATA
>   MENUITEM LABEL "Find" ACTION FIND
>   MENUITEM LABEL "Select" ACTION SELECT
>
> ACTIONMENU LABEL "Update"
>   MENUITEM LABEL "Stay" ACTION UPDATE STAY
>   MENUITEM LABEL "Next" ACTION UPDATE NEXT
>   MENUITEM LABEL "Return" ACTION UPDATE RETURN
>   MENUITEM LABEL "Update" ACTION UPDATE
>
> ACTIONMENU LABEL "Exit" ACTION RETURN
```

ALIGN

Changes the default positioning of objects on a screen.

Syntax

ALIGN [alignment-group]...

alignment-group

Sets the horizontal position of the ID-number, label, and data positions of subsequent fields. The general form of an alignment group is:

([[**ID**] **column1**] [,[[**LABEL**] **column2**] [,[[**DATA**] **column3**]]])

The ID, LABEL, and DATA keywords are used only for documentation. As a result, the specification for an alignment group can be shortened to:

([**column1**] [,[**column2**] [,[**column3**]]])

Limit: Column numbers can range from 1 to 131 (you must have a 132 column terminal to successfully use column values greater than 80). There can be a maximum of 20 alignment groups. An alignment group must contain at least one number.

Default alignment: (ID 1, LABEL 4, DATA 21) or (1,4,21)

Discussion

The ALIGN statement is part of the layout section of your screen design. It governs the default positioning of everything that follows it in the screen design until either another ALIGN statement or the end of the layout section is encountered. The first entity positioned after an ALIGN statement is positioned at the start of the first alignment group.

Suppressing Field ID-Numbers and Labels

You can suppress the LABEL or DATA portions of a field by omitting column numbers in an alignment group. To omit a column number in an alignment group, include commas to delimit the omissions.

For example, the statement

```
> ALIGN (,,25)
```

suppresses the display of both ID-numbers and labels for all fields that follow, and positions the DATA portion of the fields in column 25.

Similarly, the statement

```
> ALIGN (12,,15)
```

sets just the ID and DATA portions of the alignment group, but the statement

```
> ALIGN (12,15,30)
```

sets the ID, LABEL, and DATA portions.

If the ID-number is suppressed, the ID SAME option is assumed. If the LABEL is suppressed, the NOLABEL option is assumed.

Specifying the ALIGN statement without options resets the alignment to the default values.

Using Multiple Alignment Groups

With multiple alignment groups, fields are positioned from left to right across the screen until all groups are filled. Once a line is filled, fields are positioned from left to right on the next line.

Other Positioning Attributes

The ALIGN statement doesn't override specific positioning (specified with the AT option) in COMMAND, FIELD, SUBSCREEN, and TITLE statements. An alignment group is only used to fill in positions that are not specified.

Although it is possible to use the ID, LABEL, and DATA options on the FIELD statement, the ALIGN statement is easier to change.

Example

The following example shows you how to position fields using the ALIGN statement. In this example:

- The first ALIGN statement suppresses the ID of the fields.
- The second ALIGN statement includes the ID, label, and data.
- The third ALIGN statement suppresses the labels from the fields.

The QUICK screen defined by the following statements re-creates the alignment that's found in an employee skills profile.

```
> SCREEN NEWEMP &
>   MODE AT 1, 70 FIELDMARK &
>   MESSAGE ON LINE 23 ACTIONBAR ON LINE 2
> ACTIONMENU LABEL "next" ACTION NEXT DATA
> ACTIONMENU LABEL "delete" ACTION DELETE
> ACTIONMENU LABEL "edit" ACTION FIELDMARK
> ACTIONMENU LABEL "entry" ACTION ENTRY
> ACTIONMENU LABEL "find" ACTION FIND
> ACTIONMENU LABEL "select" ACTION SELECT
> ACTIONMENU LABEL "update" ACTION UPDATE
> ACTIONMENU LABEL "quit" ACTION RETURN
>
> FILE EMPLOY1 PRIMARY
> FILE SKILLS DETAIL OCCURS 5
> TITLE "Employment And Skills Information" AT ,25
> SKIP 2
> ALIGN (,1,15)
> FIELD EMPLOYEE OF EMPLOY1 REQUIRED NOCHANGE &
>    LOOKUP NOTON EMPLOY1
> FIELD LASTNAME OF EMPLOY1 REQUIRED NOCHANGE
> FIELD FIRSTNAME OF EMPLOY1
> SKIP TO LINE 8
> TITLE "Employment Info" AT ,5
> DRAW FROM 9,5 TO 9,20
> SKIP 2
> ALIGN (1,5,20)
> FIELD JOINEDYEAR OF EMPLOY1
> FIELD JOINEDMONTH OF EMPLOY1
> FIELD JOINEDDAY OF EMPLOY1
> FIELD DATELEFT OF EMPLOY1
> FIELD BRANCH OF EMPLOY1 REQUIRED NOCHANGE
> FIELD DIVISION OF EMPLOY1 REQUIRED NOCHANGE
> FIELD POSITION OF EMPLOY1 REQUIRED NOCHANGE
> FIELD DATEAPPOINTED OF EMPLOY1
> FIELD NOOFAPPTS OF EMPLOY1
> SKIP TO LINE 8
>
> TITLE "Skills" AT ,53
> DRAW FROM 9,53 TO 9,58
> SKIP 2
>
> ALIGN (50,,55)
> CLUSTER OCCURS WITH SKILLS
> FIELD SKILL OF SKILLS
> CLUSTER
```

BUILD

Compiles the current screen design, optionally listing generated procedures.

Syntax

BUILD [DETAIL|NODETAIL] [LIST|NOLIST]

DETAIL|NODETAIL

DETAIL writes the procedures that are generated by this design to the temporary source statement save file. NODETAIL doesn't. The save file is QKSAVE (**MPE/iX**) or qksave.qks (**OpenVMS, UNIX, Windows**).

Default: NODETAIL

LIST|NOLIST

LIST lists the procedures generated by this design; NOLIST doesn't.

Default: NOLIST

Discussion

The BUILD statement completes the screen design and performs the following steps:

1. Constructs the required default procedures.
2. Saves the compiled screen in a file named in the SCREEN statement.
3. If accessing an ALLBASE/SQL database, constructs and compiles SQL statements.
4. Displays a visual representation of the screen's layout (unless SET NOLIST LAYOUT is in effect).

Implicit Screen Building with the GO Statement

If a screen is executed using the GO statement, the DETAIL, LIST, NODETAIL, and NOLIST options are not valid on a subsequent BUILD statement for the screen. QDESIGN can't write or list the procedures since they have already been generated by a GO statement and are not regenerated by the BUILD statement.

Example

This example uses the LIST and DETAIL options to display and save the procedures that are generated by the screen design. In this example, when the BUILD LIST DETAIL statement is entered, QDESIGN generates procedural code that can then be saved in the source file.

```
> SCREEN SKILLMNT
>
> FILE SKILLS PRIMARY
>
> FILE SKILLDT DETAIL OCCURS 10
>
> FIELD EMPLOYEE OF SKILLS REQUIRED NOCHANGE
> FIELD SKILL OF SKILLS REQUIRED NOCHANGE
> CLUSTER OCCURS WITH SKILLDT
> FIELD SKILLDESC OF SKILLDT
> CLUSTER
> BUILD LIST DETAIL
>
>
> PROCEDURE APPEND
>   BEGIN
>     ACCEPT SKILLDESC OF SKILLDT
>     END
> PROCEDURE ENTRY
>   BEGIN
```

```
>     ACCEPT EMPLOYEE OF SKILLS
>     ACCEPT SKILL OF SKILLS
>     FOR SKILLDT
>       BEGIN
>         PERFORM APPEND
>         END
>     END
> PROCEDURE PATH
>   BEGIN
.
.
.
```

CANCEL

Cancels the screen design specifications.

Syntax

CANCEL [CLEAR]

CLEAR

Removes any source statements in the temporary source statement save file once the screen design specifications are canceled. The CANCEL statement doesn't clear the source statement save file unless you include the CLEAR option.

The save file is QKSAVE (**MPE/iX**) or qksave.qks (**OpenVMS, UNIX, Windows**).

Limit: The CANCEL statement doesn't cancel SET statement options.

Discussion

Clearing the Source Statement Save File

The CLEAR option of the CANCEL statement ensures that erroneous statements are not saved in the temporary save file. This is important when saving source code to permanent files. Using the SAVE statement saves the entire temporary save file. If this file isn't cleared by the CLEAR option of the CANCEL statement, all statements entered (including those with errors) are saved.

Example

This example shows how the CANCEL statement is used to correct errors in design statements.

```
> SCREEN NEWORD
>
> FILE SKILLS PRIMRY
               ^^^^^^
*E* Expected: @ . IN <eol> OCCURS NEED PRIMARY SECONDARY DETAIL MASTER DESIGNER
DELETE REFERENCE AUDIT ALIAS NOAPPEND AUTOCOMMIT NODELETE OPEN MYVIEW NOITEMS
CLOSE COUNT CACHE TRANSACTION GLOBAL
```

Screen designers commonly forget to clear QDESIGN's temporary save file, qksave. All statements before the CANCEL CLEAR are purged from QDESIGN's temporary save file. The SAVE statement saves all statements entered since the CANCEL CLEAR and copies everything in QDESIGN's temporary save file to a permanent file called, in this example, NEWORDS.

```
> CANCEL CLEAR
>
> SCREEN NEWORD
>
> FILE SKILLS PRIMARY
> GENERATE NOLIST NODETAIL
>
> SAVE NEWORDS
```

CLUSTER

Groups a set of screen entities.

Syntax

CLUSTER [option]...

Options

CLUSTER Options		
AT	BLOCK EACH|ALL	FOR
HIDDEN	ID|NOID	MARK|NOMARK
OCCURS	VERTICAL	

AT [line],column

Positions the first occurrence of the cluster at the specified line and column on the screen. If the line is missing, the current line is assumed. The starting column establishes the upper left-hand corner of the cluster boundary.

BLOCK EACH|ALL

Specifies how QDESIGN generates BLOCK TRANSFER constructs in the default procedures for fields within a cluster occurrence.

BLOCK EACH treats each cluster occurrence as a distinct block of information. The fields within each occurrence are treated as a single block of information. BLOCK ALL treats all of the cluster occurrences as a single block of information.

For screens with the PANEL option, if neither BLOCK EACH nor BLOCK ALL is specified, then a single BLOCK TRANSFER control structure is generated for the entire screen in the default procedures. QUICK processes the cluster (for Panel input) as though a BLOCK ALL option were present. For more information, see (p. 372).

Default: When no BLOCK option is specified on the CLUSTER statement, a panel input screen treats the cluster as if a BLOCK ALL option was included.

Limits: Valid only when the PANEL option is specified for either the SET or SCREEN statement.

FOR [lines],[columns]

Establishes the boundaries for one occurrence of the cluster in lines and columns.

Default: Without the FOR option, QDESIGN assigns a width equal to the screen width, and the number of lines occupied by the cluster for the length.

HIDDEN

Suppresses the screen ID display, but lets the user reference the field by ID-number.

ID [BASE m] [INCREMENT m] [AT [line],column]
ID n [AT [line],column]
ID NEXT [AT [line],column]
NOID

Control the manner in which QDESIGN assigns ID-numbers to cluster occurrences.

ID [BASE m] [INCREMENT n]

Sets the starting field ID-number in the first occurrence of the cluster to m. Optionally, the starting field ID-number in lines and subsequent occurrences can be incremented by n. One of the options must be specified. ID and NOID are mutually exclusive.

Limit: 1 to 99

ID n

Groups all occurrences of the cluster under the common ID-number specified by the value of n.

Limit: 1 to 99

ID NEXT

Groups all occurrences of the cluster under the next available ID-number.

AT [line], column

Positions the first digit of the ID-number at the specified line and column relative to the starting line of the screen. If the line isn't specified, the current line is assumed.

NOID

Specifies that none of the components of the cluster have ID-numbers assigned to them and can't be referenced from the Action field.

MARK|NOMARK

MARK enables fieldmarking for a cluster with an ID-number.

NOMARK disables the default fieldmarking for a cluster with an ID-number when fieldmarking is enabled.

OCCURS n
OCCURS WITH record-structure|item

Controls how many occurrences of the cluster appear on the QUICK screen.

n

Repeats the cluster a specified number (n) of times.

Limit: 1 to 255

WITH record-structure|item

Repeats the cluster as many times as the named record-structure or item.

In this case, QDESIGN uses the number specified for the OCCURS option of the corresponding FILE statement.

The general term item specifies the name of either a record item or temporary item. If a record item is named, QDESIGN uses the number specified for the OCCURS option of the ITEM statement in the data dictionary. If a temporary item is named, QDESIGN uses the number specified for the OCCURS option of the TEMPORARY statement.

The OCCURS WITH option functions differently from the OCCURS n option even if the number of occurrences is the same. The OCCURS n option always processes n occurrences, but the OCCURS WITH option processes only currently active occurrences of the specified file.

VERTICAL

Numbers the clusters from top-to-bottom (starting with the upper left-hand cluster) rather than from left-to-right.

Discussion

The CLUSTER statement is part of the layout section of your screen design. The CLUSTER statement groups sets of screen entities, including

- fields (including field labels and ID-numbers)
- titles and other screen literals
- statements that control screen layout, such as the SKIP and ALIGN statements

- SUBSCREEN and COMMAND statements
- lines created with the DRAW statement

As part of the screen layout section, you can use clusters to repeat groups of entities in a screen design.

Grouping Fields on a QUICK Screen

When you include a CLUSTER statement with no options in your screen design, the fields in the resulting cluster don't repeat. A cluster with no options doesn't change the current cursor position and is ended by a subsequent CLUSTER statement.

A CLUSTER statement with no options allows you to use the DRAW statement and screen layout options, such as SKIP and ALIGN, on specific groups of fields, uninhibited by the limits normally imposed by CLUSTER statement options.

Using the SKIP Statement Within a Cluster

You can use a SKIP statement to establish the starting line of a cluster, but you must enter the SKIP statement before the CLUSTER statement, as in

```
> SKIP TO 7
> CLUSTER OCCURS WITH BRANCHES
```

If a SKIP statement follows a CLUSTER statement, QDESIGN considers it to be one of the layout statements in the cluster. You can use this feature to insert blank lines between cluster occurrences.

You can't position a cluster with a SKIP TO statement that references a line before the current cursor position (unless you are in a cluster with no options).

Cluster Facts

Clusters follow these principles:

- All entities defined in statements and included between the CLUSTER statement and the next CLUSTER statement (or the end of the layout section) are part of the cluster.
- Entities defined in statements that lie outside the cluster definition are not part of the cluster.
- When a statement is outside the cluster, but places an entity within the cluster boundaries, the entity isn't repeated with the cluster.
- QDESIGN issues an error if a screen entity is defined within the cluster, but the statement declaring the entity places it outside the cluster boundary.
- By default, all fields within a given cluster boundary are treated as a single block. Fields within blocks are delimited by the BLOCK TRANSFER control structure in QDESIGN's generated procedures.
- Clusters can't be nested. If you want to show repeating items within a record-structure, you can only include the record-structure once in the screen design.
- The space that the cluster occupies on the screen defines the cluster size. The cluster size defines the boundary of one occurrence of the cluster.

ID-Numbers for Cluster Occurrences

If no ID-number is specified, an ID-number is assigned to each component of the cluster in the same way that the ID-numbers are assigned to other design entities. If a NOID, ID n, or ID NEXT option is used, all layout statements within the cluster are assumed to have the same ID-number.

Side-by-Side Clusters

You can create side-by-side clusters by using the ALIGN statement and the CLUSTER statement together. The ALIGN statement sets up alignment groups that QDESIGN uses when creating clusters, if the entities in the clusters can fit within the boundaries of the alignment group.

Examples

The following examples illustrate how QDESIGN uses CLUSTER statements to group fields together.

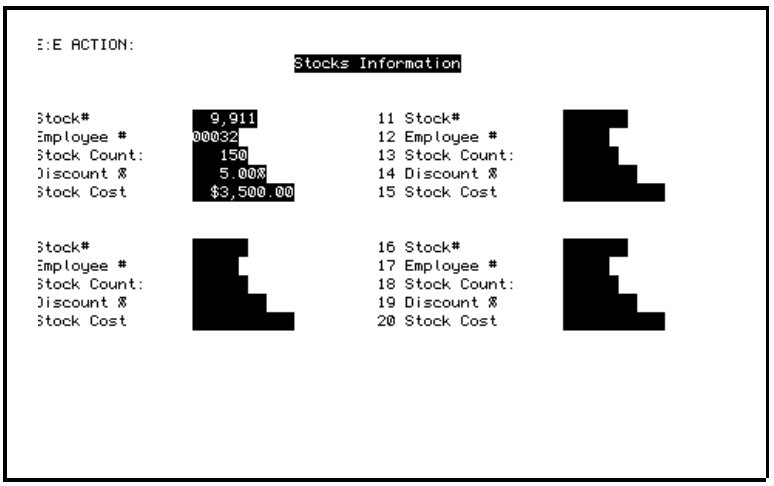
Using the CLUSTER Statement for a Repeating Primary Record-structure

In this example:

- OCCURS WITH builds a cluster based on the items in the STOCKS record-structure.
- FOR causes the cluster to occupy 7 rows and 40 columns.

```
> SCREEN STOCK
>
> FILE STOCKS OCCURS 4
>
> HILITE DATA INVERSE
> HILITE TITLE INVERSE
> TITLE "Stocks Information" CENTERED
> SKIP 2
>
> CLUSTER OCCURS WITH STOCKS FOR 7,40 VERTICAL
> FIELD STOCKNUM OF STOCKS REQUIRED NOCHANGE
>   LOOKUP NOTON STOCKS
> FIELD EMPLOYEE OF STOCKS
> FIELD STOCKCOUNT OF STOCKS
> FIELD DISCPCT OF STOCKS
> FIELD STOCKCOST OF STOCKS
>
> SKIP 2
> CLUSTER
```

The resulting screen looks like this:



Using the CLUSTER Statement for Repeating Items

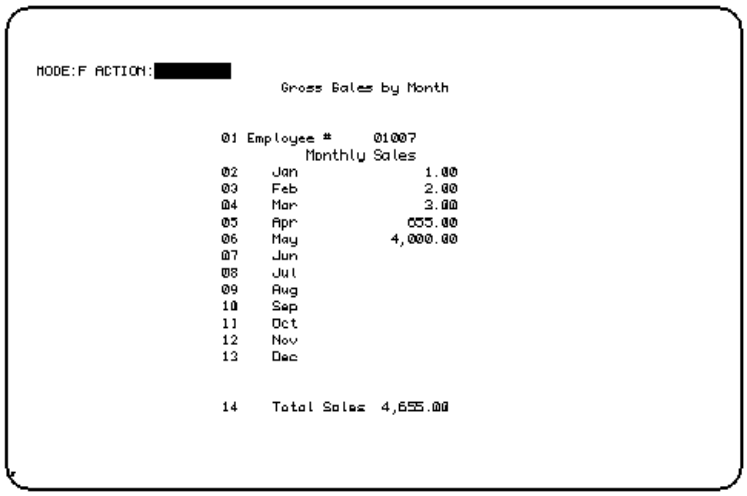
Group repeating fields on a screen by preceding the FIELD statement for that item with the CLUSTER OCCURS WITH item statement. Clustered items must be defined in the data dictionary as having more than one occurrence.

In this example, MONTHLYGROSS repeats 12 times (once a month) in the record SALES.

```
> SCREEN SALE
>
> FILE SALES
>
> TITLE "Gross Sales by Month" CENTERED
> SKIP 2
>
> ALIGN (ID 23, LABEL 26, DATA 41)
> FIELD EMPLOYEE OF SALES REQUIRED NOCHANGE &
>   LOOKUP NOTON SALES
> SKIP TO 5
> CLUSTER OCCURS WITH MONTHLYGROSS OF SALES AT 7,1
> FIELD MONTHLYGROSS OF SALES NOLABEL
```

```
> CLUSTER
> TITLE "Monthly Sales" at 6,33
> TITLE "JAN" AT 7,29
> TITLE "FEB" AT 8,29
> TITLE "MAR" AT 9,29
> TITLE "APR" AT 10,29
> TITLE "MAY" AT 11,29
> TITLE "JUN" AT 12,29
> TITLE "JUL" AT 13,29
> TITLE "AUG" AT 14,29
> TITLE "SEP" AT 15,29
> TITLE "OCT" AT 16,29
> TITLE "NOV" AT 17,29
> TITLE "DEC" AT 18,29
> SKIP 1
> ALIGN (23,29,41)
> FIELD TOTALSALES OF SALES
```

The resulting screen looks like this:



COMMAND

Executes an operating system command or runs a program.

Syntax

COMMAND stringlitem [option]...

string|item

Specifies the command to execute. The command can be represented either by a string or a character-type item.

Limit: 255 COMMAND statements per QUICK screen.

Options

COMMAND Options		
AUTO	CLEAR	HIDDEN
ID|NOID	[ENTRY] IF	INPUT B|C|SAME
LABEL|NOLABEL	MARK|NOMARK	NOCONSOLE
NOWARN	ON ERROR	REFRESH
RESPONSE	WAIT|NOWAIT	

AUTO

Invokes the named command automatically when the standard entry sequence reaches this statement, if the ENTRY procedure was generated by QDESIGN.

CLEAR option

Clears an area of the terminal memory before invoking the command. Any output to the terminal from the command appears starting on the first line of the cleared area. Lines cleared are refreshed automatically when the screen is reactivated and QUICK is ready to prompt the user. If multiple COMMAND, REPORT, or RUN statements are combined with ID SAME, the area cleared is refreshed after the last statement in the chain has completed execution and QUICK is ready to prompt the user.

ALL

Clears the entire terminal memory.

SCREEN

Clears the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears the area between and including lines n to m, numbering from the first line of terminal memory. LINES n by itself clears line n only.

HIDDEN

Suppresses the screen ID display, but lets the user reference a field by ID-number in MARK mode.

ID n [AT [line],column]
ID NEXT [AT [line],column]
ID SAME
NOID

Control how and where QUICK screen field ID-numbers are assigned.

ID n

Explicitly specifies an ID-number.

Limit: 1 to 99

ID NEXT

Uses the next ID-number in sequence.

AT [line],column

Positions the first digit of the ID-number at the specified line and column relative to the starting line of the screen. If the line is missing, the current line is assumed.

ID SAME

Instructs QDESIGN to omit the ID-number on the command. To execute the command, use the ID-number of the previous field.

NOID

States that no ID-number is assigned to this command and the command can't be referenced from the Action field.

[ENTRY] IF condition

Invokes the specified command in the standard entry sequence only if this condition is satisfied. QDESIGN generates an identical IF condition in the default ENTRY procedure.

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

Limit: The IF option is evaluated only during the standard entry sequence; otherwise, it is ignored.

Default: If the condition is satisfied, AUTO is assumed.

INPUT B|C|SAME (MPE/iX)

Puts the terminal in the specified input mode prior to executing the command. The terminal is put back into the original mode after completion of the command.

Default: The mode it was in before QUICK was invoked.

B

Puts the terminal in Block mode. This should only be used for commands that must run in Block mode.

C

Puts the terminal in Character mode.

SAME

Leaves the terminal in the current input mode. If the screen has Block mode capability, this option should only be used for commands that do not write to the terminal.

LABEL [string] [AT [line],column] | NOLABEL

Declares the label and its position.

[string] [AT [line],column]

Indicates the command label and, optionally, the position of the label on the screen.

The AT option positions the first character of the label at the specified line and column relative to the starting line of the screen. If the line isn't specified, the current line is assumed.

Default: The item name or the first word in the COMMAND string.

NOLABEL

Specifies that no label is to appear for the command.

MARK|NOMARK

MARK enables fieldmarking for a command with an ID-number.

NOMARK disables the default fieldmarking for a command with an ID-number when fieldmarking is enabled.

NOCONSOLE (Windows)

Suppresses opening a Command Console window. Normally QUICK opens a second Command Console window to run the command. If the command runs in the background, does not require user input, or does not display useful output, the second command console window may not be necessary.

NOWARN

Specifies that operating system warning messages issued during the execution of the command should not be displayed.

ON ERROR CONTINUE|TERMINATE

Specifies the action to be taken if an operating system error occurs during the execution of a command. If TERMINATE is in effect, an operating system error causes QUICK to process the error as it would for an ERROR verb. If CONTINUE is specified, an operating system error is ignored and processing continues as if the error had not occurred.

For information on the ERROR verb, see (p. 420).

Default: TERMINATE

REFRESH option

Clears and rewrites an area of the terminal memory when the screen is reactivated and QUICK is ready to prompt the user. REFRESH options are performed before, and in addition to, an automatic refresh from any CLEAR option.

ALL

Clears and rewrites the entire terminal memory.

SCREEN

Clears and rewrites the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears and rewrites the area between and including lines n to m, numbering from the first line of terminal memory. LINES n alone refreshes line n only.

RESPONSE

Prompts the QUICK user for a response after the command has executed. QUICK resumes processing after the user responds, preventing the screen from being refreshed immediately.

WAIT|NOWAIT (Windows)

The WAIT option instructs QUICK to suspend current screen processing until the command has executed, at which time control returns to the screen. The NOWAIT option specifies that screen processing continues immediately and the command executes concurrently.

Default: NOWAIT

Discussion

The COMMAND statement is part of the layout section of the screen design. It makes a program or an operating system command (which can execute a program) available to the QUICK screen user.

As in the FIELD and SUBSCREEN statements, the operating system command can be labeled and given a position and ID-number on the screen.

The **noosaccess** and **nodcl** (**OpenVMS**) program parameters have no affect on the COMMAND statement. That is, you will not be prevented from executing an operating system command or running a program using the COMMAND statement even if QUICK has been invoked using **noosaccess** or **nodcl**.

UNIX, Windows: The command runs in a subprocess from the main QUICK process, however it starts a separate shell (UNIX) or command (Windows). Hence the results of a setenv (UNIX) or set (Windows) command will not be accessible from QUICK or any later commands. For this, use the SETSYSTEMVAL function.

OpenVMS: The **notrusted** program parameter will disable the use of the COMMAND statement and cause an error to occur if one is executed.

Example

The following example creates a menu screen that gives a screen user an option for generating summary reports in QUIZ from within a QUICK screen using the COMMAND statement. The **auto** program parameter tells QUIZ which report to run.

```
> SCREEN PERMENU MENU
> DRAW FROM 3,13 TO 7,67
> SKIP TO 4
> TITLE "Future Industries" CENTERED
> SKIP TO 6
> TITLE "Personnel System - Main Menu" CENTERED
>
> SUBSCREEN STAFF LABEL "Staff Screen"
> SUBSCREEN PROJEC LABEL "Projects Screen"
> SUBSCREEN BRNCHS LABEL "Branches Maintenance"
> SUBSCREEN DIVISNS LABEL "Positions Maintenance"
>
> COMMAND 'quiz auto=staff' LABEL &
>  "Report Staff by Project"
> BUILD
```

CURSOR

Identifies and describes how a cursor, table, or view is used by the screen.

Syntax

CURSOR cursor-name|tablespec [sql-substitution...] [option]...

Limit: A maximum of 31 files, record-structures, and cursors can be declared in a screen design. There can be a maximum of 1023 columns per cursor.

cursor-name

The name of a cursor defined by the PowerHouse SQL DECLARE CURSOR statement.

tablespec

The name of a table or view declared in a relational database.

The general syntax for tablespec is:

MPE/iX, OpenVMS:	**[owner.]table-name [IN database]**
UNIX, Windows:	**[[server-name.]database-name.] [owner-name.]table-name** If server-name is included in a Sybase tablespec, double quotes are required for the server-name and database-name. For example, `"dbsvr01.accnt".manager.billings_tbl`

For Oracle, the syntax is:

[owner-name.]table-name[@database-linkname]

If the database-linkname is included, it is treated as part of the table-name, and double quotes are required. For example,

```
manager."billings_tbl@dblnk01"
```

Oracle synonyms may be used for table-names. For more information about how PowerHouse uses Oracle synonyms, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

sql-substitution

An sql-substitution can be specified for any substitution variable defined on the DECLARE CURSOR statement. Two default 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)

Limit: Any sql-substitution must appear before any other options.

For more information, see Chapter 1, "About PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

Options

CURSOR Options		
type	ALIAS	AUTOCOMMIT
CACHE	COUNT	KEEP

CURSOR Options		
KEY	NOAPPEND	NODELETE
NOITEMS	OCCURS	OPEN
TRANSACTION		

type

Specifies the relationship of the cursor to the screen and to other files and cursors on the screen.

Type must be one of:

AUDIT [WITH record]	DELETE	DESIGNER
DETAIL	MASTER	PRIMARY
REFERENCE	SECONDARY	

Default: If no type is given, PRIMARY is assumed, except for cursors that are included in the receiving list of the SCREEN statement. Cursors that are passed from higher-level screens are always assumed to be MASTER.

For detailed information about these types, see the FILE statement on (p. 126).

ALIAS name

Assigns an alternative name to the cursor. When a cursor is declared more than once in a screen design, the ALIAS option assigns a unique identifier name for each declaration. Once the alias is assigned, subsequent references to the cursor must use this name.

AUTOCOMMIT

Indicates that the transaction performing the retrieval from the reference file is automatically committed after the retrieval is completed. Automatic retrievals include retrievals from lookups or implicit retrievals of reference items for display.

Limit: Valid when type is REFERENCE only.

CACHE [n]

Specifies that QUICK is to maintain more primary or detail record buffers than can be displayed on the screen. These record buffers may be accessed programmatically by the screen designer and browsed by the screen user.

If you do not specify CACHE, the cache size is set to the number of occurrences as specified with the OCCURS option for this record structure. If the OCCURS option is not used, the size of the cache is set to one.

Limit: This option may only be used with either the primary record-structure or a detail record-structure but not both.

n

Specifies the upper limit for the number of record buffers in the cache. If you are concerned that the size of the record combined with the associated detail and secondary records will use excessive memory, use this option to specify an upper limit.

If not specified, QUICK sets the size based on the current requirements and will grow dynamically up to 255 record buffers.

The minimum size of the cache is the number specified on the OCCURS option. If this option is not used, the minimum size is one.

Limit: 1 to 255

COUNT [NEGATIVE] [INTO] [ITEM] item [,[NEGATIVE] [INTO] [ITEM] item]...

Uses the named items to maintain a count of the data records entered into this file. The named items should normally be in record structures in higher-level MASTER files, so the proper value is maintained from one screen to the next.

The count is automatically incremented when data records are entered and reduced when data records are deleted. For record structures in DELETE files, the count is decremented when the record structure is actually updated (that is, when the PUT verb is executed). The NEGATIVE option reverses these activities. INTO is used only for documentation.

Limit: The maximum number of items that can be counted into is 21.

KEEP [CURSOR]

This option is obsolete.

KEY column-name [,column-name]...

Allows you to identify a column or set of columns that uniquely identify a row in the table. All column-names must be in the project-list of the cursor declaration.

If the KEY option is not specified, sequential access is used. A warning message is issued indicating that there is no unique key available for re-retrieval. If there is no key available and a row of the table is updated, the entire table will be updated.

NOAPPEND

Suppresses the automatic generation of the APPEND procedure and PERFORM APPEND verb for a record structure in a repeating PRIMARY file. If this option is specified, Append processing can't be used for the record structures in the repeating PRIMARY file.

Limit: Valid only for record structures in PRIMARY files.

NODELETE

Suppresses the automatic generation of a DELETE verb for this record structure in the DELETE and DETAIL DELETE procedures.

NOITEMS

Doesn't generate automatic item initialization.

OCCURS n [TIMES]
OCCURS WITH [ITEM] item|[FILE]record

Repeats the data records on the screen.

n [TIMES]

Repeats the data records the specified number of times on the screen.

Limit: 1 to 255

WITH [ITEM] item|[FILE]record

Repeats the data records of this file as many times as the named data record or item repeats on the screen.

OPEN [READ|UPDATE]

OPEN READ opens a data structure for read access only. OPEN UPDATE opens a data structure for read and write access.

TRANSACTION transaction_name [FOR {CONSISTENCY| {[CONCURRENCY] phase-option[,phase-option]...}]...

Defines transactions used for relational data structures.

transaction_name

Any valid PowerHouse name.

FOR CONSISTENCY

Determines that a relational data structure is associated with a particular transaction in Consistency model.

Limit: Only one transaction association can be specified.

FOR [CONCURRENCY] phase-option [,phase-option]...

Determines that the relational data structure is associated with a particular transaction or transactions in Concurrency model.

Limit: Up to three transaction associations can be specified, one per phase.

phase-option

Specifies the screen phase with which the transaction is associated. The phase-options are:

PROCESS	The phase in which you are entering, correcting, or changing data records on the screen.
QUERY	The phase in which data is retrieved from the database.
UPDATE	The phase in which data is updated.

By default, all relational data structures are associated as follows:

Model	Transaction	Phase
Concurrency	Query	Query[1]
	Update	Process
	Update	Update
Optimistic	Query	Query[2]
	Update	Update[3]
Consistency	Consistency	Consistency
Dual	Query	Query
	Update	Process
	Update	Update
	Consistency	Consistency

Model	Transaction	Phase

[1]*Note that for a screen allowing only the activity FIND, by default read-only data structures are associated with the Query transaction for all phases.*

[2]*All "read" activities are associated with the transaction associated with the Query phase. By default, this is the Query transaction.*

[3]*All "write" activities are associated with the transaction associated with the Update phase. By default, this is the Update transaction.*

For ALLBASE/SQL and Oracle, the Update transaction is used for all activities in the Concurrency and Optimistic models.

Discussion

When using the CURSOR statement, generated procedural code includes SQL code. Relational tables can be accessed through FILE statements but no corresponding SQL code is generated and SQL can't be used.

When a CURSOR statement refers to a DECLARE CURSOR statement that defines a join, QDESIGN only generates data manipulation commands in the UPDATE procedure for the first table in the join. If it is not possible to determine what this table is (i.e. derived table), then you get a syntax error.

Retrieval Assumptions for Cursors

If nothing else is specified, cursor retrieval is sequential. This is true regardless of the type of cursor. In other words, there is no assumed linkage between cursors based on index segment name matching, as there is for files. To relate one cursor to another (for example a DETAIL cursor to a PRIMARY cursor), specify the retrieval criteria in the DECLARE CURSOR statement or include an ACCESS statement.

Scrolling Records

The CACHE option specifies that QUICK creates a cache of record buffers to store retrieved and entered data records. As data records are found, QUICK automatically moves data records in and out of the cache. As the size of the cache may be larger than the number of occurrences displayed on the screen (as controlled by the OCCURS n option), users may scroll backwards and forwards through the cache, viewing previous data records that have been scrolled off the screen.

For a cached primary record structure, users can scroll backwards and forwards through primary records retrieved by the FIND procedure. Scrolling is also supported for primary data records entered by way of the ENTRY and APPEND procedures.

For a cached detail record structure, users can scroll backwards and forwards through the detail records retrieved by the DETAIL FIND procedure associated with an individual primary record structure. Scrolling is also available for detail records entered by way of the APPEND procedure.

For more information about cached records, see (p. 57).

Example

In the following example, the CURSOR statement describes the role of the EMPLOYEES table on the screen. The table can be qualified with an ownername and the IN database option.

```
> SCREEN EMPC
> CURSOR OWNER.EMPLOYEES IN EMPBASE PRIMARY &
>    KEY EMPLOYEE
```

In the next example, the WHERE substitution specified on the CURSOR statement is inserted in the generated SQL SELECT even though a ::WHERE substitution-variable does not exist on the cursor declaration. The code preceded by ___ is displayed when SET LIST SQL is used.

```
> SET LIST SQL
> SQL DECLARE EMPLIST CURSOR FOR &
```

```
>   SELECT EMPLOYEE, FIRST_NAME, LAST_NAME, &
>     EMPLOYEES.BRANCH,BRANCH_NAME &
>     FROM EMPLOYEES, BRANCHES
> SCREEN EMPBRANCHC
> CURSOR EMPLIST &
>   WHERE (EMPLOYEES.BRANCH = BRANCHES.BRANCH) &
>   PRIMARY KEY EMPLOYEE
> __ Sql after substitutions are applied:
__ SELECT EMPLOYEE, FIRST_NAME, LAST_NAME,
__ EMPLOYEES.BRANCH, BRANCHES.BRANCH,
__ BRANCH_NAME
__     FROM EMPLOYEES, BRANCHES
__     where EMPLOYEES.BRANCH = BRANCHES.BRANCH
```

[SQL] DECLARE CURSOR (query-specification)

Defines a set of data as a run-time view.

Syntax

[SQL[IN database]]
DECLARE name CURSOR FOR
query-specification [UNION [ALL] query-specification...]
[ORDER BY sort-specification]

IN database

Specifies the name PowerHouse uses to attach to the database. This is the name used to declare the database in PDL.

DECLARE name

Defines a logical name used to identify the set of data resulting from the query.

Limit: The name must be unique within the scope of the cursor. For a description of a cursor's scope, see (p. 94).

query-specification [UNION [ALL] query-specification...]

The query-specification defines a collection of rows that will be accessible when the cursor is opened.

The ALL option on the UNION option indicates that redundant duplicate rows are retained; otherwise, they are eliminated.

Parentheses are used in a union of three or more query specifications to enforce precedence in eliminating duplicate rows in the unioned sets. For example, a union of the three query-specifications X, Y and Z, must be written as (X UNION Y) UNION Z or X UNION (Y UNION Z).

For more information, see (p. 169).

ORDER BY sort-specification

The sort-specification syntax is:

{columnspec|n} [ASC|DESC][,{columnspec|n} [ASC|DESC]]...]

The columnspec must identify a column of the project-list. The default sort order is ascending.

The integer refers to the position of the column in the project-list. In the following example, the integer 2 refers to the derived column of averages.

```
> SQL DECLARE Y CURSOR &
>  FOR SELECT SP.PNO, AVG(SP.QTY) &
>     FROM SP &
>     GROUP BY SP.PNO &
>  ORDER BY 2
```

If the cursor definition involves a UNION, the sort specification may refer to column names if the corresponding column names of each query specification are identical; otherwise, the sort specification must reference an integer.

Discussion

The Scope of a Cursor

You may declare a cursor before the SCREEN statement or in the data definition section of a screen.

A cursor defined before a SCREEN statement or between a CANCEL and a SCREEN statement is accessible to all screens compiled during the QDESIGN session as long as no CANCEL statement is encountered.

A cursor defined after a SCREEN statement is valid until a BUILD or CANCEL statement is encountered.

Example

```
> SQL IN EMPLOYEESDATABASE &
>   DECLARE EMPSKILLS CURSOR FOR &
>   SELECT EMPLOYEES.ID, EMPLOYEES.FIRSTNAME, &
>   EMPLOYEES.LASTNAME, S.SKILL, &
>   FROM EMPLOYEES, SKILLS S &
>   WHERE EMPLOYEES.ID = S.ID &
>   AND EMPLOYEES.ID IN &
>   (SELECT ID FROM SELECTEDEMPLOYEES)
```

For more information about using cursors, see Chapter 1, "About PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

[SQL] DECLARE CURSOR(stored procedure)

Calls a stored procedure or stored function from the specified database.

Syntax

```
    [SQL [IN database]]
DECLARE name CURSOR FOR
CALL stored-procedure|stored-function
    [([ITEM] item [IN [OUT]]|[OUT]
        [,[ITEM] item [IN [OUT]]|[OUT]]...)]
    [ON ERROR CONTINUE|TERMINATE]
    [RETURNING return-parameter]
    [[RESULT] SET item [,item]...]
```

IN database

Specifies against which database the stored procedure or function is executed.

Limit: Stored procedure calls are valid for DB2, ODBC, Oracle, Oracle Rdb (declared as TYPE RDB in the dictionary), and Sybase databases.Stored function calls are valid only for Oracle databases.

DECLARE name

Defines a logical name used to identify the set of data resulting from the stored procedure.

CALL stored-procedure|stored-function

The name of a stored procedure or stored function in the database.

The syntax for a procedure name varies with the RDBMS. For information on a specific database system, see "Stored Procedures" in the *PowerHouse and Relational Databases* book.

([ITEM] item [IN [OUT]]|[OUT] [,[ITEM] item [IN [OUT]]|[OUT]]...)

Items which are passed to the stored procedure or Oracle stored function, or received from the stored procedure. Input parameters can be temporary, defined, or record items. Output parameters can be temporary or record items.

Blob items may also be used for both input and output parameters when calling an Oracle stored procedure or stored function.

IN

Specifies that the item is an input parameter.

IN OUT

Specifies that the item is both an input and output parameter. The changed values of the input/output parameters are available to PowerHouse when stored procedure execution is complete.

OUT

Specifies that the item is an output parameter. The values of the output parameters are available to PowerHouse when stored procedure execution is complete.

Default: IN

ON ERROR CONTINUE|TERMINATE

Specifies the action to be taken if an SQL statement fails. If the TERMINATE option is in effect, the SQL error causes QUICK to process the error as it would for an ERROR verb. If CONTINUE is specified, the SQL error is ignored and the processing continues as if the error had not occurred.

For information on the ERROR verb, see (p. 420).

Default: TERMINATE

Limit: This option is valid only for Oracle, Oracle Rdb, and Sybase databases.

RETURNING return-parameter

The return-parameter must be defined as a temporary or record item.

For DB2, ODBC, and Sybase, identifies the item that contains the return status from a stored procedure.

For Oracle, identifies the item that contains the value returned by a stored function upon completion of the Oracle stored function.

Limit: Valid for Oracle stored functions but not valid for Oracle stored procedures. For Sybase, the return-parameter must be defined as a 32-bit (4-byte) integer.

[RESULT] SET item [,item]...

The description of the result set returned from the stored procedure. Each item is defined using a name, datatype and, optionally, its size:

name sql-datatype [(n)]

To identify the item datatypes that match your RDBMS datatypes, see "Relational PowerHouse Datatypes" in the *PowerHouse Rules* book.

Limit: This option is valid only for DB2, ODBC, Oracle, and Sybase. Only one result set can be returned from a stored procedure.

Discussion

Stored procedures and stored functions are collections of SQL statements and logic that are stored in a database. Calls to stored procedures can take input parameters from a calling program, and return values for output parameters to a calling program. A stored procedure in DB2, ODBC, Oracle, or Sybase may also return result sets.PowerHouse supports a single result set per execution of a stored procedure.

For information on stored procedures of specific database systems, see "Stored Procedures" in the *PowerHouse and Relational Databases* book.

Examples

In the following example, the DECLARE CURSOR statement declares a cursor, EMPSKILLS, for the stored procedure, SPEMPLOYEESKILLS, that returns a result set consisting of five items (ID, FIRSTNAME, LASTNAME, SKILL, and SKILLLEVEL).

```
> SQL IN EMPLOYEESDATABASE &
> DECLARE EMPSKILLS CURSOR FOR &
>     CALL SPEMPLOYEESKILLS(EMPLOYEEID IN, EMPCOUNT OUT) &
>     RESULT SET ID DECIMAL, &
>         FIRSTNAME VARCHAR(20), &
>         LASTNAME VARCHAR(20), &
>         SKILL CHARACTER(10), &
>         SKILLLEVEL FLOAT
```

This example declares a cursor for the stored procedure spCheckPrice in an Oracle database.

```
> SQL IN PartsDb_ORCL DECLARE Part_Price CURSOR FOR &
>  CALL spCheckPrice (PartNo int IN, Price int OUT)
```

DEFINE

Assigns a name to an expression.

Syntax

DEFINE name [type[*n] [type-option]]
[= conditional-expression|case-processing]

Limit: A maximum of 1023 defined and temporary items can be declared in a screen design.

name

Names the defined item.

You can give a defined item the same name as a record item in any of the record-structures on the FILE statement.

Limit: Must begin with a letter and can't exceed 64 characters.

type[*n]

Establishes the physical format of the defined item.

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

Default: NUMERIC

*n

Specifies the number of characters or digits that can be entered in the defined item.

type-option

Indicates the set of options that further characterize the item datatype. The type options are CENTURY, NUMERIC, SIGNED, UNSIGNED, and SIZE.

CENTURY INCLUDED|EXCLUDED

Indicates whether or not the date will contain a century prefix.

Limit: Valid for defined items of type DATE, JDATE and PHDATE.

Default: For DATE items, the default is determined by the SYSTEM OPTIONS statement in the data dictionary. For PHDATE and JDATE items, the default is CENTURY EXCLUDED.

NUMERIC

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

Limit: Valid only for ZONED datatypes.

SIGNED|UNSIGNED

Indicates whether the datatypes INTEGER, PACKED, and ZONED are SIGNED or UNSIGNED. When the SIGNED option is used with INTEGER, negative values can be stored. A datatype INTEGER with the UNSIGNED option can't store negative values. The datatypes PACKED and ZONED can store positive or negative numbers, whether or not the SIGNED or UNSIGNED option is specified.

Limit: Valid only for INTEGER, PACKED, and ZONED datatypes.

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

SIZE m [BYTES]

Specifies a storage size in bytes.

Use the SIZE m BYTES option when the default size isn't the size that's required for the item.

Limit: Not valid for datatype NUMERIC or G_FLOAT.

conditional-expression

Sets the value of the defined item based on a condition.

A DEFINE statement without a conditional-expression is used to receive defined items and the expressions they represent from higher-level screens.

A conditional-expression specifies an expression that, when evaluated, results in the value of the defined item. The expression is calculated every time the defined item name is referenced during execution.

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

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. The comparison is calculated once for every record complex when the data to be evaluated is available. If there is a match, 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 or spaces are assigned. The general form 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]

When the 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.

A colon may be substituted for the THEN keyword. The OF keyword is optional and is for documentation only. When the type of the defined item is character, the resulting value must be a string enclosed in quotation marks, as in

```
> DEFINE PROJECTNAME CHARACTER*20 = &
>   CASE OF PROJECTCODE &
>   WHEN 1001 THEN "PRODUCTION" &
>   WHEN 1002 THEN "PROMOTIONS" &
>   DEFAULT "UNKNOWN"
```

When the type of the defined item is numeric or date, the resulting value must be numeric.

value-set

Specifies one or more values and/or one or more ranges of values. The general form is:

value [TO value][[,] value [TO value]]...

The values assigned to the defined item by the CASE option must be of the same type as the defined item. For example, if you create the defined item PROJECT_NAME and specify that it is a character-type item, you must assign a string to the item PROJECT_NAME:

```
> DEFINE PROJECT_NAME CHARACTER*20  &
>   = CASE OF EMPLOYEES  &
>   WHEN 1001 THEN "PRODUCTION"  &
>   WHEN 1002 THEN "PROMOTIONS"
```

Limit: Case-processing and conditional-expressions cannot be used together in any combination.

Discussion

The DEFINE statement is part of the data section of a screen design. It allows you to dynamically calculate the value of an expression. The expression is evaluated every time the defined-item name is referenced during execution. You can include the DEFINE statement anywhere in the data section, as long as it doesn't reference items that aren't yet declared. The defined item can be declared as a display field which defaults to ID SAME.

Passing Defined Items between Screens

To pass a defined item to a lower-level screen, use a DEFINE statement without a conditional-expression on that lower-level screen. Passing defined items allows you to access the expression from a lower-level screen without having to redeclare it. If a type isn't specified, type NUMERIC is assumed. The value obtained when the defined item name is referenced on the lower-level screen is the result of the evaluation of the expression defined on the higher-level screen.

Example

The following example uses the DEFINE statement to ensure that the proper invoice total is entered into INVOICE data records. The DEFINE statement multiplies the value entered for PRICE with the value entered for QUANTITY and then adds the TAX amount.

```
> SCREEN INVOICE
>
> FILE INVOICES OCCURS 9
> FILE STOCK REFERENCE
>
> DEFINE DOUBLECHECK NUMERIC*8 = &
>  PRICE * QUANTITY + TAX
>
>
> SKIP 1
>
.
.
.
> CLUSTER OCCURS WITH INVOICES
>   FIELD INVOICENO REQUIRED &
>     NOCHANGE &
>     LOOKUP NOTON INVOICES
>   FIELD STOCKNO REQUIRED &
>     LOOKUP ON STOCK
>   FIELD PRICE REQUIRED
>   FIELD QUANTITY REQUIRED
>   FIELD TAX
>   FIELD TOTAL REQUIRED
>   FIELD DOUBLECHECK &
>     PICTURE "^^^^.^^" &
>     DISPLAY
> CLUSTER
>
> BUILD
```

DESCRIPTION

Provides a description of a screen or a field.

Syntax

DESCRIPTION [OF] SCREEN | {[FIELD] field} string [[,]string]...

[OF] SCREEN| {[FIELD] field}

Specifies a description for a specified entity.

[OF] SCREEN

Specifies that you're defining a description for the current QUICK screen design. The description appears when the QUICK screen user enters the Extended help command (??) in the Action field of that screen.

[OF] [FIELD] field

Specifies a description for the named field. The field description appears when the QUICK screen user enters the Extended help command (??) while the cursor is positioned on that field.

Limit: The field must have been previously declared with a FIELD statement.

string [[,] string]...

Indicates a line or lines of descriptive text for the screen or field.

Limit: A combined maximum of 255 lines can be specified for descriptions and help messages in each screen design.

Discussion

The DESCRIPTION statement can be used in both the data and layout sections of the screen design.

The description is displayed when the QUICK user enters the Extended Help command (??). (**OpenVMS**: ?? or GOLD/PF2)

Example

The following example illustrates how to add descriptions that serve as online documentation for your QUICK screens. The description appears when the screen user enters ?? while the cursor is positioned in the Action field.

```
> SCREEN MAINMENU MENU
> DESCRIPTION OF SCREEN &
>   "The Future Industries Main Menu grants you ",&
>   "access to order and invoice processing", &
>   " screens, as well as to system maintenance",&
>   " screens if you have the necessary" &,
>   " application security class ",  &
>   " ", &
>   "To access Order Processing, press 1",  &
>   "To access Invoice Processing, press 2", &
>   "To access the Maintenance Menu, press 3"
.
.
.
```

DRAW

Draws lines and boxes on a screen.

Syntax

DRAW [option] [FROM] line1 [,column1] [TO] [line2] [,column2]

Options

The DRAW options are char, DOUBLE, THICK, and THIN.

char|DOUBLE|THICK|THIN

The char option draws lines using the specified character to simulate line drawing.

A char is a single displayable character (a letter, number, or special character) enclosed in double or single quotation marks.

Default: THIN

[FROM] line1 [,column1] [TO] [line2] [,column2]

Specifies box-drawing coordinates. The first set of coordinates is the upper left-hand corner of the box. The second set of coordinates is the lower right-hand corner of the box.

If the FROM and TO line coordinates are identical, a horizontal line is drawn. If the two column coordinates are identical, a vertical line is drawn. An error message is issued if both FROM and TO coordinates are equal, or if the TO coordinates are less than the FROM coordinates.

If line drawing is not supported by the terminal used at run time, and if no line-drawing character has been specified in QKGO, an asterisk (*) is used to simulate line drawing.

Default: Current line for the line; column 1 for the FROM coordinate; the last column of the screen for the TO coordinate. QDESIGN issues errors if you overwrite existing fields, including the Mode and Action fields and Action bars.

Discussion

The DRAW statement is part of the layout section of the screen design. It allows the QUICK screen designer to specify line and box drawing.

The drawing coordinates are relative to the starting row and column of the current screen, and not the first row and column of the terminal screen. The DRAW statement doesn't affect the current line position.

The characters that are available for drawing lines and line connections are dependant on your terminal or terminal emulator. On Windows, the characters are dependant on the font or code page used by the Console window.

Fonts and Line Drawing (Windows)

You can apply numerous Windows fonts to a Console window. However, only fonts that have both the ASCII character set and the ASCII extended character set should be used with QUICK.

The line drawing characters used by QUICK on Windows are part of the ASCII extended character set. If the font used does not contain the ASCII extended character set, then proper line drawing will not be possible or improper line drawing characters will be displayed.

For example, the font Lucida Console supports both the ASCII and the extended ASCII characters set.

Example

The following example draws boxes around three main menu selections:

```
> SCREEN MAINMENU MENU &
```

```
>    NOMODE &
>    ACTION LABEL &
>       "Enter a number to select an option: " &
>    AT 1,33
>
> SKIP TO LINE 4
>
> TITLE "Main Menu" CENTERED
>
> SKIP TO LINE 8
>
> DRAW 8,10 TO 15,30
> DRAW 8,50 TO 15,70
> DRAW 16,25 TO 22,55
>
> SUBSCREEN ORDMAIN &
> NOLABEL &
> ID 1 AT 10,19
>
> SUBSCREEN INVMAIN &
> NOLABEL &
> ID 2 AT 10,59
>
> SUBSCREEN MAINT &
> NOLABEL &
> ID 3 AT 18,39
> TITLE "Order" AT 11,18
> TITLE "Invoice" AT 11,57
>
> TITLE "Processing" AT 12,16
> TITLE "Processing" AT 12,56
>
> TITLE "System" AT 19,38
> TITLE "Maintenance" AT 20,35
>
> BUILD
```

EXIT

Ends a QDESIGN session.

Syntax

EXIT

Discussion

The EXIT statement ends your QDESIGN session and returns control to the operating system or to the invoking program. The EXIT statement can be abbreviated E, EX, or EXI.

You can also use the QUIT statement to leave QDESIGN and return control to the operating system or invoking program.

FIELD

Creates fields on the screen that correspond to items for data entry and display.

Syntax

FIELD item [option]...

Limit: A maximum of 255 fields can be declared in a screen design.

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, or a predefined item. The general form of a record item is:

item [OF record-structure]

The OF record-structure qualifier isn't valid for defined or temporary items.

Default: The ASSUMED record-structure. See the SET statement for a discussion of the ASSUMED record-structure option

Options

FIELD Options		
ALLOW|NOALLOW	AUTONEXT|NOAUTONEXT	BWZ|NOBWZ
CENTURY	CHARACTER|DATE|NUMERIC	DATA AT
DEFAULT	DISPLAY	DOWNSHIFT|UPSHIFT|NOSHIFT
DUPLICATE	[ENTRY] IF	ERRORCALL|NOERRORCALL
FILL	FIXED	FLOAT
FOR	FORCE|NOFORCE CENTURY	FORMAT
HELP	HIDDEN	HILITE DISPLAY
ID|NOID	INPUT	LABEL|NOLABEL
LEADING	LOOKUP	MARK|NOMARK
NOCHANGE	NOCORRECT	NOECHO
NOENTRY	NOFORMAT	NORECALL
NOSELECT	NULL VALUE	NULLSEPARATOR|NONULLSEPARATOR
OMIT	OUTPUT	PATTERN
PICTURE	POPUP	PREDISPLAY
REFRESH	REQUIRED	REVERSE
RJ	SELECTBOX	SEPARATOR
SIGNIFICANCE	SILENT	SIZE
TRAILING	VALUES	

ALLOW|NOALLOW CENTURY

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

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

Default: To find out the active value of the option, you must look at the ELEMENT, the USAGE, and the SYSTEM OPTIONS statements. If the option is unspecified on the FIELD statement, the active value is taken from the ELEMENT statement. If the option is unspecified on the ELEMENT statement or a related USAGE, the active value is taken from the SYSTEM OPTIONS statement.

Limit: If used on a non-date field, the field is treated as a date.

AUTONEXT [AUDIBLE]|NOAUTONEXT

Specifies how QUICK reacts when the input size of a field is reached.

AUTONEXT [AUDIBLE]

Instructs QUICK to move automatically to the next field when the input size of the field is reached.

If the entry doesn't fill the field size (for example, if a date is entered without a separator), the QUICK screen user must press [Return] to move to the next field.

AUDIBLE

Sounds a beep before moving to the next field when the field is filled.

NOAUTONEXT

Instructs QUICK not to move automatically to the next field when the input size of the field is reached. When a field is full, the cursor remains on the last position of the field until the user presses [Return].

Default: NOAUTONEXT

BWZ|NOBWZ

BWZ (blank when zero) indicates that the item is displayed as blanks if its value is 0.

NOBWZ indicates that a value of 0 for a numeric item is displayed as zero.

CENTURY INCLUDED|EXCLUDED

Specifies whether or not the century is to be stored in the item. This option forces a non-date field to be processed as a date.

This option affects the storage of the item, not the display characteristics. Use FORMAT to specify how a date field is to be displayed. The item must be defined with sufficient storage space for the century (a minimum size of 8) if the INCLUDED option is used.

CHARACTER|DATE|NUMERIC

Processes the field as the type specified. The field type is normally determined by the item type, but the two types may differ under the following circumstances:

- If you have specified CHARACTER, NUMERIC, or DATE on the FIELD statement, then the field type assumes the specified field type, regardless of the item type.
- If a field is given a FORMAT or SEPARATOR option, the field type is assumed to be date, regardless of the item type.
- If a field has a CENTURY option, it is assumed to be date, regardless of the item type.
- If a field is given any of the following numeric attributes then the field type is assumed to be numeric, regardless of the item type.

ALLOW CENTURY	BWZ
FILL	FLOAT
FORCE CENTURY	INPUT SCALE
LEADING SIGN	NOBWZ

NULL SEPARATOR	OUTPUT SCALE
SIGNIFICANCE	TRAILING SIGN

- If you specify character options on numeric fields, QUICK converts the item to type CHARACTER.

Having field types that differ from item types allows standard editing of a type that is different from that assumed for the item (such as checking for a date in a numeric item or checking for a number in a character item). Also, data can be displayed in a form different from that assumed for the item (such as displaying a name when a numeric code is actually stored in the item). Specifying the NUMERIC option of a FIELD statement for a character item forces a numeric check of the contents, and right-justifies and zero-fills the item.

If an item is numeric but the field is specified as a CHARACTER type or vice versa, any PICTURE option specified for the item in the data dictionary is ignored. Since a picture typically follows the item type, results could be misleading. To prevent this, specify a PICTURE option on the FIELD statement in these cases. Field types that are different than item types are frequently used in conjunction with designer-specified INPUT and OUTPUT procedures for a field.

DATA AT [row],column1

Positions the first character of the data at the specified row and column relative to the starting line of the screen.

Default: If the row isn't specified, the current row is assumed.

DEFAULT expression

Uses the value that results from this expression as the default value for the field. If, during entry, the QUICK screen user presses only [Return] in response to a field prompt, QUICK evaluates the expression and places the result in the record buffer.

If the DEFAULT and REQUIRED options are used in the same statement, the DEFAULT option is ignored.

Limit: This option applies only to a temporary item or to an item in a new data record that has not previously had a value entered.

DISPLAY [ON ENTRY|FIND]

Causes QUICK to display data in a field and not to accept entries for this field.

The DISPLAY option generates DISPLAY verbs, rather than ACCEPT verbs, in the default procedures for fields corresponding to record and temporary items. Fields for defined items are always display fields.

You can override the DISPLAY option by replacing the generated DISPLAY verbs with ACCEPT verbs, or by writing numbered DESIGNER procedures.

ON ENTRY|FIND

Limits the display of data to either Entry mode or Find mode.

Default: Displays the data in both Entrymode and Findmode, and does not accept entries.

DOWNSHIFT|UPSHIFT|NOSHIFT

Shifts the entered value of a character field into the specified case. DOWNSHIFT shifts the entered value to lowercase; UPSHIFT shifts the entered value to uppercase; NOSHIFT overrides upshift or downshift attributes in the data dictionary. DOWNSHIFT and UPSHIFT are applied prior to editing.

Shifting all values into the same case prevents ambiguous entries. Otherwise, uppercase and lowercase letters are treated as distinct.

Limit: Non-alphabetic characters within character items are not affected.

Default: The shift option specified for the corresponding element in the data dictionary.

DUPLICATE

Duplicates the value for this field from the previous occurrence of the field on the screen if the screen user doesn't enter a value. If this is the first data record on the screen, this option duplicates the value from the last data record entered on the previous screenload of data (if one exists) for the current session.

Limit: This option only applies to temporary items or new record items with no previously entered values. INITIAL values (from an ITEM statement) and DEFAULT values are not duplicated.

[ENTRY] IF condition

Prompts for data in the specified field in the standard entry sequence if this condition is satisfied, provided the ENTRY procedure was generated by QDESIGN.

Use the optional keyword ENTRY to avoid the syntax ambiguity that arises when the IF option is specified immediately after a HILITE option with a trailing ELSE control structure. QDESIGN generates an identical IF control structure in the default ENTRY procedure.

Limits: The IF option is evaluated only once during the standard entry sequence. The IF option is ignored for fields in PANEL screens.

ERRORRECALL|NOERRORRECALL

ERRORRECALL instructs QUICK to redisplay; NOERRORRECALL instructs QUICK not to redisplay for correction data that fails the edit check on the field. These options override the QKGO Error Recall specifications.

Default: NOERRORRECALL

FILL char

Specifies the character used to "fill" unused space to the left of the most significant digit, float character, or leading sign in the picture. The fill character also replaces unnecessary leading nonsubstitution characters, including commas and leading spaces. However, the SIGNIFICANCE option can force the display of leading zeros. For more on formatting for numeric items, see (p. 108), (p. 112), (p. 118), and (p. 121).

Fill	Float	Picture	Value	Display
*	$	" ^,^^^,^^^.^^"	123456	***$1234.56

Limit: Causes non-numeric items to be treated as numeric.

Default: A blank

FIXED

Permanently displays the current value for this field when the screen first appears. In effect, the value becomes part of the screen background, and isn't rewritten every time data is displayed.

If the value of the corresponding item was changed since the value was first displayed, either in the original screen or a subscreen, the new value is displayed upon return from any subscreen call.

FLOAT char

Specifies the float character. The float character is inserted immediately to the left of the most significant digit. For example, currency values might be displayed with a dollar sign ($) as a float character. To ensure that there is always room for the float character, either a space or an extra substitution character (by default, ^) must be added to the left side of the picture. Using a nonsubstitution character (for example, a space) leaves all the substitution characters free to accept data input and decreases any chance of field overflow.

Examples:

Value	Picture	Float	Display
1234	"^^^^^"	$	$1234
56789	" ^^^.^^"	$	$567.89

Limit: Causes non-numeric items to be treated as numeric.

Default: No float character is used.

FOR [row], column

Specifies a scrolling field.

row

The number of rows in the field. If the number is greater than 1, the field is created as a multiple-row scrolling field.

Default: 1

column

Scrolling field width. There is no column default.

When the size specified for a field is smaller than the maximum number of allowable characters or digits in the associated record item, the generated field is scrollable. Scrollable fields are indicated by the "<" symbol (which appears to the left of the field) and the ">" symbol (which appears to the right of the field). The "<" symbol indicates that there are undisplayed characters to the left of the scrollable field. Similarly, the ">" symbol indicates that there are undisplayed characters to the right of the scrollable field. To scroll data contained in a scrollable field, QUICK screen users can use left and right arrow keys.

MPE/iX: If the screen is in Block mode, then the line times column must be equal to the field item size.

FORCE|NOFORCE CENTURY

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

If FORCE CENTURY is applied to a date with a two-digit year format, the ALLOW CENTURY option is implied. That is, the field becomes a horizontal scrolling field, so the user can enter the required century.

Default: To find out the active value of the option, you must look at the ELEMENT, the USAGE, and the SYSTEM OPTIONS statements. If the option is unspecified on the FIELD statement, the active value is taken from the ELEMENT statement. If the option is unspecified on the ELEMENT statement or a related USAGE, the active value is taken from the SYSTEM OPTIONS statement.

Limit: If used on a non-date field, the field is treated as a date.

FORMAT date-format

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

Date-format	Example	Date-format	Example
YYMMDD	01/05/23	YYMMMDD	01/MAY/23
YYYYMMDD	2001/05/23	YYYYMMMDD	2001/MAY/23
YYMM	01/05	YYMMM	01/MAY

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

YYYY - four digit year (e.g., 2001)

MM - two digit month (e.g., 05)

MMM - three character month name (e.g., MAY)

DD - two digit day for a month (e.g., 23)

DDD - three digit day for a year (e.g., 365)

Regardless of the output order of the date, the internal working format is YYMMDD (for dates without centuries), YYYYMMDD (for dates with centuries), and YYYDDD (for Julian dates)

The FORMAT option governs data entry by determining the way you can enter date values. Dates can always be entered in the format specified in the FORMAT option, with or without the established separator character and with either the MM or MMM month format.

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

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

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

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

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

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

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

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

HELP string

Displays a one-line message when QUICK screen users enter a help command (?) in the field.

Limit: A combined maximum of 255 lines can be specified for descriptions and help messages in each screen design. If not specified, then the help message from the dictionary is used.

HIDDEN

Hides the ID-number for the field. Use the HIDDEN option to allow you to reference a given field using its hidden ID-number (even though no ID-number for that field appears on the screen when using MARK mode). The HIDDEN option also allows you to write numbered DESIGNER procedures for fields that have no displayed ID-number.

HILITE DISPLAY highlight-option [IF condition [ELSE highlight-option IF condition]... [ELSE highlight-option]]

Changes the highlighting characteristics of a field based on conditions that are evaluated when the field is displayed. More than one highlight option can be used for data satisfying a particular condition.

The HILITE statement overrides the display settings of any previous HILITE statement. If no final ELSE portion is specified on the HILITE expression, and none of the conditions are satisfied, the field is displayed with the highlighting determined by the last HILITE statement that affected display highlighting.

highlight-option

Specifies the highlighting options used.

The highlight-option can be one or more of the following:

Highlight	Effect
BLINKING (**MPE/iX, OpenVMS, UNIX**)	Highlights the object with blinking.
color [ON color]	Highlights the data in the field in the specified color. Color can be one of: RED, WHITE, BLUE, CYAN, GREEN, MAGENTA, BLACK, or YELLOW. *Note:* Colors are ignored if you attempt to assign colors to fields that are displayed on monochrome monitors.
DEFAULT	Applies default highlighting to the object.
HALFTONE (**MPE/iX, OpenVMS, UNIX**)	Highlights the object with half intensity or alternative intensity, depending on the terminal type.
INVERSE	Highlights the object with inverse video (reverses the normal background and foreground settings).
OFF	Cancels highlighting.
UNDERLINE (**MPE/iX, OpenVMS, UNIX**)	Highlights the object with an underline.

Limit: If either the DEFAULT or the OFF options is specified, none of the other highlighting options can be used.

ID SAME
ID n [AT [line],column]
ID NEXT [AT [line],column]
NOID

Declares the ID-number and its position. Controls the assignment of an ID-number to a field.

ID SAME

Omits the ID-number for the field. To reference the field, use the ID-number of the last field for which an ID was specified.

ID AT [line,]column

Specifies a location for the ID-number for the field.

ID n

Specifies an ID-number explicitly.

Limit: 1 to 99

ID NEXT

Assigns the next ID-number in sequence.

AT [line,]column

Positions the first digit of the ID-number at the specified line and column relative to the starting line of the screen. If the line is missing, the current line is assumed.

NOID

Assigns no ID-number to this field; the field can't be referenced from the Action field.

Default: For NOPANEL screens, ID NEXT for all fields. For PANEL screens, ID NEXT for the first field in each cluster occurrence; ID SAME for all other fields within each cluster occurrence that occurs with a repeating PRIMARY or DETAIL file; NOID for fields in clusters that don't occur with a repeating PRIMARY or DETAIL file.

INPUT [SCALE] n

Sets the value used as a scaling factor when values are entered for a numeric item.

The entered value is multiplied by ten raised to the power of the input scale (that is, 10^n) before it is stored. SCALE is used only for documentation.

Limit: Scale values range from -20 to 20. Causes non-numeric items to be treated as numeric.

LABEL string [AT [line],column]
NOLABEL

Declares the field label and its position.

LABEL string [AT [line],column]

Indicates the field label and, optionally, the position of the label on the screen.

The AT option positions the first character of the label at the specified line and column relative to the starting line of the screen. If the line isn't specified, the current line is assumed.

Default: If no option is specified, the LABEL or HEADING string from the data dictionary is used. If neither the LABEL nor the HEADING option is specified in the data dictionary, the item name is used.

NOLABEL

Indicates that no field label is to appear.

LEADING [SIGN] char

Specifies a single character that's placed to the left of the most significant digit (or float character, if used) to indicate that the numeric value displayed is a negative number.

To enter a negative value on a screen for a temporary item, a LEADING SIGN or TRAILING SIGN option must be specified on the corresponding FIELD statement.

Sufficient substitution characters (by default, ^) or nonsubstitution characters (any other characters) must be provided in the picture to accommodate the leading sign. If the picture is too small, overflow occurs, displayed with crosshatches (#). SIGN is used only for documentation.

For example, the number -1578 could be formatted as follows:

Picture	Leading sign	Trailing sign	Display
"^,^^^"	"-"	none	#####
" ^^^,^^^"	"-"	none	-1,578
" ^^^,^^^ "	"("	")"	(1,578)

Limit: Causes non-numeric items to be treated as numeric.

LOOKUP [ON|NOTON] record-structure [lookup-option]...
[,[ON|NOTON] record-structure[lookup-option]...]...

or

LOOKUP [ON|NOTON] cursor-reference[sql-substitution...]
[lookup-option]...
[,[ON|NOTON] cursor-reference[sql-substitution...]
[lookup-option]...]...

ON|NOTON

Specifies that the field value must either exist (ON) or must not exist (NOTON) in the named record-structure or cursor-reference.

LOOKUP ON can also be used to retrieve data for display. If the NOTON option is used, no data transfer takes place.

Default: ON

Limit: A maximum of 10 lookups are allowed on one FIELD statement.

record-structure

Specifies the record-structure to which the lookup is being applied.

cursor-reference[sql-substitution...]

Specifies the cursor-reference to which the lookup is being applied. The cursor-reference is a cursor or table named in a CURSOR statement.

An sql-substitution can be specified for any substitution variable defined on the DECLARE CURSOR statement. Two default 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 a substitution is:

substitution-variable (text)

For more information, see Chapter 1, "PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

The following options are valid if the LOOKUP is on a cursor-reference:

- AUTOCOMMIT
- NOWARN
- OPTIONAL
- MESSAGE

The following options are not valid if the LOOKUP is on a cursor-reference:

- VIAINDEX

- VIA
- USING
- BACKWARDS
- GENERIC|NOGENERIC
- SEQUENTIAL

LOOKUP Options		
AUTOCOMMIT	BACKWARDS	GENERIC|NOGENERIC
MESSAGE	NOWARN	OPTIONAL
SEQUENTIAL	USING	VIAINDEX
VIA		

AUTOCOMMIT

Automatically commits the transaction associated with the lookup after the lookup is completed. For example,

```
> FIELD EMPLOYEE LOOKUP &
>   ON table1, &
>   ON table2 AUTOCOMMIT
```

If each table uses a distinct transaction, only the second transaction is committed. If both lookups are done in the same transaction, the transaction is committed after the second lookup.

BACKWARDS

Reverses the sequence in which the record-structure is normally read.

Limit: Valid only for C-ISAM, DISAM, RMS ISAM, and IMAGE datasets with keyed access.

Limit: The BACKWARDS and SEQUENTIAL options cannot be used together for RMS ISAM files.

GENERIC|NOGENERIC

GENERIC allows partial-index retrieval; NOGENERIC prevents partial-index retrieval.

Limit: Not valid for IMAGE indexes, unless they are B-Tree and OMNIDEX indexes. Not valid with a cursor-reference.

Default: GENERIC

MESSAGE n|=string-expression|string

Displays a message if the lookup fails. The general terms string and string-expression specify the message to be displayed.

If a number (n) is specified, QUICK searches for a message in the designated message file: qkmsgdes (**MPE/iX, OpenVMS**) or qkmsgdes.txt (**UNIX, Windows**).

For more information, see Chapter 4, "Messages in PowerHouse", in the *PowerHouse Rules* book.

Limit: The MESSAGE and NOWARN options are mutually exclusive.

NOWARN

Suppresses the display of messages when an optional lookup fails.

If NOWARN is specified, QUICK assumes the lookup is optional even if it isn't specified, since combining NOWARN with a required lookup would defeat the purpose of the lookup.

Limit: The MESSAGE and NOWARN options are mutually exclusive.

OPTIONAL

Continues processing even if the access fails.

If no data record is found when the LOOKUP ON option is retrieving data for display, a data record is created that contains initial or default values for each item. These values are taken from the data dictionary and any ITEM statements. If no initial values are specified in the data dictionary or in an ITEM statement, any CHARACTER item is initialized to spaces, and any NUMERIC or DATE item is initialized to zeros.

Limit: Not valid for PRIMARY files.

SEQUENTIAL

Accesses the data records in the file sequentially.

Limit: SEQUENTIAL and USING can't be used in the same LOOKUP option. Valid only for PRIMARY files. Not valid with a cursor-reference.

Limit: The BACKWARDS and SEQUENTIAL options cannot be used together for RMS ISAM files.

USING expression [,expression]...

Accesses an associated file using the results of a specified expression as

- the corresponding linkitems value for an indexed file
- the data record number for record-structures in a direct or relative file
- the column value in a relational table

For direct files, there can be only one value which QUICK interprets as a record number.

For indexed files, there can be more than one value (i.e., segmented indexes), but QUICK interprets the values as a single index value in that file.

If the record-structure belongs to a direct file, there can only be one expression specified, which must be numeric. Otherwise, a series of expressions can be specified which correspond one-to-one with the segments established by either the VIA or VIAINDEX options. If neither the VIA nor the VIAINDEX option is specified, and the record-structure has only one associated index, this index is used as if the VIAINDEX option had been specified.

If a file is a relational table, there can be several values in the USING option, which QDESIGN interprets as the values of the columns in the table. The VIA or VIAINDEX options must be used to indicate which columns the values belong to if more than one index is in use or if no index is used.

If the VIA option is specified, the number of expressions specified must correspond one-to-one with the number of linkitems specified on the VIA option.

If the VIAINDEX option is specified and the VIA option isn't, the number of expressions may be less than or equal to the number of segments contained within the specified index. There must always be at least one expression.

Limit: 255 expressions. Not valid for a cursor-reference.

Limit (**MPE/iX**)**:** IMAGE does not support retrieval via an initial subset of the segments of an index, unless the index is a B-Tree or OMNIDEX index. An expression must be specified for every segment of the index.

The following options specify retrieval for the lookup:

VIAINDEX indexname

The name of a PowerHouse index as defined in the data dictionary or a relational database. This option enforces data record retrieval to be performed in index order. Lookups can only be done against indexed files or tables in relational databases.

Limit: Not valid for a cursor reference.

VIA linkitem [,linkitem]...

Accesses the record-structure via the specified linkitems.

For record-structures in indexed files, the declared linkitems must define a series of segments contained within an index structure associated with the record-structure. In this case, the first declared segment is the first segment within the index, the second declared segment is the second segment within the index, and so on.

Limit: 255 segments. Not valid for a cursor-reference.

Limit (**MPE/iX**): IMAGE does not support retrieval via an initial subset of the segments of an index, unless the index is a B-Tree or OMNIDEX index. The series of linkitems must include all of the segments in the index.

MARK|NOMARK

MARK enables fieldmarking for a field with an ID-number.

NOMARK disables the default fieldmarking for a field with an ID-number when field marking is enabled.

Default: MARK

MISSING|NULL VALUE [NOT] ALLOWED

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.

For more information about controlling null value entry in QDESIGN, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Default: NULL VALUE ALLOWED

NOCHANGE

States that once a value has been put on file, it can't be altered by an entry in this field.

This option is added automatically to fields that represent segments of indexes if the GENERATE statement is used to construct the FIELD statements. The option applies if the predefined condition NEWRECORD is false. It isn't affected by any predefined mode conditions. The ACCEPT, PROMPT, and REQUEST verbs can't be used to override the NOCHANGE option, although a LET verb can be used to make changes directly to the corresponding item.

To change the value of a data item in a field that has the NOCHANGE option specified, the QUICK screen user must first delete the existing data record and then enter new values for it.

Temporary item fields with NOCHANGE cannot be changed regardless of whether they occur with a file or not.

The CACHE option does not affect the behavior of NOCHANGE.

Limit: Does not apply to defined item fields.

NOCORRECT

States that once a value is entered, it can't be altered until it is put on file.

This option applies if the predefined condition NEWRECORD is true. The NOCORRECT option isn't affected by any predefined mode conditions. The ACCEPT, PROMPT, and REQUEST verbs can't be used to override the NOCORRECT option, although a LET verb can be used to make changes directly to the corresponding item.

Temporary item fields with NOCORRECT cannot be corrected regardless of whether they occur with a file or not.

The CACHE option does not affect the behavior of NOCORRECT.

Limit: Does not apply to defined item fields.

NOECHO

Suppresses the display of any value entered in the field when the terminal is operating in full duplex.

The NOECHO option fields are always redisplayed as blanks.

NOENTRY

Omits this field from the standard entry sequence.

QDESIGN doesn't generate an ACCEPT verb in the default ENTRY procedure. For more information, see (p. 314).

NOFORMAT n

Instructs QUICK to display the contents of FIELDTEXT as is after the OUTPUT procedure without applying any additional formatting.

This option is specified when the FORMATNUMBER function is used to format a value in the OUTPUT procedure. This ensures that other specified formatting and scaling, either from the FIELD statement or from the dictionary, is ignored.

n

Specifies the display width of the field. FIELDTEXT is truncated on the right to fit the display width. If non-blank characters are truncated, overflow characters (#) are displayed instead of the contents of FIELDTEXT.

NORECALL (OpenVMS)

Turns off single-line recall capability for data fields.

NOSELECT

Disables the field from being used in Select mode. Prevents a SELECT verb from being generated in the default SELECT procedure.

If a SELECT verb is included in a user-defined SELECT procedure, the NOSELECT option prevents the QUICK screen user from using the field in the selection process.

If no SELECT procedure is generated or written, the NOSELECT option prevents the QUICK screen user from using the field's ID-number as a selection indicator when selecting data records.

NULLSEPARATOR|NONULLSEPARATOR

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

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

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

Default: To find out the active value of the option, you must look at the ELEMENT, the USAGE, and the SYSTEM OPTIONS statements. If the option is unspecified on the FIELD statement, the active value is taken from the ELEMENT statement. If the option is unspecified on the ELEMENT statement or a related USAGE, the active value is taken from the SYSTEM OPTIONS statement.

Limit: If used on a non-date field, the field is treated as a date.

OMIT [ON ENTRY|FIND]

States that, under normal circumstances, the data for this field cannot be entered or displayed on the screen (although its appearance can be forced by a procedure).

ON ENTRY

Prevents entry and display during Entry and Correct modes.

ON FIND

Prevents display and changes during Find, Change, and Select modes. You can override the OMIT option by using verbs.

OUTPUT [SCALE] n

Establishes the output scaling factor. The value displayed is multiplied by 10 raised to the power of the output value (that is, 10^n) before it is displayed. The result is rounded after scaling.

QUICK displays only the integer portion of a number by doing a right-to-left replacement of each substitution character (by default, ^) in the picture with digits from the number.

No matter which picture is specified, only the integer portion (123) of the number is displayed.

Number	Picture	Output Scale	Display
123.456	"^^^"	0	123
123.456	"^.^^"	0	1.23
123.456	"^^.^"	0	12.3

To display the decimal portion of the number, you must use the OUTPUT SCALE option.

Number	Picture	Output Scale	Display
123.456	"^^^^^^"	0	123
123.456	"^^^^^^"	1	1235
123.456	"^^^^^^"	2	12346
123.456	"^^^^^^"	3	123456
123.456	"^^^^^^"	-1	12
123.456	"^^^^^^"	-2	1
123.456	"^^^^^^"	-3	0

For proper decimal alignment on output, you must make proper use of the SCALE and PICTURE options.

Number	Picture	Output Scale	Display
123.456	"^^^.^"	1	123.5
123.456	"^^^.^^"	2	123.46
123.456	"^^^.^^^"	3	123.456

Limit: -20 to 20. Causes non-numeric items to be treated as numeric.

PATTERN string|=string-expression

Specifies a string of characters and metacharacters that provides a general description of values. The entry must match the specified pattern string to be valid. For more information about patterns, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

PICTURE string

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

If the PICTURE and SIGNIFICANCE options are used together, the PICTURE option must be specified first. The number of substitution characters in the default picture for numeric defined items is the number of digits in the item plus one. For example,

```
> DEFINE X NUMERIC SIZE 4
```

gives a picture of
"^^^^^"

Limit: 60 characters

How PowerHouse Formats CHARACTER Items

Character-type items are formatted in the following manner:

1. The item is processed from left to right, substituting one character from the item for each substitution character (by default, ^) in the picture. Nonsubstitution characters remain unchanged.
2. If there are fewer substitution characters in the picture than characters in the item value, the remaining characters in the item are not displayed.
3. If there are more substitution characters in the picture than characters in the item value, the item is padded with spaces on the right.

For example, the item "FHSMITH" is formatted as follows:

Picture	Display
none	FHSMITH
"^^^^^"	FHSMI
"^.^.^^^^^"	F.H.SMITH

How PowerHouse Formats NUMERIC Items

Numeric items are formatted in the following manner:

1. The item value is scaled by the output scale and rounded to the nearest whole number.
2. The integer portion of the item is processed from right to left substituting one digit from the item for each substitution character (by default, ^) in the picture until all significant (non-zero) digits have been transferred. Nonsubstitution characters remain unchanged.
3. Until the significance is reached, leading zeros are substituted for each substitution character, and nonsubstitution characters remain unchanged.
4. The float character is added.
5. Leading or trailing sign characters or both are added for negative numbers.
6. The remaining portion of the picture is filled with the fill character.
7. If there isn't enough room in the picture to hold all the significant digits of the item value, or the LEADING SIGN, TRAILING SIGN, or FLOAT characters, it is filled with the overflow character (the crosshatch #).

For example, the number 1578 is formatted as follows:

Picture	Display
"^^^"	###
"^^^^"	1578
"^^.^^"	15.78
"^^,^^^,^^^"	1,578

If the numeric field formatting options do not provide the formatting you require, the FORMATNUMBER function provides extended functionality. For more information, see Chapter 6, "Functions in PowerHouse", in the *PowerHouse Rules* book.

POPUP [FROM [application line1],column1] [TO [application line2],column2] [ON mode[,mode]...]

Creates a pop-up data entry window at optionally-specified coordinates, which can open automatically under specified conditions.

FROM [application line1],column1 TO [application line2],column2

The top left and bottom right corners of the data-entry window specified in the application line coordinates.

Default: If application line1 is missing, the current row minus 5 is used and application line2 defaults to the current row plus 5. Column1 defaults to 5; column2 defaults to 75.

[ON mode[,mode]...]

Specifies in what mode the data entry window is to be automatically popped up. The mode can be one:

DATA	Whenever the user is prompted in the field (a combination of EDIT, INPUT, and REQUEST).
EDIT	After the user enters an incorrect value in the field.
INPUT	When the user is prompted in the field to enter data in Entry, Change, and Correct mode. The window does not pop-up when an edit check fails unless you specify EDIT or DATA as well.
REQUEST	When the user is prompted in an index segment field in Find mode or prompted in this field in Select mode when defining retrieval criteria.

Default: If ON mode is not specified, the pop-up window only appears when the user enters the Popup Toggle (+) command.

PREDISPLAY

Displays any initial value for this field as soon as the screen appears if the mode is changed or if a cluster is initialized; it doesn't fix the value in the screen background. The PREDISPLAY option is assumed for all fields on a slave screen.

REFRESH

Redisplays the value for this field when returning from a lower-level screen.

REQUIRED

Declares that a null response from the QUICK screen user isn't acceptable. A valid entry must be supplied. This option is added automatically to fields that represent index segments when the GENERATE statement is used to construct the FIELD statements.

Limit: If the DEFAULT and REQUIRED options are used in the same statement, the DEFAULT option is ignored.

REVERSE

Specifies that QUICK prompts the screen user beginning at the right side of the field. As characters are entered, QUICK inserts them into the field from right to left.

RJ

Right-justifies a character string when it is displayed in a QUICK screen field.

Limit: Valid only for character items.

SELECTBOX [FROM [application line1],column1] [TO [application line2],column2] [ON mode[,mode]...]

Creates a selection box which can appear under specified conditions.

When selection processing is enabled, the values are drawn from the dictionary or VALUES option. If the VALUES option contains a range, only the first and last values are displayed.

FROM [application line1],column1 [TO [application line2],column2]

The top left and bottom right corners of the Selection box specified in the application line coordinates. If the specified box is too narrow for the display of the selection list, the display values are truncated and the full form of the selected value is transferred to FIELDTEXT.

Default: Appears directly under the field in question. The width is the field width; the depth is the number of values (up to 8) unless the box does not fit on the screen, in which case it then becomes a scrolling Selection box.

[ON mode[,mode]...]

Specifies in what mode the selection box is to be automatically popped up. The mode can be one:

DATA	Whenever the user is prompted in the field (a combination of EDIT, INPUT, and REQUEST).
EDIT	After the user enters an incorrect value in the field.
INPUT	When the user is prompted in the field to enter data in Entry, Change, and Correct mode. The window does not pop-up when an edit check fails unless you specify EDIT or DATA as well.
REQUEST	When the user is prompted in an index segment field in Find mode or prompted in this field in Select mode when defining retrieval criteria.

Default: If ON mode is not specified, the selection box only appears when the user enters the Select Box (#) command.

SEPARATOR char

Specifies the character used to separate the year, month, and day portions of a date item.

Limit: Valid only for date items.

Default: The system default from the data dictionary is used. If no system default is specified, a slash (/) is used.

SIGNIFICANCE n

Establishes the minimum number (n) of characters displayed. It is generally used to force the display of leading nonsubstitution characters and leading zeros.

For example, the number 0.1578 would be formatted as follows:

Picture	Significance	Display
"^^.^^^^"	6	0.1578
"^^.^^^^"	4	1578

Limit: If the PICTURE and SIGNIFICANCE options are used together, the PICTURE option must be specified first. Causes non-numeric items to be treated as numeric.

SILENT

Specifies that a field doesn't appear on the screen.

A silent field is included to perform standard field editing of a data item that has been entered as a related set of subitems in the preceding fields. When this option is used, all related fields should have the same ID-number so that the edit is performed when any of the data item's components are changed.

The silent field should be the last one of a set of related fields. A silent field has the ID SAME option specified regardless of any other ID option. You can't override this option procedurally.

Limit: The SILENT option is mutually exclusive of all but the IF, LOOKUP, PATTERN, and VALUES options.

SIZE n

Specifies the number (n) of characters in the entry field.

Normally this option isn't required, since the size of the field is determined from the picture or from basic attributes of the element, usage, or relational column. However, the SIZE option can be used when the field size must be larger or smaller than the corresponding element size. A field that has a SIZE option that specifies a length less than the field length is scrollable.

Record items based on elements defined with sizes greater than 60 characters are automatically assigned a field size of 60.

TRAILING [SIGN] string

Specifies one or two characters that are placed in the right-most portion of the picture to indicate that the numeric value displayed is a negative number. Sufficient nonsubstitution characters must be provided in the right-most portion of the picture to accommodate the trailing sign. If the picture is too small, it is filled with the overflow character (by default, the crosshatch #). The format options LEADING SIGN and TRAILING SIGN can be used together to place parentheses around negative numbers. SIGN is used only for documentation.

To enter a negative value on a screen for a temporary item, a LEADING SIGN or TRAILING SIGN option must be specified on the corresponding FIELD statement.

For example, the number -1578 could be formatted as follows:

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

Limit: Causes non-numeric items to be treated as numeric.

VALUES value-caption-set

Specifies acceptable entry values for the field, and overrides values specified in the dictionary, if any. The values must be consistent with the element size and type. Numeric values are scaled by the input scale. Dates values must be specified in the YYYYMMDD or YYMMDD format.

value-caption-set

Specifies one or more values and/or one or more ranges of values. A caption string may be specified with each value for display in selection boxes where applicable. The general form is:

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

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

Captions specified in the FIELD statement override captions specified in the dictionary, if any.

Discussion

The FIELD statement is part of the layout section of the screen design. It creates a field on the screen in which the QUICK screen user can enter, display, or change data values contained in a corresponding data record, temporary item, or defined item.

Fields for Defined and Temporary Items

If the field corresponds to a defined item, the ID SAME and DISPLAY options are assumed.

Fields corresponding to temporary items can't be used to enter selection criteria in Select mode.

Date and Time Support in QUICK

A FIELD statement can be automatically generated for a DATETIME datatype with a default time format and field size. The date portion format can be specified in the dictionary, as in:

```
> ELEMENT TWO DATE SIZE 16 FORMAT MMDDYYYY
```

In the previous example, the full format for item TWO is MMDDYYYY HH:MM:SS.NN.

The following table shows the possible results when entering data into field TWO:

Entered	Displayed
09/29/1987 23:59:59.99	09/29/1987 23:59:59.99
09291987 23595999	09/29/1987 23:59:59.99
09/291987 23:5959.99	09/29/1987 23:59:59.99
09/29/1987 012233.44	09/29/1987 01:22:33.44
09/29/1987 00:00:00.00	09/29/1987 00:00:00.00
09/29/1987 23:59:59	09/29/1987 23:59:59.00
09/29/1987 0	09/29/1987 00:00:00.00
0 0	[empty field]
0	[empty field]
[Return]	[empty field]

Rules for Entering the Time Portion

1. Both digits of each component must be entered. If the hour is "1", then the entry must be "01". However, truncating the time, such as leaving off the seconds value, is valid.
2. The time separators do not have to be used but if they are, they must be ":" and "." in their proper positions.
3. A space must separate the date and time portions.
4. The time component limits are: 0-23 for hours, 0-59 for minutes and seconds, and 0-99 for hundredths of seconds.

Example

The STAFF screen in the following example demonstrates the use of the FIELD statement. In this example:

- REQUIRED prevents the QUICK screen user from making a null entry for EMPLOYEENUMBER. LOOKUP NOTON ensures that an entered value isn't already on file.
- NOCHANGE prevents values from being changed once they're stored.

- CITY, PROVSTATE, and POSTALCODE are assigned the same ID-number as the STREET field. When the QUICK screen user enters the ID-number for the STREET field to make corrections, QUICK prompts for each of the four fields.
- The pattern on the POSTALCODE field allows the entry of a Canadian postal code or an American zip code.
- PHONE appears with the label "Telephone number" and a help string. If the QUICK screen user enters "?", this string appears in the message line. If there is no phone number, BWZ means that nothing is displayed.
- LOOKUP ON ensures that values entered for BRANCHCODE already exist in BRANCHES.
- DISPLAY displays the value for BRANCHNAME so that QUICK screen users can see which branch the entered BRANCHCODE refers to.
- OPTIONAL causes QUICK to continue processing even if the lookup fails because the position entered doesn't already exist in POSITIONS.
- VALUES determines allowable entries in the SEX field. Entries other than "M" or "F" are not accepted. UPSHIFT changes entries to uppercase to match the values "M" and "F".
- FORMAT overrides the formatting specified in the data dictionary. Dates appear in the format MMMDDYY (displayed as FEB 14 91). Entries in DATEJOINED don't have to include the specified SEPARATOR (spaces). However, the date value must be entered in the order month-day-year.
- BONUSPAY is displayed in the format established by the picture string. A dollar sign float character is prefixed immediately to the left of the value.

```
> SCREEN STAFF
>
> FILE EMPLOYEES
>   ITEM DATEJOINED &
>      INITIAL REMOVECENTURY(SYSDATE)
> FILE BRANCHES REFERENCE
> FILE DIVISIONS REFERENCE
> FILE POSITIONS REFERENCE
>
> HILITE TITLE UNDERLINE
> TITLE "STAFF SCREEN" AT 1,30
> HILITE TITLE OFF
> SKIP 1
>
> FIELD EMPLOYEENUMBER OF EMPLOYEES  &
>   REQUIRED NOCHANGE &
>   LOOKUP NOTON EMPLOYEES
>
> FIELD LASTNAME OF EMPLOYEES  &
>   REQUIRED   &
>   NOCHANGE &
>   LABEL "Last Name"
>
> FIELD FIRSTNAME OF EMPLOYEES  &
>   LABEL "First Name"
> FIELD STREET OF EMPLOYEES  &
>   LABEL "Address"
>
> SKIP
> ALIGN (,,21) (,,44) (,,50)
>
> FIELD CITY OF EMPLOYEES ID SAME
> FIELD PROVSTATE OF EMPLOYEES ID SAME
> FIELD POSTALCODE OF EMPLOYEES  &
>   PATTERN "(^#^ #^#)|(#####(-####)<)" ID SAME
> ALIGN
> FIELD PHONE OF EMPLOYEES  &
>   LABEL "Telephone number" &
>   HELP "Please include the area code."  &
>   PICTURE "(^^^) ^^^-^^^^" &
>   BWZ
>
```

```
> ALIGN (1,4,21) (,,25)
> FIELD BRANCHCODE OF EMPLOYEES  &
>   LOOKUP ON BRANCH CODE"
>
> FIELD BRANCHNAME OF BRANCHES  &
>   DISPLAY   &
>   ID SAME
>
> FIELD DIVISION OF EMPLOYEES  &
>   LOOKUP ON DIVISIONS   &
>   LABEL "Division"
>
> FIELD DIVISIONNAME OF DIVISIONS  &
>   DISPLAY  &
>   ID SAME
>
> FIELD POSITION OF EMPLOYEES  &
>   LOOKUP ON POSITIONS OPTIONAL  &
>   LABEL "Position"
>
> FIELD POSITIONTEXT OF POSITIONS  &
>   DISPLAY  &
>   ID SAME
>
> ALIGN
>
> FIELD SEX OF EMPLOYEES  &
>   VALUES "M", "F"  &
>   UPSHIFT   &
>   LABEL "Sex"
>
> FIELD DATEJOINED OF EMPLOYEES  &
>   FORMAT MMDDYY SEPARATOR " " &
>   LABEL "Date Joined"  &
>   DEFAULT SYSDATE &
>   ID SAME
>
>
> FIELD BONUSPAY OF POSITIONS  &
>   PICTURE " ^,^^^.^^" &
>   FLOAT "$"
>
> BUILD
```

Selection Box Value Captions

In the following screen, the Language field includes the captions "English", "French", and "German" for the entry values "E", "F", and "G" respectively. If the field's selection box is opened, then it displays the captions in place of the entry values:

```
.
.
.
> FIELD LANGUAGE LABEL "Language" &
>   SELECTBOX &
>     VALUE "E" CAPTION "English" &
>     VALUE "F" CAPTION "French" &
>     VALUE "G" CAPTION "German"
.
.
.
```

FILE

Identifies and describes a record-structure accessed by the screen.

Syntax

FILE record-structure [option]...

Limit: A maximum of 31 files, record-structures, and cursors can be declared in a screen design. There can be a maximum of 1023 items per record-structure.

record-structure

A record-structure declared in the data dictionary, a table declared in a relational database, or a subfile.

The general form of the record-structure is:

record [IN file]|[owner.]table [IN database]|*subfilespec

IN database

The PowerHouse name of the relational database attached to the current dictionary.

ALLBASE/SQL follows the SQL standard in allowing database table names to be qualified by an ownername. PowerHouse also supports an ownername on record names. You can specify full table names on QDESIGN FILE statements.

The general syntax for a full table name in PowerHouse is:

[owner.]table [IN database]

Both owner and table are identifiers. PowerHouse imposes an overall maximum of 64 characters for the combined length of owner.table. PowerHouse upshifts the owner and table unless the **noshift** program parameter or the SET NOSHIFT statement is specified. The owner name SYSTEM is reserved for ALLBASE/SQL system tables, and can be used to access metadata of the database.

An implicit alias of the table name is assumed when an owner.table is first encountered. For example,

```
> ACCESS OWNNAM.TABNAM IN TESTTABL
```

implicitly assumes an alias of TABNAM.

All ALLBASE/SQL DBEnvironment entities, such as modules or tables, have owners. If a program needs to access an entity owned by another user, you specify the owner as part of the entity name.

By default, the owner of a module created by PowerHouse is

MPE/iX:	USERNAME@ACCOUNTNAME, with USERNAME being the name of the user running the component, and ACCOUNTNAME the logon account. To permit an application builder to specify an owner, the **owner** program parameter is available for all components. For example, `:QDESIGN INFO="OWNER=CHRISB@DOC"`
OpenVMS, UNIX, Windows:	USERNAME, which is the user's logon name. To permit an application builder to specify an owner, the **owner** program parameter is available for all components. For example, `qdesign owner=scott1`

If you use the **owner** program parameter, modules created by PowerHouse are owned by the specified ownername. Also, any unqualified table names are qualified with this ownername.

To create modules owned by another ownername, ALLBASE/SQL security requires that you have Database Administrator (DBA) authority.

The following are valid types of ownernames:

- authorization group

- class
- user logon name
- **MPE/iX:** username@accountname

IN file

The dictionary file name where the record-structure is located.

*subfilespec

The name of the subfile.

Limit: The subfile must exist; QDESIGN cannot create subfiles.

Options

FILE Options		
type	ALIAS	AUTOCOMMIT
CACHE	CLOSE	COUNT
MYVIEW	NEED|ALL	NOAPPEND
NODELETE	NOITEMS	OCCURS
OCCURS WITH	OPEN	SIGNAL|NOSIGNAL
TRANSACTION	WAIT|NOWAIT	

type

Specifies the relationship of the file to the screen and to other files on the screen.

The general term type must be one of the following:

AUDIT [WITH record]	DELETE	DESIGNER
DETAIL	MASTER	PRIMARY
REFERENCE	SECONDARY	

Default: If no type is given, PRIMARY is assumed, except for files that are included in the receiving list of the screen statement. Files that are passed from higher-level screens are always assumed to be MASTER.

Each of the file types is discussed in detail in the sections that follow.

AUDIT [WITH record]

An AUDIT record-structure is used to record data changes in a file, and is normally associated with another record-structure (the file to be audited) as in

```
> FILE EMPLOYEES
> FILE AUDITRECORD AUDIT WITH EMPLOYEES
Item EMPLOYEE initialized (fixed) to EMPLOYEE of EMPLOYEES.
```

When QUICK performs an update on the associated file, an AUDIT record is also written. If the AUDIT record-structure is a SEQUENTIAL or DIRECT file, each new audit record is appended to the end of the file. (The file is opened for append access.) Indexed files are opened for update access.

The WITH option specifies the record-structure whose updates are to be recorded. The AUDIT data record is updated automatically when its associated record-structure is updated. No PUT verb is generated in the UPDATE procedure for the record-structures in AUDIT files.

If the WITH option is not specified, then QDESIGN assumes no associated file exists. Do not specify the WITH option if you want to use one AUDIT file to audit more than one file on the screen. If the WITH option isn't specified, the data record is treated as an add-only data record and a PUT verb is generated in the UPDATE procedure for the AUDIT file; however the screen designer is responsible for updating the record status of the AUDIT file. If the designer does not update the record status of the AUDIT file, no audit record is written.

A record-structure in an AUDIT file is automatically passed to a lower-level screen when its associated record-structure is passed. The record-structure in an AUDIT file can't be explicitly passed and shouldn't be declared again on the lower-level screen. However, if the MYVIEW option has been included with the FILE statement on the lower-level screen, a record-structure in an AUDIT file on the higher-level screen is ignored. This permits you to declare a record-structure in an AUDIT file on the lower-level screen that relates to that screen alone.

A data record written to the AUDIT file is an "after-image". An "after-image" documents the status of the file or "new" values at that point. The "before-image" (previous file status) is documented by an earlier data record produced when the data record was written to the file. Because there is no "after-image" for deleted data records, the record content prior to deletion is written to the AUDIT file.

An AUDIT record-structure must be defined in the data dictionary. The record-structure typically consists of all the items in the associated file, including items that document details such as time, date, user, and type of change.

DELETE

A DELETE record-structure allows data records to be deleted from a file that isn't otherwise declared on the screen. These data records are meant to be deleted in relation to deletions of primary or secondary data records on the current screen. A DELETE record-structure should occur the same number of times as the record-structure to which it is related, as in

```
> FILE EMPLOYEES OCCURS 2
> FILE SKILLS DELETE OCCURS WITH EMPLOYEES
Item EMPLOYEE initialized (fixed) to EMPLOYEE of EMPLOYEES.
```

A DELETE record-structure can't be passed to a lower-level screen.

DESIGNER

Specifies a record-structure that is totally under the designer's control. QUICK makes no assumptions about a DESIGNER record-structure, except that, by default, its associated file is opened for read and write access. A record-structure in a DESIGNER file can be passed to a lower-level screen where it should also be declared as type DESIGNER.

DETAIL

A DETAIL record-structure is used when there are a variable number of data records for each PRIMARY file record, as opposed to the fixed relationship that can be described with a SECONDARY file. The relationship of PRIMARY to DETAIL is one-to-many. Append processing can be used in a screen that has a DETAIL record-structure.

The record-structures in a DETAIL file can be passed to a lower-level screen and declared as type MASTER on the receiving screen.

QDESIGN initializes all identically named items in the DETAIL file to their value in the PRIMARY file.

Limit: One DETAIL record-structure per QUICK screen. You can't declare both a PRIMARY and a DETAIL record-structure with multiple occurrences in the same screen design.

MASTER

Specifies a record-structure in a PRIMARY, SECONDARY, or DETAIL file passed from a higher-level screen to the current screen. Record-structures received on a screen are assumed to be in MASTER files if not specified. They can in turn be passed to a lower-level screen and declared as type MASTER on the receiving screen.

PRIMARY

Specifies the principal file accessed by the screen. Choose the most frequently needed file as the PRIMARY file for a screen.

There should always be one PRIMARY file for screens other than menu and slave screens. A record-structure in a PRIMARY file can be passed to a lower-level screen where it is declared as type MASTER on the receiving screen. The user activities Enter, Find, Change, and Delete are directed mainly towards the PRIMARY record-structure.

Append processing is available when there are multiple occurrences of the record-structure on a screen.

Limit: You can't declare both a PRIMARY and a DETAIL record-structure with multiple occurrences in the same screen design.

REFERENCE

Specifies a read-only file used for data validation or retrieval. A record-structure in a REFERENCE file is accessed automatically when an item in the record-structure is referenced. A record-structure in a REFERENCE file can be passed to a lower-level screen where it must be declared as type REFERENCE.

SECONDARY

Specifies a record-structure in a file that is also to be updated; it is related to record-structures in PRIMARY or DETAIL files.

If the SECONDARY file is related to the PRIMARY file, QDESIGN initializes all identically-named items in the SECONDARY file to their value in the PRIMARY file. The relationship of PRIMARY to SECONDARY is 1:n where "n" is the fixed number in the OCCURS clause.

If the SECONDARY file is related to the DETAIL file by an OCCURS WITH option, QDESIGN first initializes all identically-named items in the SECONDARY file to their value in the DETAIL file. QDESIGN initializes any remaining identically-named items in the SECONDARY file to their value in the PRIMARY file. The relationship of DETAIL to SECONDARY is 1:1.

The record-structures in a SECONDARY file can be passed to a lower-level screen and declared as type MASTER on the receiving screen.

ALIAS name

Assigns an alternative name to the record-structure. When a record-structure is declared more than once in a screen design, the ALIAS option assigns a unique identifier name for each declaration. Once the alias is assigned, subsequent references to the record-structure must use this name.

AUTOCOMMIT

Indicates that the transaction performing the retrieval from the reference file is automatically committed after the retrieval is completed. Automatic retrievals include retrievals from lookups or implicit retrievals of reference file items for display.

Use AUTOCOMMIT to ensure that the reference files retrieval sees the most recent version of the database. This option can also terminate the retrieval transaction immediately and, therefore, release any locks that it may have acquired. (You may want to specify a distinct transaction for the lookup.)

Limit: Valid for REFERENCE files only.

CACHE [n]

Specifies that QUICK is to maintain more primary or detail record buffers than can be displayed on the screen. These record buffers may be accessed programmatically by the screen designer and browsed by the screen user.

If you do not specify CACHE, the cache size is set to the number of occurrences as specified with the OCCURS option for this record structure. If the OCCURS option is not used, the size of the cache is set to one.

Limit: This option may only be used with either the PRIMARY or a DETAIL record structure, but not both.

n

Specifies the upper limit for the number of record buffers in the cache. If you are concerned that the size of the record combined with the associated detail and secondary records will use excessive memory, use this option to specify an upper limit.

If not specified, QUICK sets the size based on the current requirements and will grow dynamically up to 255 record buffers.

The minimum size of the cache is the number specified on the OCCURS option. If this option is not used, the minimum size is one.

Limit: 1 to 255

For more information about the CACHE option, see Scrolling Records on (p. 140).

CLOSE

Closes the file when leaving the current screen for a higher-level screen. If the file is opened on a higher-level screen, the CLOSE option has no effect.

COUNT [NEGATIVE] [INTO] item1 [,[NEGATIVE] [INTO] item2]...

Uses the named items to maintain a count of the data records entered into this file. The named items should normally be in record-structures in higher-level MASTER files, so the proper value is maintained from one screen to the next.

The count is automatically incremented when data records are entered and reduced when data records are deleted. For record-structures in DELETE files, the count is decremented when the record-structure is actually updated (that is, when the PUT verb is executed). The NEGATIVE option reverses these activities. INTO is used only for documentation.

Limit: The maximum number of items that can be counted into is 21.

MYVIEW

Overrides the current screen's default view of the record-structure. This option is used when there is more than one data record type (for example, as in files with more than one record-structure) in the file and the screen is receiving the record-structure from a higher-level screen that has a different view of the data record. The MYVIEW option can also be used with the same data record layout to allow processing specific to that screen's copy of the data record. When used with record-structures in MASTER files, the MYVIEW option tells QUICK to ignore ITEM FINAL options and record-structures in AUDIT files from higher-level screens. When the MYVIEW option isn't used, ITEM FINAL options and record-structures in AUDIT files are ignored on lower-level screens.

NEED n|ALL

Places the specified number of data records (n or ALL) on file, regardless of the current data record status. This option causes the PUT verb to treat the data record as changed, even if it isn't. To have any effect, the PUT verb must still be executed for the NEED option. This option can be used to add blank data records or data records with initial and final values only.

Limit: Valid for record-structures in PRIMARY, SECONDARY, DETAIL, and DESIGNER files. Also valid for record-structures in AUDIT files that have not had the WITH option of the FILE statement specified.

n

Specifies the minimum number of occurrences of the data record to add to the file. At least this many occurrences of data records must be declared on the screen.

ALL

Indicates that the number of data records to add to the file is the number of data records of this file declared on the screen (as specified by the OCCURS option).

NOAPPEND

Suppresses the automatic generation of the APPEND procedure and PERFORM APPEND verb for a record-structure in a repeating PRIMARY file. If this option is specified, Append processing can't be used for the record-structures in the repeating PRIMARY file.

Limit: Valid only for record-structures in PRIMARY files.

NODELETE

Suppresses the automatic generation of a DELETE verb for this record-structure in the DELETE and DETAIL DELETE procedures.

NOITEMS

Doesn't generate automatic initialization for items in this record-structure.

For more information on automatic initialization, see (p. 153).

OCCURS n [TIMES]
OCCURS WITH [ITEM] item |[FILE] record-structure

Repeats the data records on the screen.

n [TIMES]

Specifies the size of the occurrence window, that is how many record buffers in the cache are to be displayed on the screen.

Limit: If caching is used, the OCCURS n option can only be used on one data structure. The OCCURS n option must appear on the same data structure as the CACHE option. The range for n is 1 to 255.

For more information about the OCCURS option and the record cache, see Scrolling Records on (p. 140).

WITH [ITEM] item|[FILE] record-structure

Repeats the data records of this record-structure as many times as the specified record-structure or item repeats on this screen.

OPEN [n|{EACH LEVEL}] [DBMODE n]|[access-type] [exclusivity] [GLOBAL]

Specifies the OPEN options.

n

Forces QUICK to perform a separate open for the file's associated record-structures, even if the OPEN mode is otherwise compatible with a previous open.

Limit: 0 to 16

EACH LEVEL

Forces QUICK to perform a separate open for the file's associated record-structures at each screen level where a screen refers to the record-structure. At each level, the open number is identical to the screen level number.

For relational databases, the OPEN EACH LEVEL option assigns a transaction number to the database read operations. As with files, the assigned transaction number is to be the same as the screen level number.

Limit: OPEN n|EACH LEVEL are ignored for tables in ALLBASE/SQL databases.

File Opens for Relational Databases

The OPEN n and OPEN EACH LEVEL options of the FILE statement define a specific transaction that QUICK uses for database access.

This method is obsolete and should not be used in new applications. Use of the TRANSACTION statement is the recommended method.

DBMODE n (MPE/iX, UNIX, Windows)

Used to specify the open mode for IMAGE and Eloquence databases. IMAGE supports open modes from 1 to 8. Eloquence accepts open modes from 1 to 9. The new mode, DBMODE 9, only allows PowerHouse to read the database, but allows other concurrent users of the database to read and update data. Eloquence only fully supports modes 1, 3, 8 and 9. All other modes are mapped to one of these supported modes. This means that if a PowerHouse application uses modes 2, 4, 5, 6 or 7, it might not give the same results with Eloquence as it does with IMAGE.

For more information about open modes, refer to your IMAGE or Eloquence documentation.

GLOBAL

By default, threads do not share file opens. This option lets another thread share a file open as long as the GLOBAL option is used on the FILE statement on each screen and the OPEN numbers are the same.

This option has no impact on relational transactions.

access-type

The access-type options are:

APPEND	Opens a record-structure's associated file for write access only. Data records are added to the file after any that already exist. Limit: Not valid for indexed record-structures or relational tables.
CLEAR	Opens a record-structure's associated file for write access only. The CLEAR option deletes all data records in the file when processing begins. Limit: Not valid for relational tables and indexed files
READ	Opens a record-structure's associated file for read access only.
UPDATE	Opens a record-structure's associated file for read and write access. The UPDATE option allows existing data records to be updated in place. Limit: Not valid for SEQUENTIAL files.
WRITE	Opens a record-structure's associated file for write access only. **MPE/iX, UNIX, Windows**: For direct, relative (MPE/iX), and sequential files, the WRITE option deletes all data records in the record-structure when processing begins. The file can be read and written to by others unless the exclusivity option specifies otherwise. **OpenVMS**: For direct, relative, and sequential files, records written overwrite existing records starting at the beginning of the file. Any data not overwritten is retained. The file cannot be read or written to by others until it is closed. **MPE/iX, OpenVMS, UNIX, Windows**: For indexed files, adding a record that has the same unique key value as an existing record causes the existing record to be replaced. Adding a record with a new key value causes that record to be added. Existing data is retained. The file cannot be read or written to by others until it is closed. Limit: Not valid for relational tables.

exclusivity

The exclusivity options are:

EXCLUSIVE	No other user or application can open the record-structure's associated file. The file can't be opened again by the current or any other process until it has been closed. Limit: Not valid for sequential files (**UNIX**) or relational databases. Limit: UPDATE EXCLUSIVE is not valid for **DISAM** files.
SEMIEXCLUSIVE (**MPE/iX, OpenVMS**)	No other user or application can open the record-structure's associated file for write access. If the file is already opened for writing by either the current application or another application, the SEMIEXCLUSIVE open fails and QUICK issues an error message. Limit: Not valid for relational databases.
SHARE	Another user or application can open the record-structure's associated file for read or write access.

Default: SHARE

SIGNAL|NOSIGNAL [ON CLOSE] (OpenVMS)

SIGNAL sends (posts) an End-of-File (EOF) signal to the mailbox when the process closes the file; NOSIGNAL does not.

Limit: Applies to Mailbox files (MBX) only.

Default: NOSIGNAL

TRANSACTION transaction_name [FOR {CONSISTENCY| {[CONCURRENCY] phase-option[,phase-option]...}}]...

Defines transactions used for relational data structures.

TRANSACTION

Specifies that the transaction is associated with the relational table.

Full transaction support is only available in relational databases.

NOCOMMIT is used by the Query transaction of any screen that contains a MASTER file that was passed to the screen.

transaction_name

Any valid PowerHouse name.

FOR CONSISTENCY

Determines that a relational data structure is associated with a particular transaction in Consistency model.

Limit: Only one transaction association can be specified.

FOR [CONCURRENCY] phase-option [,phase-option]...

Determines that the relational data structure is associated with a particular transaction or transactions in Concurrency model.

Limit: Up to three transaction associations can be specified, one per phase.

phase-option

Specifies the screen phase with which the transaction is associated.

The phase options are:

PROCESS	The phase in which you are entering, correcting, or changing data records on the screen.
QUERY	The phase in which data is retrieved from the database.
UPDATE	The phase in which data is updated.

If no model is specified, as in the following example, the transaction is associated with the file for all phases in all screen modes:

```
> FILE EMPLOYEES
> TRANSACTION QUERY
```

By default, all relational data structures are associated as follows:

Model	Transaction	Phase
Concurrency	Query	Query[1]
	Update	Process
	Update	Update
Optimistic	Query	Query[2]
	Update	Update[3]
Consistency	Consistency	Consistency
Dual	Query	Query
	Update	Process
	Update	Update
	Consistency	Consistency

[1]*Note that for a screen allowing only the activity FIND, by default read-only data structures are associated with the Query transaction for all phases.*

[2]*All "read" activities are associated with the transaction associated with the Query phase. By default, this is the Query transaction.*

[3]*All "write" activities are associated with the transaction associated with the Update phase. By default, this is the Update transaction.*

WAIT|NOWAIT [ON SEND|RECEIVE|FULL] (OpenVMS)

WAIT specifies that the process waits for the message from the mailbox; NOWAIT does not. When WAIT is specified without options, it is set for all subsequent options.

Default: NOWAIT

Limit: Applies to Mailbox files (MBX) only.

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

ON SEND

WAIT ON SEND waits for the message sent to a mailbox to be received before returning control to the process; NOWAIT ON SEND does not.

ON FULL

WAIT ON FULL specifies that if a message is sent to a full mailbox, the process waits until the message can added before returning control to the process. NOWAIT ON FULL specifies that if a message is sent to a full mailbox, that control be returned to the process immediately and an error status is issued.

ON RECEIVE

WAIT ON RECEIVE indicates that when reading messages the process will wait for a message to be posted if one is not already there. When using this option, an empty mailbox is not treated as an End-of-Data condition; instead, and End-of-File condition is always processed as an End-of-Data condition and could be used to control processing. NOWAIT ON RECEIVE specifies that a message should be returned if one exists, but if the mailbox is empty, control should be returned to the process immediately.

Discussion

The FILE statement is part of the data section of the screen design. It names a record-structure and describes the relationship of that record-structure to the screen. The FILE statements for a screen design describe the interface between the screen and the record-structures.

Files, Record-structures, and the Screen

The relationship between files, record-structures, and the screen can be further defined by the ACCESS, ITEM, SELECT, and TARGET statements, each of which overrides FILE statement options.

The ITEM statements must be declared on the screen where the file and record-structures are first declared. If the file is received from a higher-level screen, the ITEM statements at this level are ignored. If a file with the MYVIEW option specified is received from a higher-level screen, only ITEM statements declared on the lower-level screen are valid.

Declaring Record-structures in the Correct Order

In the screen design, QUICK record-structure types should be declared in the following order so that items can be initialized automatically with values from preceding files:

1. MASTER
2. PRIMARY
3. AUDIT associated with PRIMARY
4. DELETE associated with PRIMARY
5. SECONDARY associated with PRIMARY
6. AUDIT associated with SECONDARY
7. DELETE associated with SECONDARY
8. DETAIL
9. AUDIT associated with DETAIL
10. DELETE associated with DETAIL
11. SECONDARY occurring with DETAIL
12. other

The record-structures in MASTER, PRIMARY, SECONDARY, DETAIL, and DELETE files are updated when the QUICK screen user updates the screen. Note, however, that record-structures in DELETE files are updated only if the data records are marked for deletion. The record-structures in AUDIT files are updated automatically with their related record-structures. The record-structures in REFERENCE files are not updated.

The Relationship of PRIMARY and SECONDARY Record-structures

The major user activities of entering, finding, changing, and deleting data can be directed against PRIMARY and closely-related SECONDARY record-structures' associated files.

Both PRIMARY and SECONDARY data records can be maintained from the same screen. The situations under which this occurs can be termed extended record, inverted master, and fixed SECONDARY records.

Extended Record

An extended record occurs when two record-structures are similar but the second record-structure has different items. The record-structures may be partially redundant for historical reasons. The extended record situation may also be one logical record that, for some reason, is split into two physical records.

For example, assume that the file BRANCHES has two items: BRANCH and BRANCHNAME. Another file, CONTRIBUTIONS, is indexed by BRANCH. However, CONTRIBUTIONS includes information other than BRANCH, such as the date the branch was formed, contribution number, and last year's contribution to company profit.

Both files can be declared in one screen design:

```
> SCREEN EXAMPLE
> FILE BRANCHES
> FILE CONTRIBUTIONS SECONDARY
Item BRANCH initialized (fixed) to BRANCH of BRANCHES.
> SKIP 1
> GENERATE NOLIST
> BUILD
```

Inverted Master

When the record-structures are in an inverted master situation, the subordinate record-structure is of primary interest, while the record-structure normally in a primary role is of secondary interest.

A typical example would be two files, OWNERS and PATENTS; each owner may own many patents but patents are of primary interest. The user won't want to go through a separate OWNERS screen for every record of the PATENTS file. It's best to enter patents on a PATENTS screen, look up and display the owner's name for validation, and update any new owner name from the detail screen.

To do this, make the OWNERS file a secondary file and include the nodelete option, as in

```
> FILE OWNERS SECONDARY NODELETE
```

The NODELETE option prevents deletion of the OWNERS file record when the user deletes a PATENTS file record (because the owner may have other patents). Owners must be deleted on the OWNERS screen. New owners, however, can be added on the PATENTS screen.

The design for this inverted master looks like this:

```
> SCREEN PATENT
> FILE PATENTS
> FILE OWNERS SECONDARY NODELETE
Item OWNER initialized (fixed) to OWNER OF PATENTS.
> FIELD OWNER OF PATENTS LOOKUP ON OWNERS OPTIONAL
> FIELD OWNERNAME OF OWNERS IF NEWRECORD OF OWNERS
```

When the user enters a value for OWNER on the PATENTS screen, QUICK retrieves and displays the related OWNERS file record. If no record exists, the user is prompted for a value of OWNERNAME which gets updated on the OWNERS file.

Fixed Data Records

When a fixed number of SECONDARY records exist for every PRIMARY record, the user can handle a transaction on one screen. The fixed number of SECONDARY records (per SECONDARY file) associated with a PRIMARY record is physically limited by the screen display capacity.

If the PRIMARY record-structure occurs more than once, however, the SECONDARY record-structure must occur the same number of times. For example, either the PRIMARY record-structure occurs once and the SECONDARY record-structures occur a fixed number of times, as in:

```
> FILE A
> FILE B SECONDARY OCCURS 2
```

```
> FILE C SECONDARY OCCURS 6
```

or the PRIMARY record-structure repeats and the SECONDARY record-structures occurs on a one-to-one basis with the PRIMARY, as in:

```
> FILE A OCCURS 3
> FILE B SECONDARY OCCURS WITH A
> FILE C SECONDARY OCCURS WITH A
```

Relating SECONDARY Record-structures to Repeating Record-Structures

You can relate SECONDARY record-structures to both the PRIMARY and the DETAIL record-structures in the same screen design. To relate a secondary record-structure (or other types of record-structures) to a detail record-structure, use the OCCURS WITH option of the FILE statement. QDESIGN generates the correct procedures automatically. A SECONDARY record-structure that is related to the primary record-structure is processed in the ENTRY, FIND, and DELETE procedures. A SECONDARY record-structure that occurs with the DETAIL record-structure is processed in the APPEND, DETAIL FIND, and DETAIL DELETE procedures.

Retrieval Assumptions for PRIMARY Record-Structures

QUICK prepares to retrieve data records from the PRIMARY file when the screen user specifies Find mode in the Action field of the screen. The screen user can choose one of several retrieval alternatives by supplying a value for any of the indexes in the primary record. If the screen user doesn't specify a value, retrieval is sequential. Supplying values must be done in the PRIMARY file.

The standard Find mode sequence prompts the screen user for a value for each segment of each index in the primary data record. QDESIGN automatically provides the sequence for these values. To get this prompt, a field (or fields) must exist on the screen into which the user can enter a segment value. If QDESIGN can't find fields to prompt the user for a value (or values, for multiple-segment indexes), a retrieval alternative for that index is not allowed.

QUICK retrieves PRIMARY data records based on the first complete segment value entered by the user (in response to prompts in index fields). If the user simply presses [Return] in response to all index field prompts, QUICK retrieves primary data records sequentially by default.

In a hierarchical screen relationship, retrieval alternatives are constructed for the PRIMARY record-structure of the highest-level data screen only. For a hierarchical relationship to exist, the values of lower-level PRIMARY file indexes are determined from values in MASTER file data records passed from higher-level screens. If the files have identically-named items, QDESIGN can establish the retrieval mechanism without any help.

In more complex situations, such as where the names of items don't agree, or where a calculation must be performed, you must supply an access statement. For instance, if the skills record-structure is indexed by employee (an item in the employees record-structure), QDESIGN automatically constructs a retrieval mechanism for the skills record-structure based on the value of the item employee in the employees record-structure, as in

```
> SCREEN SKILL RECEIVING EMPLOYEES
> FILE EMPLOYEES MASTER
> FILE SKILLS
Item EMPLOYEE initialized (fixed) to EMPLOYEE of EMPLOYEES.
```

But if the segment has different names in the employees and skills record-structures, QDESIGN can't establish the retrieval mechanism automatically. If the item is named employeeno in the employees record-structure and employee in the skills record-structure, you can supply this information with an access statement, as in

```
> SCREEN SKILLS RECEIVING EMPLOYEES
> FILE EMPLOYEES MASTER
> FILE SKILLS
> ACCESS VIA EMPLOYEE USING EMPLOYEENO OF EMPLOYEES
```

An alternative design to supply the same information is:

```
> SCREEN SKILL RECEIVING EMPLOYEES
> FILE EMPLOYEES MASTER
> FILE SKILLS
>   ITEM EMPLOYEE   &
>   INITIAL EMPLOYEENO OF EMPLOYEES FIXED
```

```
> ACCESS VIA EMPLOYEE
```

In this case, the item EMPLOYEENO's current value from the EMPLOYEES record-structure is put directly into the record buffer for the item EMPLOYEE with an ITEM statement. Access is performed using this value. The ITEM statement makes the USING option unnecessary.

The number of primary data records retrieved depends on the occurs option. In this example,

```
> FILE EMPLOYEES OCCURS 2
```

QUICK retrieves primary data records two at a time.

QUICK always requires retrieval of at least one primary data record. If no primary data record exists, processing of the transaction doesn't proceed.

Retrieval Assumptions for SECONDARY Record-Structures

The retrieval of secondary data records is based on retrieval of the primary data record. Once a primary data record is retrieved, secondary data records are retrieved based on identically-named items in PRIMARY and MASTER files. This retrieval is assumed to be optional in the FIND procedure. Retrieval of secondary data records in other situations (such as by a lookup) isn't optional.

Retrieval Assumptions for MASTER Record-Structures

Data records for a MASTER record-structure are assumed to have been retrieved on a higher-level screen. If not specified, passed record-structures are assumed to be MASTER record-structures.

Retrieval Assumptions for REFERENCE Record-Structures

The retrieval from REFERENCE files is unique, such that retrieval always gets the first data record of the chain. Although automatic retrieval is assumed to be optional, other types of retrieval are required.

Automatic Retrieval of REFERENCE Record-Structures

A slightly different process occurs for what is called "automatic retrieval" of data records from REFERENCE record-structures' associated files. In the following situation,

```
> FILE EMPLOYEES
> FILE POSITIONS REFERENCE
.
.
.
> FIELD POSITION LOOKUP ON POSITIONS
> FIELD POSITIONTEXT DISPLAY ID SAME
```

no explicit retrieval information is specified. However, QUICK can construct retrieval information to perform the lookup using the third method of record retrieval, described previously. QUICK determines that the linkitem POSITION of the POSITIONS record-structure is also an item of the PRIMARY record-structure, and uses the current value of the item POSITION to perform the lookup. The current value of POSITION is provided when the user enters a value in the POSITION field on the screen. Since the appropriate data record from the POSITIONS file will be brought into the record buffer for the lookup, the display of the item POSITIONTEXT from that data record can take place directly.

The situation changes in Find mode. When displaying retrieved data in Find mode, no lookups take place. As a result, the REFERENCE file is not retrieved. In this case, QDESIGN performs an "automatic retrieval". This data is retrieved in the following manner:

1. QUICK looks for an ACCESS statement associated with the REFERENCE record-structure.
2. If there is no ACCESS statement, QUICK looks for retrieval information specified with the first LOOKUP ON option or GET verb associated with the record-structure.
3. If retrieval specifications from steps 1 and 2 are not available, then QDESIGN attempts to match a linkitem of the REFERENCE record-structure with an identically-named item in a PRIMARY, SECONDARY, or MASTER record-structure declared on the screen. A LOOKUP has an implied USING which is used if no overriding USING option is given.

Multiple Retrieval Methods for REFERENCE Record-Structures

Automatic retrieval of REFERENCE files can lead to unexpected results if multiple retrievals (through either the LOOKUP option or the GET verb) are performed on the same REFERENCE file using different retrieval methods to display or to reference data.

For example, if two fields have LOOKUP options on the same REFERENCE file using different indexes, both lookups are done as expected and the two fields are validated. However, if you attempt to display data from the two lookups, you will not get the expected results. Referencing the REFERENCE record-structure items causes automatic retrieval to be performed using the automatic retrieval method established in QDESIGN.

In the following screen design, the first LOOKUP option establishes the automatic retrieval method via the linkitem EMPLOYEE, using EMPLOYEE of EMPLOYEES:

```
> FILE NAMEINDEX PRIMARY
> FILE EMPLOYEES REFERENCE
> FIELD EMPLOYEE OF NAMEINDEX LOOKUP ON EMPLOYEES
> FIELD BRANCH OF EMPLOYEES DISPLAY
> FIELD LASTNAME OF NAMEINDEX LOOKUP ON EMPLOYEES
> FIELD FIRSTNAME OF EMPLOYEES DISPLAY
```

If (in the entry sequence) the user enters values for EMPLOYEE and LASTNAME that retrieve different EMPLOYEES records, the values for BRANCH and FIRSTNAME are both taken from the EMPLOYEES data record that corresponds to the first retrieval (using the item EMPLOYEE). The sequence of events can be described as follows:

1. The user enters a value for EMPLOYEE. QUICK attempts to retrieve an EMPLOYEES data record using the value in EMPLOYEE. If retrieval succeeds, processing continues; otherwise, the user is reprompted at the EMPLOYEE field. Since this is the first retrieval of the REFERENCE file for the transaction, the value QUICK uses to retrieve the EMPLOYEES data record becomes the value for automatic retrieval. The index for automatic retrieval (via EMPLOYEE) and where to get the value (using EMPLOYEE OF NAMEINDEX) is established by QDESIGN when the screen is built.
2. When the item BRANCH is referenced for display, QUICK compares the value established for automatic retrieval with what is in the buffer for the EMPLOYEES file. If the value is the same, QUICK does not have to retrieve an EMPLOYEES data record based on automatic retrieval, since it already has the correct data record. QUICK then displays the value for BRANCH.
3. The user enters a value for LASTNAME. QUICK attempts to retrieve an EMPLOYEES data record using the value in LASTNAME. Since no REFERENCE file item is being referenced, automatic retrieval is not performed, and retrieval proceeds normally based on the value in LASTNAME. If the retrieval succeeds, processing continues; otherwise, the user is reprompted at the LASTNAME field.
4. When the item FIRSTNAME is referenced for display, QUICK compares the value for EMPLOYEE that was established for automatic retrieval (in step 1) with the value in the buffer for the EMPLOYEES file. If the value is the same, the value for FIRSTNAME is displayed. If the value is different, QUICK retrieves the EMPLOYEES data record it first retrieved in step 1, and displays the value of FIRSTNAME from that data record.

If values from different retrieval paths must be displayed, the REFERENCE record-structure must be declared once for each different path using the ALIAS option on the FILE statement. An alternative is to use a DESIGNER record-structure to display values from different retrieval paths. Since there is no automatic retrieval for files other than REFERENCE files, the DESIGNER record-structure need only be declared once. However, if data is required for display in Find or Select mode, you must retrieve the required DESIGNER file data records by using a POSTFIND or DETAIL POSTFIND procedure, because QUICK does not retrieve these records automatically.

Retrieval Assumptions for DESIGNER Record-structures

The retrieval of data records from DESIGNER files is unique (similar to a 'GET first'), preventing the retrieval of data records down a chain. When you use FOR loop processing with a DESIGNER record-structure that has a repeating index, only the first data record in a chain is retrieved. Use WHILE RETRIEVING to read down a chain.

Retrieval Assumptions for DELETE Record-structures

When updating a DELETE file, QUICK makes successive reads and deletions of data records in the DELETE file using matching index values. QUICK continues to read and delete these records until no more related records are found. As with hierarchical relationships, retrieval for data records from DELETE files is constructed automatically, based on identically-named items in previously-declared files. Retrieval from DELETE files is always optional.

Using the TRANSACTION Option

If you use the TRANSACTION option but do not specify any transaction association option (or do not include an explicit phase list), then it is assumed that the transaction is to be used during all screen phases for database activities involving the file. For example,

```
> FILE EMPLOYEES IN LIFE &
> TRANSACTION MYTRANSACTION
```

For the Concurrency model, several transactions may be associated with one file, with each transaction being used for specific phases of screen processing, as in:

```
> FILE EMPLOYEES IN LIFE DETAIL &
> TRANSACTION QUERY FOR QUERY, PROCESS &
> TRANSACTION MYUPDATE FOR UPDATE
```

Restrictions on Passed Files

All received files at execution-time inherit the transaction (and its characteristics) that was associated with that file on the parent screen at the time the subscreen was called. A received file cannot be associated with a new transaction (or new transaction characteristics) on a subscreen.

Closing Relational Files

We recommend that you do not use the CLOSE verb (distinct from the SQL CLOSE) or the CLOSE option of the FILE statement on relational tables. The results are unpredictable if there are any uncommitted transactions related to the table or database when the CLOSE is performed.

When a CLOSE verb or option is used, QUICK will immediately commit all transactions against that database (not just the ones associated with the table), and logically detach from the database. If errors are encountered, a rollback is attempted, and rollback pending does not apply.

This behavior can affect performance, since the attach must be re-established before work can continue against that database. It may also affect data integrity, since unrelated transactions may be committed as a result of the CLOSE. In addition, if there are other physical databases involved in the same PowerHouse transaction, committing the transactions against only one database may result in inconsistent data.

If your intent is to commit one or more transactions, then we suggest you use the COMMIT verb. For more information, see the COMMIT verb on (p. 380) and the ROLLBACK verb on (p. 465). See also information about the transaction control options available on the SCREEN, FILE, FIELD, and TRANSACTION statements.

Subfile Support in QUICK and QDESIGN

Subfiles can be used in the same places and in the same way as direct files. They can be read either sequentially or by record number. New records can be added to the end of the subfile. Existing records can be overwritten. Records cannot be deleted once they've been added to the subfile. QUICK and QDESIGN use subfiles created by QUIZ and QTP; they cannot create subfiles.

Scrolling Records

The CACHE option specifies that QUICK creates a cache of record buffers to store retrieved and entered data records. As data records are found, QUICK automatically moves data records in and out of the cache. As the size of the cache may be larger than the number of occurrences displayed on the screen (as controlled by the OCCURS n option), users may scroll backwards and forwards through the cache, allowing them to view previous data records that have been scrolled off the screen.

For a cached primary record structure, users can scroll backwards and forwards through primary records retrieved by the FIND procedure. Scrolling is also supported for primary data records entered by way of the ENTRY and APPEND procedures.

For a cached detail record structure, users can scroll backwards and forwards through the detail records retrieved by the DETAIL FIND procedure associated with an individual primary record structure. Scrolling is also available for detail records entered by way of the APPEND procedure.

For more information about cached records, see (p. 57).

Caching Restrictions

The CACHE and OCCURS options have the following restrictions:

- The CACHE option can be used on either a PRIMARY or a DETAIL record structure but not both.
- A DETAIL record structure cannot be declared on the same screen as a PRIMARY record structure that has either the CACHE or OCCURS n option.
- If caching is used, the OCCURS n option can only be used on one record structure. The OCCURS n option must appear on the same record structure as the CACHE option.

If you want to cache a single PRIMARY record-structure, you need to indicate to QDESIGN what fields are to be scrolled by using the CLUSTER statement.

For example, if you have the screen:

```
> SCREEN ORDERS
> FILE ORDERS
> FIELD ORDER_NUMBER
.
.
.
```

you can modify the screen as follows:

```
> SCREEN ORDERS
> FILE ORDERS CACHE
> CLUSTER OCCURS WITH ORDERS
> FIELD ORDER_NUMBER
.
.
.
```

If you do not add the CLUSTER statement, QUICK will cache the PRIMARY record structure but has no mechanism to determine how and what to scroll. Consequently, QUICK Action commands such as NEXT RECORD and PREVIOUS RECORD do not work as expected.

Using Caches with Subscreens and Threads

A subscreen or thread may have an active cache independent of any cache created on the calling screen. Upon return to the calling screen, the cache for the calling screen is active. The cache for the subscreen is deleted when the screen is exited. For thread screens, QUICK maintains any caches that are in use, when the user toggles between screens.

Examples

The following QDESIGN statements track the frequency of deposits into an employee's pension fund.

- EMPLOYEES is the PRIMARY record-structure. NODELETE prohibits screen users from deleting EMPLOYEES records by suppressing the generation of a DELETE verb for the EMPLOYEES record-structure.
- BENEFITS is labeled SECONDARY because a screen can have only one primary record-structure, which in this case is EMPLOYEES.

```
> SCREEN PENSION
>
> FILE EMPLOYEES NODELETE
>
> FILE BENEFITS SECONDARY
```

```
>
> FILE PAY DETAIL OCCURS 10 &
>   COUNT INTO PENSIONPERIODS &
>   OF BENEFITS
```

The use of a detail file allows an employee with one data record in the employees record-structure to have multiple data records in the pay record-structure. The occurs option sets the number of times that the pay record-structure is displayed on the screen for data entry, in this case, 10. The count into option calculates the number of data records added to the pay record-structure and stores the number in the pensionperiods item of the benefits record-structure.

```
> FILE EMPDET REFERENCE
>  ACROSS VIAINDEX EMPLOYEENUMBER &
>  USING EMPLOYEENUMBER
```

The following EMPDET record-structure allows for data validation with LOOKUP ON options of FIELD statements. For more information, see the FIELD statement on (p. 105).

- Data from the EMPDET record-structure can also be displayed. By default, write and delete access are not available for a record-structure in a REFERENCE file.
- Data records in the SKILLS record-structure are marked for deletion if corresponding data records in the PAY record-structure are deleted.

```
> FILE SKILLS DELETE OCCURS WITH PAY
>
> TITLE "Employee Benefits" AT 1,30
> DRAW FROM 7,1 TO 9,80
> DRAW FROM 15,1 TO 15,80
> SKIP 2
> FIELD EMPLOYEENUMBER OF EMPLOYEES &
>    REQUIRED NOCHANGE &
>    LOOKUP NOTON EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> FIELD CITY OF EMPLOYEES
> FIELD EMPLOYEEAGE OF EMPDET &
>    DISPLAY
>
> HILITE TITLE INVERSE
> TITLE "Pay Period Pension" AT 8,4
> TITLE "Pay Period Pension" AT 8,44
> SKIP 1
>
> ALIGN (1,,4) (,,22)
> CLUSTER OCCURS WITH PAY FOR 1,40 VERTICAL
>
> FIELD PAYPERIOD OF PAY
> FIELD PENSIONTOTAL OF BENEFITS
> CLUSTER
> BUILD
SKILLS accessed via EMPLOYEENUMBER.
BENEFITS accessed via EMPLOYEENUMBER.
PAY ACCESSED VIA EMPLOYEENUMBER.
```

In this example, the AUTOCOMMIT option ensures that the most recently committed branch information is available. The LOOKUP transaction is defined separately to allow the lookups to be independent of the processing of the EMPLOYEES record. Without a separate transaction, the Query or Update transaction is committed.

```
> SCREEN EMPLOYEES
> TRANSACTION LOOKUP READ ONLY
> FILE EMPLOYEES IN LIFE
> FILE BRANCHES IN LIFE REFERENCE &
>   TRANSACTION LOOKUP AUTOCOMMIT
```

Several transactions may be associated with one file (this is the default for Concurrency), with each transaction being used for specific phases of screen processing.

If you use the TRANSACTION option but do not specify any transaction association option (or do not include an explicit phase list), then it is assumed that the transaction is to be used during all screen phases for database activities involving the file.

GENERATE

Generates FIELD statements.

Syntax

GENERATE [DETAIL|NODETAIL] [LIST|NOLIST]

DETAIL|NODETAIL

DETAIL writes the results of the GENERATE statement, rather than just the GENERATE statement itself, to QDESIGN's temporary source statement save file: QKSAVE (**MPE/iX**) or qksave.qks (**OpenVMS, UNIX, Windows**).

NODETAIL writes just the GENERATE statement to the temporary source statement save file.

Default: DETAIL

LIST|NOLIST

LIST causes QDESIGN to display the results of the GENERATE statement on your terminal; NOLIST doesn't.

Default: LIST

Discussion

The GENERATE statement is part of the layout section. It is a designer convenience that automatically generates FIELD statements for all items in the declared PRIMARY, SECONDARY, and DETAIL files.

The GENERATE statement has no effect on a slave screen.

General Rules for Generating Items

When selecting items to generate in fields, QDESIGN follows three basic rules:

1. The last redefinition of an item (or the substructure of that redefinition) is generated as a field, or
2. The first substructures of an item are generated as fields, or
3. The item is generated as a field.

Assumptions Made by the GENERATE Statement

The GENERATE statement makes the following assumptions:

- The LOOKUP NOTON option is generated for segments of any unique index. If a field is a segment of a unique index of either a PRIMARY or SECONDARY record-structure, QDESIGN generates this field and all subsequent fields in index order. If the unique index is made up of multiple segments, QDESIGN groups all the related segments together under the same number and applies the LOOKUP NOTON option to the last segment generated in this manner. If segments of other unique indexes are encountered during this process, the other segments of those indexes are grouped under the same ID-number as well.
- All lookups generated for indexes made up of multiple segments also contain a USING expression list that names the segments as values. This method ensures that the GENERATE statement builds in the same order as record items.
- If the designer has not specified an alignment, and if there are more fields generated than can fit on the screen using the default alignment, the alignment generated is
- (ID 1, LABEL 4, DATA 21) (ID 41, LABEL 44, DATA 61)
- QDESIGN generates the REQUIRED and NOCHANGE options for a field that corresponds to an index or index segment.
- QDESIGN generates the CHARACTER FOR 1,18 option for a field that is a blob datatype.

- If a data entry field is too long for the screen, the SIZE option and an applicable value are automatically included in the FIELD statement. The field size truncates to comply with screen limits.
- If there is a record-structure in a REFERENCE file, QDESIGN examines the indexes in that record-structure in the sequence established in the data dictionary. It looks for a segment that matches one of the items of the record-structure in the PRIMARY file and generates a LOOKUP ON option.
- A CLUSTER statement is generated for a record-structure in a PRIMARY, DETAIL or SECONDARY file with multiple occurrences.

Automatic Exclusion of Null Values in QDESIGN

The GENERATE statement automatically assigns field attributes that correspond to the database integrity constraints. If you use GENERATE to produce the FIELD statements for a QUICK screen for a relational data structure, QDESIGN generates the NULL VALUE NOT ALLOWED attribute for fields that correspond to database columns with the NOT NULL attribute. QUICK doesn't accept the entry of a null value in that field.

Generating the REQUIRED Option in the GENERATE Statement versus the NULL VALUE NOT ALLOWED Option in the FIELD statement

QDESIGN automatically generates the REQUIRED option for all fields that correspond to relational items on which indexes are declared. The FIELD statement option, NULL VALUE NOT ALLOWED applies only to items in relational data structures, and prevents you from entering the null value character in the field. The REQUIRED option prevents you from entering a null response (that is, simply pressing [Return]) to a field prompt.

If you specify the NULL VALUE NOT ALLOWED option without the REQUIRED option for a field that allows null values, then if you enter a null response, QUICK places a null value in the field. The NULL VALUE NOT ALLOWED option only prohibits you from entering null values directly. To prevent QUICK from supplying unintended null values, specify REQUIRED with NULL VALUE NOT ALLOWED.

The GENERATE Statement and Clusters

For screens with repeating PRIMARY, SECONDARY or DETAIL record-structures, QDESIGN automatically generates CLUSTER statements. Similarly, for items that are defined with multiple occurrences in the data dictionary, QDESIGN generates a CLUSTER statement for that item's FIELD statement if the file that contains the item doesn't itself occur.

Examples

The following example illustrates how the GENERATE statement automatically creates FIELD statements. In this example:

- Everything following the GENERATE statement is created by QDESIGN.
- QDESIGN automatically generates FIELD statements for all items in the record-structures accessed by the screen.
- REQUIRED NOCHANGE and LOOKUP NOTON are generated for segments in indexes for the PRIMARY record-structure.

```
> SCREEN PARTBASE
> FILE PARTS PRIMARY
>
> GENERATE
>
> FIELD PARTNUMBER OF PARTS REQUIRED NOCHANGE
> FIELD PARTVARIANT OF PARTS ID SAME
>  REQUIRED NOCHANGE  &
>  LOOKUP NOTON PARTS  &
>             VIA PARTNUMBER, PARTVARIANT  &
>             USING PARTNUMBER OF PARTS, &
>                   PARTVARIANT OF PARTS
> FIELD PARTNAME OF PARTS
> FIELD QOH OF PARTS
```

```
> FIELD UNITCOST OF PARTS
> FIELD UNITMARKUP OF PARTS
```

The following example illustrates the GENERATE statement with a DETAIL record-structure that occurs 5 times. In this example:

- GENERATE DETAIL causes QDESIGN to write all generated statements to QDESIGN's temporary save file.
- CLUSTER statements are generated for the fields in the repeating DETAIL file.

```
> SCREEN INVBASE
> FILE INVOICEMASTER PRIMARY
> FILE INVOICEDETAIL DETAIL OCCURS 5
>
> GENERATE DETAIL
>
> FIELD INVOICENUMBER OF INVOICEMASTER &
>   REQUIRED NOCHANGE  &
>   LOOKUP NOTON INVOICEMASTER
> FIELD ORDERNUMBER OF INVOICEMASTER REQUIRED NOCHANGE
> FIELD DATEYEAR OF INVOICEMASTER
> FIELD DATEMONTH OF INVOICEMASTER
> FIELD DATEDAY OF INVOICEMASTER
> CLUSTER OCCURS WITH INVOICEDETAIL
> FIELD PARTNUMBER OF INVOICEDETAIL REQUIRED NOCHANGE
> FIELD PARTVARIANT OF INVOICEDETAIL REQUIRED NOCHANGE
> FIELD QUANTITYSHIPPED OF INVOICEDETAIL
> CLUSTER
```

It's common practice to allow QDESIGN to generate a basic screen design for your application. You can save this design and modify it during a later QDESIGN session.

GO

Runs QUICK from QDESIGN.

Syntax

GO [**filespec**]

filespec

Names either a QUICK screen to execute or a QKGO file.

If the **procloc** program parameter was used when initiating QDESIGN, it is also used by QUICK to find the QUICK screen or the QKGO file.

If no screen or QKGO file is specified, the current screen design or the last screen compiled is processed.

Discussion

The GO statement allows you to run QUICK from within QDESIGN.

Entering a GO Statement before the BUILD Statement

The GO statement doesn't save the screen permanently. Entering the GO statement before a BUILD statement causes QDESIGN to construct procedures based on the design statements and store the compiled screen in a temporary file. In such cases, the DETAIL, LIST, NODETAIL, and NOLIST options are no longer valid on a BUILD statement. Once procedures are generated (by the GO statement), they are not regenerated by the BUILD statement.

Use the BUILD statement to save the compiled screen permanently.

HILITE

Assigns highlighting features to screen entities.

Syntax

HILITE object [highlight-option...] [,object [highlight-option...]]...

object

Specifies what is highlighted. The object can be one or more of the following:

Screen Object	Default	Notes
ACTIONBAR	Inverse Halftone	
ACTIONBARMARK	Off	
ALL		Highlights all objects. If used, no other screen object can be chosen.
BACKGROUND	Off	
DATA		Highlight to use when prompting or displaying fields (a combination of DISPLAY, EDIT, INPUT, and REQUEST).
DISPLAY	Off	Highlight to use when displaying fields.
EDIT		Highlight to use after the user enters an incorrect value in the field.
ERROR	Off	Highlights error messages in the message line.
FIELDMARK	Inverse Halftone	
FIELDPOPUPBORDER	Off	
HELP	Off	
HELPBORDER	Off	
ID	Off	
INFORMATION	Off	Highlights informational messages when displayed on the message line.
INPUT	Off	Highlight to use when the user is prompted in the field to enter data in Entry, Change, and Correct mode.
LABEL	Off	
LINEDRAWING	Off	
MENU	Off	
MENUBORDER	Off	
MENUKEYS	Underline	Highlight to use for the Menukey.
MENUMARK	Inverse Halftone	

Screen Object	Default	Notes
MESSAGE	Off	Highlights messages when displayed on the message line. MESSAGE states that all message types are to be highlighted.
MESSAGEBORDER	Off	
MODE		The DISPLAY highlight option in effect when BUILD is entered.
REQUEST	Inverse Halftone	Highlight to use when the user is prompted in an index segment field in Find mode or prompted in this field in Select mode when defining retrieval criteria.
SELECTBOX	Off	
SELECTBOXBORDER	Off	
SELECTBOXMARK	Inverse Halftone	
SEVERE	Off	Highlights severe messages when displayed on the message line.
TITLE	Off	
WARNING	Off	Highlights warning messages when displayed on the message line.

highlight-options

Specifies the highlighting options used for a given highlight-object. Multiple highlights can be used.

Limit: If either the DEFAULT or the OFF option is specified, none of the other highlighting options can be used.

highlight-options		
AUDIBLE	BLINKING	color
DEFAULT	HALFTONE	INVERSE
OFF	UNDERLINE	

AUDIBLE

Sounds the terminal bell (or its equivalent) when displaying messages of the associated type of error level (INFORMATION, WARNING, ERROR, SEVERE) or greater.

Limit: Applies only to messages.

BLINKING

Highlights the object with blinking.

Limit: Not supported on Windows.

color [ON color]

Highlights the specified object or group of objects in the specified color. You can specify any the following colors:

color options		
BLACK	BLUE	CYAN

color options		
GREEN	MAGENTA	RED
WHITE	YELLOW	

The first color applies to the highlighted screen object and the ON color applies to the background. **Windows**: The ON color option is not supported, as the background color is set by the Command Console window properties.

Limit: The background is fixed on HP color terminals and the ON option is ignored.

DEFAULT

Applies default highlighting to the object. For more information on the defaults, see the HILITE statement on (p. 147).

Limit: If the DEFAULT option is chosen, none of the other highlighting options can be used.

HALFTONE

Highlights the object with half intensity or alternative intensity, depending on the terminal type.

Limit: Not supported on Windows.

INVERSE

Highlights the object with inverse video. Inverse video reverses normal background and foreground settings.

OFF

Cancels highlighting.

Limit: If the OFF option is chosen, none of the other highlighting options can be used.

UNDERLINE

Highlights the object with an underline.

Limit: Not supported on Windows.

Discussion

The HILITE statement must come before the feature it affects.

The Action field highlighting defaults to the field highlighting options in effect in the screen design when the BUILD statement is entered.

Highlighting and Hardware Limitations

If you use a terminal that doesn't have a specific highlighting option available at execution-time, that highlighting option is ignored.

Highlighting and Color (Windows)

Although a Command Console window is capable of displaying sixteen colors, QDESIGN only recognizes eight colors and, as such, QUICK will only use eight colors. These are red, green, blue, cyan, yellow, magenta, white, and black.

QUICK on Windows ignores the underline, halftone, and blinking QDESIGN highlight options. Inverse highlighting is available.

Only the foreground color is controlled by QUICK even though QDESIGN supports the syntax for the background color. QUICK uses the background color set by the Console window properties.

Changing the background color when QUICK is executing will result in improper text coloring.

Example

The following example demonstrates the use of the HILITE statement to assign different highlighting features to objects on a screen:

```
> HILITE TITLE INVERSE HALFTONE
>
> DRAW 2,20 TO 5,60
>
> SKIP TO LINE 3
>
> TITLE "Future Industries" CENTERED
>
> HILITE  &
>   REQUEST INVERSE HALFTONE,  &
>   INPUT INVERSE,  &
>   DISPLAY INVERSE,  &
>   EDIT BLINKING
```

Statements such as the ones in the preceding example can be referenced in a USE statement for all screens in a system. This results in a similar look for all screens in the system without having to specify custom highlighting for each screen.

ITEM

Assigns values to items, or performs sums and balances on items.

Syntax

ITEM item [option]...

item

Specifies a record item or indicates a column in a relational table.

Limit: The ITEM statement isn't valid for temporary, defined, or predefined items.

Options

The ITEM options are BALANCE, FINAL, INITIAL, and SUM.

BALANCE [WITH] item2 [MESSAGE string|=string-expression|n]

Issues a warning message if the item named in the ITEM statement and items named in this option don't contain equal values when the record is updated. A warning is also issued if a record is displayed after retrieval in Find or Select mode and the user moves to the next record or returns from the screen without making the values equal.

MESSAGE string|=string-expression|n

Allows you to specify a warning message. If a number is specified, QUICK searches for a message in the designated designer message file, QKMSGDES (**MPE/iX**) or qkmsgdes.txt (**OpenVMS, UNIX, Windows**).

For more information, see Chapter 4, "Messages in PowerHouse", in the *PowerHouse Rules* book.

FINAL conditional-expression

Assigns a value to the named item. The value is calculated by this expression whenever the record or relational table is updated. The expression can contain values that are determined by screen processing.

If the GENERATE statement is used, no FIELD statement is generated for this item. If a FIELD statement is added by the designer, any value entered into the field is overwritten when the expression is evaluated.

The FINAL expression evaluation is performed during PUT verb processing if the NEED option of the FILE statement has been declared or if the data record status is one of the following:

- old, changed, undeleted
- old, unchanged, undeleted
- new, changed, undeleted

INITIAL conditional-expression [FIXED]

Assigns a value to the named item. The value is calculated by this expression when the data record is initialized. This value can be overridden by the QUICK screen user or designer. The INITIAL expression evaluation doesn't affect data record status.

FIXED

Assigns a value to the named item. The value is calculated by the specified conditional-expression when the data record is initialized and when the data record is updated. The FIXED option incorporates the attributes of the INITIAL and FINAL options.

If the GENERATE statement is used, no FIELD statement is generated for this item. If a FIELD statement is specified, the field is assumed to be display only.

OMIT

Indicates that the item is a read-only column and excludes it from relational database inserts and deletes, as well as any checksum calculations. The OMIT option can be used in cases where PowerHouse cannot determine that the column is read only. For example, ODBC may not return sufficient information to determine the read status of a column. Read-only columns are columns that are updated or controlled by the database such as computed columns or columns whose value is calculated by a stored procedure. While PowerHouse can use the values from such columns, it should not attempt to update them.

Including read-only columns in inserts or deletes can result in database errors. Including such columns in checksum calculations can result in update errors due to a checksum mismatch. When PowerHouse calculates the initial checksum on retrieval, it includes all columns not identified as read only. If the database changes the value of the column, its changed value will cause the checksum calculated on re-retrieval to be different than the one originally calculated.

The OMIT option must be the only option on the ITEM statement and the only option specified for the item. A warning is issued if there are any other ITEM statements for the item and the OMIT option takes precedence. This includes any generated automatic item initialization.

Limit: Ignored for non-relational items.

SUM [NEGATIVE] [INTO] item2 [WHEN POSITIVE|NEGATIVE] [,[NEGATIVE] [INTO] item3 [WHEN POSITIVE|NEGATIVE]]...

Adds all values entered in the item named in the ITEM statement and maintains the total in the item or items named in this option. This sum is automatically

- incremented when values are entered
- reduced when a data record containing the item is deleted
- adjusted when the value is changed

item2,item3

Names the item that hold the totals.

NEGATIVE

Reverses the incrementing and decrementing activities.

WHEN POSITIVE|NEGATIVE

Specifies that summing is to be performed only if the values are positive or only if the values are negative.

Limit: The maximum number of items that can be summed into is six.

Discussion

The ITEM statement is part of the data section of the screen design. It's used to assign values, to sum entered values into other items, and to balance values. To understand the difference between the way PowerHouse and relational systems handle null values, see Chapter 1, "PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book. The INITIAL and FINAL options override the corresponding portions of automatic initialization.

Multiple ITEM statements can be entered for a given item. All ITEM statements are processed in the sequence in which they occur in the screen design.

The ITEM statement must be declared in the screen where the record-structure is first declared. Any ITEM statements for record-structures passed down from a higher-level screen are ignored. The reverse is true if the passed record-structure has the MYVIEW option specified. In that case, only ITEM statements declared on the lower-level screen are recognized while on that screen.

Automatic Item Initialization

For screens that reference more than one record-structure, QUICK performs automatic initialization of items at execution-time based on item name and type matches. At compile-time, QDESIGN issues a message that indicates how items will be initialized at execution-time. The message that's issued is equivalent to a designer-specified ITEM statement.

QDESIGN considers the kind of file in which an item occurs when it looks for matches for item initialization. In addition to the file types that you can declare, there are two "composite" file types that QDESIGN handles in a special way. These file types are:

DETAIL-SECONDARY	a SECONDARY file that occurs with a DETAIL file
DETAIL-DELETE	a DELETE file that occurs with either a DETAIL-SECONDARY file or a DETAIL file

For a relational database, the syntax option references the column in a table or view. The BALANCE and SUM options may perform differently with a relational database (see the *PowerHouse and Relational Databases* book).

Item occurs in a file of type ...	**QDESIGN looks for matching items in ...**
PRIMARY	MASTER
DETAIL	PRIMARY, MASTER
DETAIL-SECONDARY	DETAIL, PRIMARY, MASTER
SECONDARY	PRIMARY, MASTER
DETAIL-DELETE	previously declared DETAIL-SECONDARY occurring with same DETAIL, DETAIL, PRIMARY, MASTER
DELETE	previously declared SECONDARY (not occurring with DETAIL), PRIMARY, MASTER
AUDIT related to DETAIL by WITH option	related DETAIL, previously declared DETAIL-SECONDARY occurring with related DETAIL, PRIMARY, MASTER
AUDIT related to DETAIL-SECONDARY	related SECONDARY, DETAIL, PRIMARY, MASTER

Item occurs in a file of type ...	QDESIGN looks for matching items in ...
AUDIT related to DETAIL-DELETE	DELETE, DETAIL-SECONDARY, DETAIL, PRIMARY, MASTER
AUDIT related to file other than DETAIL, DETAIL-SECONDARY, OR DETAIL-DELETE	related file, SECONDARY (excluding DETAIL-SECONDARY), PRIMARY, MASTER
AUDIT not related to another file	DETAIL-SECONDARY, DETAIL, SECONDARY, PRIMARY, MASTER

When applying the rules in the preceding table, QUICK follows several additional rules that eliminate potential ambiguity:

- Only previously-declared record-structures are considered for automatic item initialization.
- Automatic item initialization is established by matching item names and types. Items of type DATE can match items of type DATE or type NUMERIC. Otherwise, the match must be exact.
- As FIELD statements are generated or entered, QDESIGN checks for name and type matches. If a match is found for an item, QDESIGN generates an ITEM statement that indicates how the field for that item will be initialized at execution-time in QUICK.
- Once a name and type match for an item is found, no further matching is attempted.
- When searching for item name matches and more than one file of the same type has already been declared, QDESIGN searches from the most-recently to the least-recently declared file.
- Files are searched in the order that's imposed by the file type, according to the rules specified in the previous table. This is true regardless of the order in which files are declared.
- Using the OCCURS WITH option for an AUDIT file is equivalent to having declared both the OCCURS WITH and the WITH options. The WITH option establishes a relationship between files for automatic item initialization.
- QDESIGN doesn't attempt to initialize items in MASTER, REFERENCE, or DESIGNER files.
- For DELETE files, only segments are initialized.
- QDESIGN doesn't attempt to initialize items in files that are declared with the NOITEMS option.

Initializing Null Values

During item initialization in QUICK, relational items are initialized according to the following precedence:

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 exists in the dictionary that corresponds to this item which has an initial value, the item is initialized to the element initial value.
3. If neither of the above is the case, the item is initialized to null if null is allowed for the item, or it is initialized to default values (spaces for character items and zero for numeric).

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 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 (spaces for character items and zero for numeric and date).

Example

The following example sums the values of items and balances them with the values of other items. In this example:

- The ITEM statement sets TODAY to the system date when the QUICK screen user invokes the screen.
- The ITEM BATCHDEBITS statement generates a warning message if the value entered for BATCHDEBITS isn't equal to TOTALDEBITS.
- Negative values for DETAILAMOUNT are added to TOTALCREDITS; positive values for DETAILAMOUNT are added to TOTALDEBITS.

```
> SCREEN BATCH
>
> FILE BATCHHEADER
>
> ITEM TODAY INITIAL SYSDATE FIXED
>
> ITEM BATCHCREDITS BALANCE WITH TOTALCREDITS &
>  MESSAGE &
>  "Balance Error. Notify Accounting of batch number."
>
> ITEM BATCHDEBITS BALANCE WITH TOTALDEBITS MESSAGE &
>  "Balance Error. Notify Accounting of batch number."
>
> FILE BATCHDETAIL SECONDARY
> ITEM DETAILAMOUNT SUM INTO TOTALCREDITS &
>  WHEN NEGATIVE, &
>  INTO TOTALDEBITS WHEN POSITIVE
>
> FIELD BATCHCREDITS
> FIELD BATCHDEBITS
> FIELD TOTALCREDITS DISPLAY
> FIELD TOTALDEBITS DISPLAY
> FIELD DETAILAMOUNT
> FIELD TODAY NOID NOLABEL DATA AT 1,70 &
>  PREDISPLAY DISPLAY
>
> BUILD
```

KEY

Specifies a dynamic function key (DFK).

Syntax

MPE/iX:	**KEY n [LEVEL n] [LOCAL] [LABEL string[SCREEN]] [BLOCKTRANSFER|NOBLOCKTRANSFER] context-option conditional-command-list |CANCEL|DISABLE|NULL**
OpenVMS, UNIX, Windows:	**KEY n [LEVEL n] [LOCAL] [LABEL string [SCREEN]] context-option conditional-command-list |DISABLE|NULL**

n

Sets the number (n) of the DFK.

Limit: 32

LEVEL n

Identifies the shift level at which this function key definition is operative. The maximum number of levels is set in QKGO.

Limit: 8

Default: 1

LOCAL

Specifies that the DFK definition for this key is not inherited by the subscreen. The subscreen inherits the definition from the closest calling screen that does not have a LOCAL option attached to the key and context in question.

LABEL string [SCREEN]

Specifies the label string for the DFK. On terminal screens that support function-key labels, QUICK displays the specified label string for the DFK.

Limit: This option applies only to terminals that support function key labels.

Default: A blank

SCREEN

Specifies that the label string will be shared between action and data context. The label will remain the same when QUICK is prompting in the Action field or in a data field. This option eliminates context-sensitivity for the label it is assigned to.

BLOCKTRANSFER|NOBLOCKTRANSFER (MPE/iX)

Turns on the data transmitting feature of a DFK. For many actions, this is similar to pressing [Enter] before pressing a function key, with the exception that the AUTOUPDATE screen option is ignored.

NOBLOCKTRANSFER turns off the data transmitting feature of a DFK. Designers can use this to prevent specific DFKs, such as those for screen refresh, from transmitting data.

Limit: BLOCKTRANSFER and NOBLOCKTRANSFER are valid only in Block mode.

Default: NOBLOCKTRANSFER

context-option

Specifies in what context the DFK definition is operative. Depending on what context-option you choose, the DFK can execute only Action commands, Data commands, or Action and Data commands.

The context-options are ACTION, DATA, ACTION AND DATA, and DATA AND ACTION.

ACTION

Specifies that the DFK is usable only when QUICK is prompting in the Action field. In the conditional command list, you can specify Action commands and/or Action and Data commands, but not Data commands.

DATA

Specifies that the DFK is usable only when QUICK is prompting in a data field. In the conditional command list, you can specify Data commands and/or Action and Data commands to be executed in data fields. If these commands return the prompt to the Action field, then you can also specify Action commands and/or Action and Data commands to be executed in the Action field.

Limit: Keys with only DATA context are valid only in Field mode.

ACTION AND DATA
DATA AND ACTION

Specifies that the DFK is usable when QUICK is prompting in the Action field or in a data field. In the conditional command list, you can specify only Action and Data commands.

Limit: Any command options included as part of the DFK definition must be valid in both the Action and data fields.

conditional-command-list

Specifies what command(s) the DFK executes and, optionally, under what conditions. 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. The general form of a command list is:

command [, command]...

For a list of the available commands, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

condition

A condition is a logical test that has the general form:

[NOT] condition [AND|OR [NOT] condition]...

For more information about conditions or conditional command lists, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

CANCEL (MPE/iX)

Defines a DFK that a user can press to discard entries made to a screen. When this key is pressed once, data transmission is turned off for the next DFK. If you press this key twice, DFKs with the BLOCKTRANSFER option behave normally.

DISABLE

Specifies that there is no definition for the designated key on this screen, even if one was defined on the calling screen. This option is inherited by any subscreen in the same way that definitions and labels are inherited.

NULL

Initiates no direct action of its own.

Discussion

KEY statements must be in the data section of your screen design.

If you intend to use DFKs, you must use QKGO to create a suitable QKGO file for your application. If you do not, QUICK does not interpret or act upon the DFK definitions that you specify. However, there are three cases where QUICK does special processing that allows you to experiment with dynamic function keys even though you have not created a QKGO file for them:

- If you run QUICK from QDESIGN with the GO statement and there is no QKGO file, QUICK recognizes and processes KEY statements.
- If a screen name rather than a QKGO file name is included in the **auto** program parameter when QUICK is started, QUICK recognizes and processes KEY statements.
- If the environment variable for QKGO points to a screen file, QUICK recognizes and processes KEY statements.

In all three cases, QUICK assumes that it has a QKGO file with following parameter settings:

Function Key Support Mode, Dynamic

Function keys, 8

Shift Levels, 2

Labels Active, N

Bank Labels, N

Lock Function Keys, N

OpenVMS: By default, logical function key 1 maps to the physical function key GOLD_1, logical function key 2 maps to the physical function key GOLD_2, and so on through to GOLD_8. You can change these default mappings by using the Terminal Interface Configuration screen in QKGO.

The following table illustrates, with examples, the rules that QUICK uses to determine the preshift level. The table assumes that all functions used are defined at all eight shift levels.

This table shows the effects of the different shifting options. How the commands were entered (for example, as part of a key or by other means, as discussed later) is not important for this example.

Initial Level	Command Sequence[1]	Effect	Command Executed	Final Level
1	SHIFT	go to level 2		
	HELP	execute	level -2 HELP	1
1	SHIFT	go to level 2		
	SHIFTLOCK	go to level 3		3
	SHIFT	go to level 4		
	HELP	execute	level-4 HELP	
1	KEYSHIFT 6	go to level 6		6
	SHIFT	go to level 7		
	SHIFT	go to level 8		
	HELP	execute	level-8 HELP	

Initial Level	Command Sequence[1]	Effect	Command Executed	Final Level
1	SHIFT	go to level 2		
	KEYSHIFT 6	go to level 6		6
	SHIFT	go to level 7		
	HELP	execute	level-7 HELP	
1	KEYSHIFT 6	go to level 6		
	SHIFT	go to level 7		
	SHIFT	go to level 8		
	SHIFTLOCK	go to level 1		1
	SHIFT	go to level 2		
	HELP	execute	level-2 HELP	

[1]*The keyname in the command sequence (for example: SHIFT, HELP, SHIFTLOCK) is a user-defined label.*

Inheritance Rules

In a system of screens, a subscreen can inherit DFK definitions from calling screens. For example, where screen A calls screen B which calls screen C, screen C can inherit key definitions from B and any from A, if they were passed down to B and not changed at B. The B screen can inherit from A, but not C.

Inheritance rules also apply to key definitions described in QKGO. For more information, see (p. 255).

The QKGO file acts as the initial calling screen. The QKGO file passes its definitions down to the first screen in a QUICK screen hierarchy.

The inheritance rule for key definitions is as follows: if a calling screen (at A) has a definition for a given key and a subscreen (at B) does not have a definition for that key with the same shift level and context, the subscreen inherits the definition from the calling screen.

Suppose that a calling screen has these key definitions:

```
> KEY 1 LABEL "Up one keylevel"  &
>             ACTION AND DATA SHIFT
> KEY 2 LABEL "Extended help"  &
>             ACTION AND DATA &
>             EXTENDED HELP
> KEY 3 LABEL "Help" &
>             ACTION AND DATA HELP
> KEY 4 LABEL "Refresh screen"  &
>             ACTION AND DATA REFRESH
> KEY 5 LABEL "Backup 1 field" &
>             DATA BACKUP
> KEY 1 LEVEL 2 LABEL "Down one keylevel" &
>             ACTION AND DATA SHIFT TO 1
> KEY 6 LEVEL 2 LABEL "Return to stop"  &
>             ACTION &
>             RETURN TO STOP
```

and that the subscreen has only these definitions:

```
> KEY 4 LABEL "Update stay" ACTION UPDATE STAY
> KEY 5 LABEL "Update return" ACTION UPDATE RETURN
```

When the subscreen is invoked, its own level-1 definitions for keys F4 and F5 are in effect. In addition, QUICK passes down the calling screen's level-1 definitions of keys F1, F2, and F3. When the subscreen first appears, QUICK displays the following label information:

Up one keylevel	Extended help	Help	Update stay	Update return	....	....	....

QUICK also passes down the level-2 definitions for F1 and F6. When the user presses F1, QUICK shifts to level 2 and displays the following label information:

Down one keylevel	....	....	....	....	Return to stop	....	....

The level-2 functions are inherited from the calling screen.

When QUICK returns from the subscreen to a calling screen, it restores the DFK definitions that were in effect on the calling screen when the user invoked the subscreen. QUICK does not pass definitions from a subscreen to the calling screen.

Note that named DESIGNER procedures included in one screen design are not accessible from other screens. Therefore, if a screen inherits a DFK definition that uses the DESIGNER command option, QUICK treats the action in the inherited definition as invalid if it tries to execute it. This is true even if there is an identically-named DESIGNER procedure defined on the subscreen. In order to execute the DESIGNER procedure, the DFK definition must be defined on the same screen.

To invoke the same DESIGNER procedure from both a calling screen and a subscreen, include the applicable KEY statement and the DESIGNER procedure in both screen specifications.

Overriding Inheritance Rules

In some instances, you may wish to create screens that do not pass function key definitions or labels to a subscreen. To override the passing of a current DFK definition to subscreens, use the LOCAL option of the KEY statement, as in

```
> KEY 1 LOCAL ACTION DESIGNER DOIT
```

The default DFK definition of a subscreen is the DFK definition on the closest calling screen that does not employ the LOCAL option.

If you do not want a subscreen to inherit a definition from any calling screen, use the DISABLE option on that subscreen. QUICK then treats the key/level/context on this subscreen as if no definition had been created. The subscreen of a calling screen that employs a disabled DFK inherits the DISABLE key definition. In other words, a DISABLE is inherited just like any other DFK definition.

If you want to disable a DFK definition on a current screen and all subscreens (by default), use the DISABLE option, as in

```
> KEY 1 ACTION AND DATA DISABLE
```

If you want to disable a DFK on the current screen, but allow a calling screen definition to be passed down to subscreens below the current screen, use the LOCAL option with DISABLE to disable only on the current screen, as in

```
> KEY 1 LOCAL ACTION AND DATA DISABLE
```

The Importance of Consistency

The definitions set for DFKs can cause the meanings of the keys to change as a user moves from screen to screen. This can cause problems for users if the meanings fluctuate randomly from screen to screen. Ideally, definitions should follow a pattern. For example, if extended help is available on several screens, the same DFK should invoke it on each screen.

MPE/iX: In Block mode, DFKs with BLOCKTRANSFER are designed to transmit data, while others (those with NOBLOCKTRANSFER) are not. To help users keep track of which DFKs transmit data and which don't, designers should set up easy-to-follow conventions. For example, key labels in uppercase could imply BLOCKTRANSFER and those in lowercase could imply NOBLOCKTRANSFER. You can also group the two different types of function keys. For example, F1 to F4 could be BLOCKTRANSFER keys, while the remainder could be NOBLOCKTRANSFER keys. Grouping of similar function keys becomes more important on terminals that do not support screen labels.

Banked and Unbanked Labels

QUICK can display the label strings that you specify in either banked or unbanked format. On HP terminals that support function key labels, the label for each DFK is two lines deep and eight characters wide.

In banked format, label strings are displayed in pairs, one on each line of the function key label. The top line of the function key label refers to the Action field context, and the bottom line refers to the Data field context.

With banked labels, each label string occupies a single line, and can therefore be a maximum of eight characters in length. With unbanked labels, the label string occupies both lines of a function key label, and can, therefore, be a maximum of 16 characters in length.

By default, QUICK uses unbanked labels. To override this default, change the **bank labels** parameter in the QKGO file. In unbanked format, the function key label displays a single label string which serves to document a single context. This means that QUICK cannot display the label strings for the Action context and the Data context simultaneously.

These statements

```
> KEY 1 LABEL "Update" ACTION UPDATE
> KEY 1 LABEL "Skip all" DATA SKIP ALL
```

produce these labels:

```
Update
Skip all
```

where Update applies to the Action field, and Skip All applies to the data fields.

If QUICK uses unbanked labels (the default), it must redisplay the function key labels when the context changes in order to provide label information that is context-specific. If QUICK uses banked labels, the Action field label strings and the Data field label strings are displayed simultaneously. QUICK doesn't have to redisplay the function key labels when the context changes.

Each type of label has inherent advantages and disadvantages. Unbanked labels can be longer and more informative than banked labels. However, they require QUICK to do extra screen I/O to redisplay the function key labels when the context changes. Banked labels do not require the extra I/O, but are less informative.

QDESIGN does not check the length of the specified label string. If the label string is shorter than the allowable maximum length, QUICK pads it with blanks for display. If the label string is longer than the maximum, QUICK truncates it.

The following example illustrates how QUICK handles label strings. If the QKGO file stipulates unbanked labels, then these statements

```
> KEY 4 LABEL "Information" ACTION INFORMATION
> KEY 4 LABEL "Extendedhelp" DATA EXTENDED HELP
> KEY 5 LABEL "Return to stop" ACTION RETURN TO STOP
> KEY 6 LABEL "Skip cluster" SCREEN DATA SKIP CLUSTER
```

cause the following function key labels when QUICK is prompting in the Action field (or, for **MPE/iX**, in BLOCKMODE):

.....	.....	.....	Informa-	Return to	Skip	.....	.....
.....	.....	.....	tion	stop	cluster	.....	.....

In Field mode, when QUICK begins prompting in the data fields, it redisplays the function key labels with information that suits the context:

.....	.....	.....	Extended help	.....	Skip cluster	.....	.....
.....	.....	.....		.....		.....	.....

The labels for keys F4 and F5 vary by context. When QUICK is prompting in a data field in Field mode, the label for F5 is blank. The blank indicates that F5 has no effect in that context.

The label for F6 does not vary. QUICK displays the label string in both contexts, even though the key has no effect in the Action field context. This occurs because the screen designer used the SCREEN option of the KEY statement, which forces QUICK to display the label string regardless of the context. If you use the SCREEN option when you specify a label string for a given key-level combination, QDESIGN does not allow you to specify a second label string for that key-level combination.

If you change the QKGO file to stipulate banked labels, then, given the same screen specification as in the previous example, QUICK displays these function key labels:

.....	.....	.....	Informat	Return	Skip	.....	.....
.....	.....	.....	Extended	.....	Skip	.....	.....

These labels are intended solely to illustrate how QUICK truncates label strings, and are not presented as models of good form. When using banked labels, QUICK uses only the first eight characters of each label string, so you can achieve greater clarity by using abbreviations.

Highlighting Labels

You can use the KEYLABEL option of the HILITE statement to specify highlighting options for the function key labels, as in

```
> HILITE KEYLABEL UNDERLINE
```

Place HILITE statements before the KEY statements to which the highlighting applies.

Example

In the following example, five custom keys are defined. The F1 key is defined as a Help key. The other keys are defined to run DESIGNER procedures that in turn run subscreens.

```
> SCREEN STCKMNT &
> MODE AT 1,70 &
> MESSAGE ON LINE 23 &
> ACTIONBAR ON LINE 3
>
> ACTIONMENU LABEL "Next"   ACTION NEXT DATA
> ACTIONMENU LABEL "Delete" ACTION DELETE
> ACTIONMENU LABEL "Entry"  ACTION ENTRY
> ACTIONMENU LABEL "Find"   ACTION FIND
> ACTIONMENU LABEL "Select" ACTION SELECT
> ACTIONMENU LABEL "Update" ACTION UPDATE
>
> KEY 1 LABEL "  HELP  " SCREEN ACTION AND DATA &
>   HELP
> KEY 2 LABEL "EmployeeMaint." SCREEN ACTION &
>   DESIGNER EMPS
> KEY 3 LABEL "Parts   Maint. " SCREEN ACTION &
>   DESIGNER PART
> KEY 4 LABEL "ContractMaint." SCREEN ACTION &
>   DESIGNER CONT
> KEY 5 LABEL "Reports" SCREEN ACTION &
>   DESIGNER REPO
>
> FILE STOCKS
> ALIGN (20,25,40)
```

```
> HILITE TITLE LINEDRAWING INVERSE
> DRAW 5,16 TO 8,64
> TITLE "  F u t u r e   I n d u s t r i e s  Inc. " &
>  At 6,17
> TITLE "         Stock File Maintenance           "  &
>  At 7,17
> SKIP 3
> HILITE TITLE LINEDRAWING DEFAULT
> FIELD STOCKNUM OF STOCKS REQUIRED NOCHANGE &
>  LOOKUP NOTON STOCKS
> FIELD EMPLOYEE OF STOCKS
> FIELD STOCKCOUNT OF STOCKS
> FIELD DISCPCT OF STOCKS
> FIELD STOCKCOST OF STOCKS
>
> PROCEDURE DESIGNER EMPS
> RUN SCREEN EMPMNT
>
> PROCEDURE DESIGNER PART
> RUN SCREEN PARTMNT
>
> PROCEDURE DESIGNER CONT
> RUN SCREEN CONTRACT
>
> PROCEDURE DESIGNER REPO
> RUN SCREEN REPRTMNU
>
> BUILD
```

MENUITEM

Specifies the action taken by a drop-down menu item.

Syntax

MENUITEM [LABEL string] ACTION conditional-command-list
[{MENUKEY char}|NOMENUKEY]

ACTION conditional-command-list

Specifies what command(s) the drop-down menu item executes and, optionally, under what conditions. 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. The general form of a command list is:

command [, command]...

For a list of the available commands, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

condition

A condition is a logical test that has the general form:

[NOT] condition [AND|OR [NOT] condition]...

For more information about conditions or conditional command lists, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

LABEL string

Displays a specified string on a pull-down Action bar menu.

Default: If no LABEL is specified, QUICK uses the value of the action_command.

{MENUKEY char}|NOMENUKEY

MENUKEY assigns a short-cut menu key that users can press to select a drop-down menu item when they're in a drop-down menu. MENUKEY overrides default menu keys applied by the MENUKEY option of the SET statement.

The menu key character should be unique among all the items in the drop-down menu. If the menu key character is not a character in the item label, then it is displayed in brackets at the end of the label.

Limit: The character can be an uppercase letter, a lowercase letter, or a number, but not a special character.

NOMENUKEY ensures that no menu key is assigned to a drop-down menu item, even if the MENUKEY option of the SET statement is used.

Menu keys are highlighted with an underline by default. You can change the highlighting by using the HILITE statement.

For more information about adding and customizing menu keys for your menu-driven QUICK screens, see (p. 70), (p. 147), and (p. 199).

Discussion

The MENUITEM statement, together with the ACTIONMENU statement and the ACTIONBAR option of the SCREEN statement, allow you to define an Action bar.

Action Bars

An Action bar presents a list of QUICK actions or menus in a menu bar that extends across the terminal window. You can invoke an Action command by selecting the appropriate entry in the Action bar or pull down menu as an alternative to entering the action at the Action field prompt.

With an Action bar, you can associate commands with a more descriptive label, and also show all the commands that are available to the user.

Adding an Action Bar to your Screen Design

To create an Action bar:

1. Specify the ACTIONBAR option of the SCREEN statement.
2. Use the ACTIONMENU statement to define the menus and actions you want to place on the Action bar.
3. Use the MENUITEM statement to define the actions for each menu to be pulled down from the Action bar.

Action Bar Menus and Actions

The Action bar can contain both menus and actions. To specify a menu, enter the ACTIONMENU statement without indicating an action option, as in

```
> ACTIONMENU LABEL "EMPLOYEES"
```

To specify the action that's performed, use the ACTION option, as in

```
> ACTIONMENU LABEL "CREATE" ACTION ENTRY
```

You can specify any QUICK Action command as an ACTION option. If the action requires an ID-number or ID-number range as a parameter, you can specify these explicitly or you can obtain the ID-numbers at run time by using the MARK or PROMPT options.

If you specify PROMPT, QUICK prompts you for ID-number values with a prompt box. If you specify MARK, QUICK determines the value from the current FIELDMARK setting. If there is no current MARK and MARK has been specified, QUICK prompts for the ID-numbers.

If you don't want your Action bar to have pull-down menus, exclude MENUITEM statements from your design statements. Instead, you can specify Action commands in the Action bar.

But if you want to specify a pull-down menu, enter MENUITEM statements after the corresponding ACTIONMENU statement, as in

```
> ACTIONMENU LABEL "EMPLOYEES"
> MENUITEM LABEL "LOCATE" ACTION FIND
> MENUITEM LABEL "NEW" ACTION ENTRY
```

If you specify MENUITEM statements following the ACTIONMENU statement, QDESIGN constructs a menu that appears to "pull down" from the Action bar when the Action menu is selected.

Each MENUITEM statement creates a new entry on the menu. The menu can accommodate any size label and any number of items: the menu border adjusts to accommodate the largest menu label, and the menu scrolls to fit the number of menu items.

Generally, menus pull down from the Action bar; that is, they appear below the corresponding Action menu on the Action bar. However, if you specify that the Action bar appear on the last three lines of the terminal, the menus pop up above the Action bar.

Action Bars and Action Fields

You can specify that both an Action bar and an Action field appear on a screen. When the screen runs, only one mode is active at a time, but you can switch between the two modes. This way, you can make a limited set of commands available through an Action bar for the novice user but still enable an Action field for the more experienced user.

If both the Action bar and Action field are available, the screen initially runs Action field prompting mode by default. To specify that the Action bar should be the initial mode, add the STARTUP option to the ACTIONBAR option of the SCREEN statement.

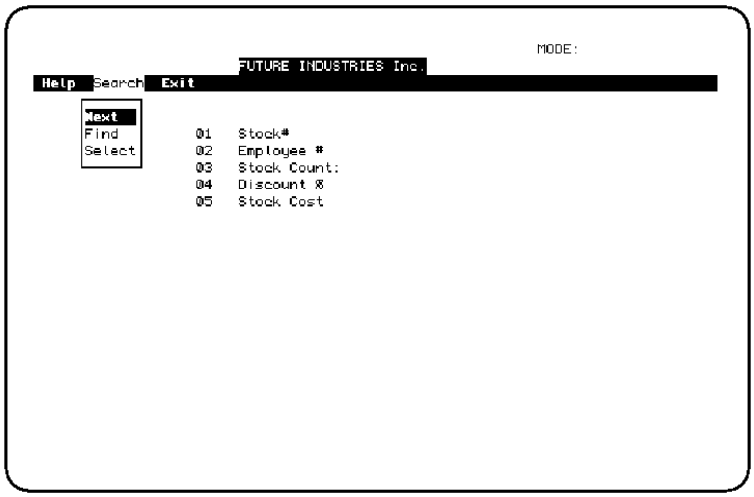
If the Action field and Action bar appear on the same line, only the current prompting mode is visible. If they are not on the same line, both the Action bar and Action field are visible, but QUICK only prompts you at the active mode.

Example

The following example is an Actionbar with pulldown menus and the MODE field moved to line 23, column 60 on the PARTINFO screen. The ACTIONMENU statements, followed by the MENUITEM statements, specify pulldown menus for the label "Search". Each MENUITEM defines a label and action for one item in the pulldown menu.

```
> SCREEN STOKINFO NOACTION MODE AT 1,60 &
>  ACTIONBAR ON LINE 3
>
> ACTIONMENU LABEL "Help" ACTION EXTENDED HELP
>
> ACTIONMENU LABEL "Search"
> MENUITEM LABEL "Next" ACTION NEXT DATA
> MENUITEM LABEL "Find" ACTION FIND
> MENUITEM LABEL "Select" ACTION SELECT
>
> ACTIONMENU LABEL "Exit" ACTION RETURN
```

The following screen shows the pulldown menu for the "Search" ACTIONMENU item:

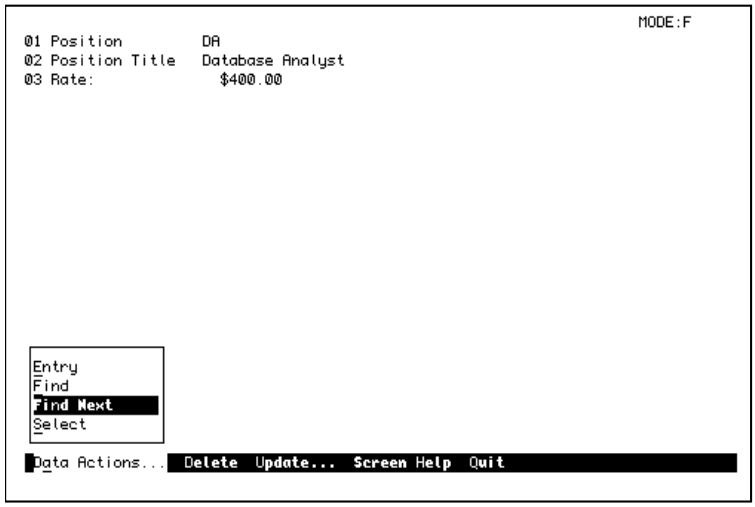


The statements in the following example create an Action bar and pull-down menus complete with menu keys for each ACTIONMENU item and MENUITEM item. When you select an option and enter the Accept command, or press a menu key, QUICK executes the associated action. The warning message means that since you've positioned the Action bar at the bottom of the screen, and there's no room for drop-down menu items to drop down, they'll drop "up" instead.

```
> SCREEN MAINTSCR NOACTION MODE AT 1,70  &
>    FIELDMARK     &
>    MESSAGE ON LINE 23  &
>    ACTIONBAR ON LINE 24
*W* Menus will appear on screen lines preceding ACTIONBAR line.
> ACTIONMENU LABEL "Data Actions..."  &
>       MENUKEY "A"
>    MENUITEM LABEL "Entry" ACTION ENTRY &
>       MENUKEY "E"
>    MENUITEM LABEL "Find" ACTION FIND &
>       MENUKEY "F"
>    MENUITEM LABEL "Find Next" ACTION NEXT DATA &
>       MENUKEY "N"
>    MENUITEM LABEL "Select" ACTION SELECT &
>       MENUKEY "S"
> ACTIONMENU LABEL "Delete" ACTION DELETE &
>       MENUKEY "D"
```

```
> ACTIONMENU LABEL "Update..." &
>       MENUKEY "U"
>    MENUITEM LABEL "Update" ACTION UPDATE &
>       MENUKEY "U"
>    MENUITEM LABEL "Update Stay" ACTION UPDATE STAY &
>       MENUKEY "S"
>    MENUITEM LABEL "Update Return" ACTION UPDATE RETURN &
>       MENUKEY "R"
>    MENUITEM LABEL "Update Next" ACTION UPDATE NEXT  &
>       MENUKEY "N"
> ACTIONMENU LABEL "Screen Help" ACTION EXTENDED HELP  &
>       MENUKEY "H"
> ACTIONMENU LABEL "Quit" ACTION RETURN   &
>       MENUKEY "Q"
.
.
.
```

The resulting screen looks like this:



QSHOW

Runs QSHOW from QDESIGN.

Syntax

QSHOW

The QSHOW statements are as follows:

Statement	Purpose
EXIT	Terminates QSHOW.
GENERATE	Generates PDL source statements.
QUIT	Terminates QSHOW.
SAVE	Saves QSHOW source statements in qshosave.
SET	Overrides default options.
SHOW	Reports dictionary definitions.
USE	Processes QSHOW statements stored in a source statement file as though they'd been entered from the keyboard.

Discussion

The QSHOW statement initiates a QSHOW session from within QDESIGN. With QSHOW you can make fast online inquiries about entities (such as elements, files, and record-structures) in the data dictionary.

When you exit from QSHOW, your QDESIGN session resumes at the point at which it was interrupted.

See Chapter 4, "QSHOW Statements" in the *PDL and Utilities Reference* book.

query-specification(SELECT)

Defines a collection of rows that will be accessible when the cursor is opened.

Syntax

SELECT [ALL|DISTINCT] {*|project-list}
FROM tablespec [,tablespec]...
[WHERE sql-condition]
[GROUP BY columnspec [,columnspec]...]
[HAVING sql-condition]

The syntax for a subquery is the same as for a query-specification with two exceptions: the subquery must project a single-column table and the syntax of the subquery includes enclosing brackets.

ALL|DISTINCT

ALL indicates that duplicate rows are included. DISTINCT indicates that duplicate rows are eliminated.

Default: ALL

*

Selects all the columns from the specified tables.

project-list

A columnspec or derived column, or a list of columnspecs and derived columns separated by commas. If names are ambiguous, they must be qualified to ensure they can be identified uniquely. DBKEY may be included in the project-list if it is supported by the underlying database system.

The syntax for a columnspec is:

[table-name.|correlation-name.]column-name

The syntax for a derived column is:

expression [AS name]

The AS option assigns an alias to a column. It can be used to uniquely identify multiple references to the same column, or to give a name to a derived column so it can be referenced in a program. Using the AS option automatically applies the DISPLAY option to the FIELD statement.

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

FROM tablespec [,tablespec]...

The FROM option identifies the tables where data in the project-list is retrieved. Rows can be retrieved from simple tables, derived tables, or joined tables. A derived table is a full query-specification including an optional ORDER BY option.

Joins are used to combine data from two or more tables based on the relationships between data in those tables. The type of join affects the rows retrieved by the query-specification.

If a correlation name is defined for the tablespec, subsequent PowerHouse references to the table must use the correlation name.

The general form of the tablespec syntax used in the following options is:

[owner.]table-name [correlation-name]

In addition to the general form of the tablespec, the following forms are also valid for this option:

(derived table) correlation-name

The correlation name must be defined for a derived table, and subsequent PowerHouse references to the derived table must use the correlation name.

tablespec CROSS JOIN tablespec

In a cross join, every possible combination of rows from the two tables being joined is created, without regard for any matching.

tablespec [INNER] JOIN tablespec ON columnspec = columnspec [AND columnspec = columnspec]...

In an inner join, a row is included in the result-set only if it has a matching row in the other table. The INNER keyword is for documentation only.

The ON option specifies the condition of the join.

tablespec LEFT|RIGHT|FULL [OUTER] JOIN tablespec ON columnspec = columnspec [AND columnspec = columnspec]...

An outer join includes all rows in the tables whether or not there are matching rows.

The left outer join returns rows from the table listed before the JOIN keyword, even if they don't have a matching row in the second table listed.

The right outer join returns rows from the table listed after the JOIN keyword, even if they don't have a matching row in the first table listed.

The full outer join returns rows from both tables listed, even if they don't have a matching row in the other table listed.

The ON option specifies the condition of the join.

WHERE sql-condition

The sql-condition defines linkage between tables in the query, and search criteria for rows to be retrieved. Only data which meets the criteria is available for use by PowerHouse.

The sql-condition is a condition which is limited for use within Cognos SQL syntax. For more information about SQL conditions, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book or refer to an SQL reference manual.

GROUP BY columnspec [,columnspec]...

This option rearranges the result-set into the minimum number of groups such that all rows within any one group have the same value for the GROUP BY columns. Rows that do not satisfy the WHERE option are eliminated before any grouping is done. The result is known as a grouped table.

To use the GROUP BY option:

- the grouping columns need not appear in the project-list
- any non-aggregate in the project-list must be used in the GROUP BY option

```
> SQL DECLARE X CURSOR FOR &
>   SELECT SP.PNO, MAX(SP.QTY), MIN(SP.QTY) &
>   FROM SP &
>   WHERE SP.SNO <> 'S1' &
>   GROUP BY SP.PNO
```

For detailed information about the GROUP BY option, refer to an SQL reference manual.

HAVING sql-condition

The HAVING option is used to eliminate groups, just as the WHERE option is used to eliminate rows. The sql-condition is a condition which is limited to use within Cognos SQL syntax. For more information about SQL conditions, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book or refer to an SQL reference manual.

```
> SQL DECLARE X CURSOR FOR &
>  SELECT SP.PNO &
>      FROM SP &
>      GROUP BY SP.PNO &
>      HAVING COUNT(*) > 1
```

Limit: The sql-condition must evaluate to a single value per group.

Discussion

Specifying Selection Criteria Using WHERE Option and Substitutions

The WHERE option is used to define selection criteria for rows to be retrieved. Only data which meets the criteria is available for use by PowerHouse. In addition to specifying selection criteria within the query-specification, you can specify selection criteria on the following statements:

- ACCESS
- CURSOR
- LOOKUP option on the FIELD statement
- [SQL] OPEN verb

Examples

The following example demonstrates the way PowerHouse creates a single SQL query combining multiple conditions from multiple statements:

```
> SET LIST SQL
> SQL IN EMPBASE DECLARE X CURSOR FOR &
>   SELECT * FROM EMPLOYEES &
>   WHERE CITY = 'BOSTON'
> SCREEN EMPLOYC
> CURSOR X WHERE(EMPLOYEE BETWEEN 1000 AND 5000) &
>   PRIMARY KEY EMPLOYEE
> ACCESS WHERE(POSITION = 'PRG') &
>   VIA EMPLOYEE
```

The final query includes all three conditions specified in the WHERE options.

```
__ Sql after substitutions are applied:
__ SELECT *
__    FROM EMPLOYEES
__    WHERE POSITION = 'PRG' and
__          EMPLOYEE BETWEEN 1000 AND
__          5000 and
__          CITY = 'BOSTON'
```

Inner Joins

The following inner join would report all customers with matching invoices:

```
> SQL DECLARE X CURSOR FOR &
>   SELECT * FROM CUSTOMER C &
>   INNER JOIN INVOICES I &
>   ON C.CUSTOMER_NUM=I.CUSTOMER_NUM
```

Outer Joins

The following example would report all customers even if they didn't have any invoices. Invoices without matching customers would not be reported.

```
> SQL DECLARE X CURSOR FOR &
>   SELECT * FROM CUSTOMER C &
>   LEFT OUTER JOIN INVOICES I &
>   ON C.CUSTOMER_NUM=I.CUSTOMER_NUM
```

The next example would report all invoices even if they didn't have any matching customer information. Customers without invoices would not be reported.

```
> SQL DECLARE X CURSOR FOR &
>   SELECT * FROM CUSTOMER C &
>   RIGHT OUTER JOIN INVOICES I &
>   ON I.CUSTOMER_NUM=C.CUSTOMER_NUM
```

QUIT

Terminates QDESIGN.

Syntax

QUIT

Discussion

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

You can also use the EXIT statement to end your QDESIGN session and return control to the operating system or invoking program.

REPORT

Executes QUIZ.

Syntax

REPORT filespec [options...]

filespec

Specifies the name of an existing compiled QUIZ report file.

Options

REPORT Options		
AUTO	CLEAR	HIDDEN
ID|NOID	[ENTRY] IF	INPUT B|C|SAME
LABEL|NOLABEL	MARK|NOMARK	NOCONSOLE
NOWARN	ON ERROR	REFRESH
RESPONSE	WAIT|NOWAIT	

AUTO

Invokes QUIZ automatically when the standard entry sequence reaches this statement, if the ENTRY procedure was generated by QDESIGN.

CLEAR option

Clears an area of the terminal memory before invoking QUIZ. Any output to the terminal from the report appears starting on the first line of the cleared area. Lines cleared are refreshed automatically when the screen is reactivated and QUICK is ready to prompt the user. If multiple COMMAND, REPORT, or RUN statements are combined with ID SAME, the area cleared is refreshed after the last statement in the chain has completed execution and QUICK is ready to prompt the user.

ALL

Clears the entire terminal memory.

SCREEN

Clears the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears the area between and including lines n to m, numbering from the first line of terminal memory. LINES n by itself clears line n only.

HIDDEN

Suppresses the screen ID display, but lets the user reference a field by ID-number in MARK mode.

ID n [AT [line],column]
ID NEXT [AT [line],column]
ID SAME
NOID

Control how and where QUICK screen field ID-numbers are assigned.

ID n

Explicitly specifies an ID-number.

Limit: 1 to 99

ID NEXT

Uses the next ID-number in sequence.

AT [line],column

Positions the first digit of the ID-number at the specified line and column relative to the starting line of the screen. If the line is missing, the current line is assumed.

ID SAME

Instructs QDESIGN to omit the ID-number on the report. To execute the report, use the ID-number of the previous field.

NOID

States that no ID-number is assigned to this report and the report can't be referenced from the Action field.

[ENTRY] IF condition

Invokes the specified report in the standard entry sequence only if this condition is satisfied. QDESIGN generates an identical IF condition in the default ENTRY procedure.

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

Limit: The IF option is evaluated only during the standard entry sequence; otherwise, it is ignored.

Default: If the condition is satisfied, AUTO is assumed.

INPUT B|C|SAME (MPE/iX)

Puts the terminal in the specified input mode prior to executing QUIZ. The terminal is put back into the original mode after completion of the command.

Default: The mode it was in before QUICK was invoked.

B

Puts the terminal in Block mode. This should only be used for commands that must run in Block mode.

C

Puts the terminal in Character mode.

SAME

Leaves the terminal in the current input mode. If the screen has Block mode capability, this option should only be used for commands that do not write to the terminal.

LABEL [string] [AT [line],column] | NOLABEL

Declares the label and its position.

[string] [AT [line],column]

Indicates the report label and, optionally, the position of the label on the screen.

The AT option positions the first character of the label at the specified line and column relative to the starting line of the screen. If the line isn't specified, the current line is assumed.

Default: The item name or the first word in the COMMAND string.

NOLABEL

Specifies that no label is to appear for the report.

MARK|NOMARK

MARK enables fieldmarking for a report with an ID-number.

NOMARK disables the default fieldmarking for a report with an ID-number when fieldmarking is enabled.

NOCONSOLE (Windows)

Suppresses opening a Command Console window. Normally QUICK opens a second Command Console window to run QUIZ. If QUIZ runs in the background, does not require user input, or does not display useful output, the second command console window may not be necessary.

NOWARN

Specifies that operating system warning messages issued during the execution of the report should not be displayed.

ON ERROR CONTINUE|TERMINATE

Specifies the action to be taken if an operating system error occurs during the execution of a report. If TERMINATE is in effect, an operating system error causes QUICK to process the error as it would for an ERROR verb. If CONTINUE is specified, an operating system error is ignored and processing continues as if the error had not occurred.

For information on the ERROR verb, see (p. 420).

Default: TERMINATE

REFRESH option

Clears and rewrites an area of the terminal memory when the screen is reactivated and QUICK is ready to prompt the user. REFRESH options are performed before, and in addition to, an automatic refresh from any CLEAR option.

ALL

Clears and rewrites the entire terminal memory.

SCREEN

Clears and rewrites the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears and rewrites the area between and including lines n to m, numbering from the first line of terminal memory. LINES n alone refreshes line n only.

RESPONSE

Prompts the QUICK user for a response after the report has executed. QUICK resumes processing after the user responds, preventing the screen from being refreshed immediately.

WAIT|NOWAIT (Windows)

The WAIT option instructs QUICK to suspend current screen processing until the report has executed, at which time control returns to the screen. The NOWAIT option specifies that screen processing continues immediately and the report executes concurrently.

Default: NOWAIT

Discussion

The REPORT statement executes QUIZ and takes the compiled report as a parameter.

REVISE

Invokes an editor to edit files from within QDESIGN.

Syntax

REVISE [*|filespec [DETAIL|NODETAIL]
[LIST|NOLIST] [USE|NOUSE]]

*****

Signifies that the temporary source statement save file is to be edited.The save file is QKSAVE (**MPE/iX**) or qksave.qks (**OpenVMS, UNIX, Windows**).

All statements you've entered since the last CANCEL CLEAR, SAVE CLEAR, or SET SAVE CLEAR statement are recorded in the save file. However, the CANCEL CLEAR, SAVE CLEAR, SET SAVE CLEAR, SAVE, and EXIT statements are not recorded.

If you specify options without specifying the name of a file, you must use the asterisk. The asterisk is not required if you are editing qksave without changing any default options.

filespec

Specifies the name of an existing permanent file. If this file does not contain component statements, use the NOUSE option. This ensures that the component will not try to execute the file when you exit from the system editor.

DETAIL|NODETAIL

DETAIL copies the contents of the revised file into the temporary save file; NODETAIL doesn't.

If you are revising a permanent file with the REVISE statement and you specify NODETAIL, then QDESIGN writes a USE statement to the current qksave file. QDESIGN processes the USE statement (the default) when you exit from the editor, as in

```
> USE ORDERS NODETAIL
```

If you revise a permanent file using the NODETAIL option, QDESIGN writes a USE statement to the current qksave file. QDESIGN processes the USE statement when you exit from the editor, as in

```
> USE ORDERS NODETAIL
```

Limit: The NODETAIL option isn't valid with qksave.

Default: DETAIL

LIST|NOLIST

LIST displays each statement from the revised file as it is processed; NOLIST doesn't.

Default: LIST

USE|NOUSE

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

Default: USE

Discussion

The REVISE statement starts an editor session in any file that you specify or the current temporary save file, qksave. When you exit from the editor, the file is processed as if you had entered a USE statement (with the LIST and DETAIL options).

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

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

Choosing an Editor

MPE/iX

By default, the PowerHouse UDC uses the file equation:

```
:FILE COGEDITR=EDITOR.PUB.SYS
```

If you want to use an editor other than EDIT/3000 when you invoke REVISE, change this file equation. For example, if you want to designate MYEDITOR as your editor, enter

```
:FILE COGEDITR=MYEDITOR
```

The REVISE statement also uses a file equation for the file EDTTEXT, if it exists. For example, if you want to designate MYFILE as the input file to the editor, enter

```
:FILE EDTTEXT=MYFILE
```

If you elect to use other editors, they must comply with the HP standard regarding the entry point BASICENTRY and the input file specification EDTTEXT.

OpenVMS

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

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

causing the REVISE statement to invoke the EDT editor.

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

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

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

If you intend to use the **nodcl** program parameter to restrict user access to the operating system, we further recommend that you do no select editors (such as TPU) that provide operating system access. When **nodcl** is in effect, users will continue to be able to access the system editor through the REVISE statement.

UNIX, Windows

The editor is set using the PHEDIT environment variable. If PHEDIT is not defined, the system checks the environment variable EDITOR. If you have not defined either of these variables, the REVISE statement fails.

RUN

Executes QTP.

Syntax

RUN filespec [options...]

filespec

Specifies the name of an existing compiled QTP run file.

Options

RUN Options		
AUTO	CLEAR	HIDDEN
ID|NOID	[ENTRY] IF	INPUT B|C|SAME
LABEL|NOLABEL	MARK|NOMARK	NOCONSOLE
NOWARN	ON ERROR	REFRESH
RESPONSE	WAIT|NOWAIT	

AUTO

Invokes QTP automatically when the standard entry sequence reaches this statement, if the ENTRY procedure was generated by QDESIGN.

CLEAR option

Clears an area of the terminal memory before invoking QTP. Any output to the terminal from the run appears starting on the first line of the cleared area. Lines cleared are refreshed automatically when the screen is reactivated and QUICK is ready to prompt the user. If multiple COMMAND, REPORT, or RUN statements are combined with ID SAME, the area cleared is refreshed after the last statement in the chain has completed execution and QUICK is ready to prompt the user.

ALL

Clears the entire terminal memory.

SCREEN

Clears the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears the area between and including lines n to m, numbering from the first line of terminal memory. LINES n by itself clears line n only.

HIDDEN

Suppresses the screen ID display, but lets the user reference a field by ID-number in MARK mode.

ID n [AT [line],column]
ID NEXT [AT [line],column]
ID SAME
NOID

Control how and where QUICK screen field ID-numbers are assigned.

ID n

Explicitly specifies an ID-number.

Limit: 1 to 99

ID NEXT

Uses the next ID-number in sequence.

AT [line],column

Positions the first digit of the ID-number at the specified line and column relative to the starting line of the screen. If the line is missing, the current line is assumed.

ID SAME

Instructs QDESIGN to omit the ID-number on the run. To execute the run, use the ID-number of the previous field.

NOID

States that no ID-number is assigned to this run and the run can't be referenced from the Action field.

[ENTRY] IF condition

Invokes the specified run in the standard entry sequence only if this condition is satisfied. QDESIGN generates an identical IF condition in the default ENTRY procedure.

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

Limit: The IF option is evaluated only during the standard entry sequence; otherwise, it is ignored.

Default: If the condition is satisfied, AUTO is assumed.

INPUT B|C|SAME (MPE/iX)

Puts the terminal in the specified input mode prior to executing the run. The terminal is put back into the original mode after completion of the run.

Default: The mode it was in before QUICK was invoked.

B

Puts the terminal in Block mode. This should only be used for commands that must run in Block mode.

C

Puts the terminal in Character mode.

SAME

Leaves the terminal in the current input mode. If the screen has Block mode capability, this option should only be used for commands that do not write to the terminal.

LABEL [string] [AT [line],column] | NOLABEL

Declares the label and its position.

[string] [AT [line],column]

Indicates the run label and, optionally, the position of the label on the screen.

The AT option positions the first character of the label at the specified line and column relative to the starting line of the screen. If the line isn't specified, the current line is assumed.

Default: The item name or the first word in the COMMAND string.

NOLABEL

Specifies that no label is to appear for the run.

MARK|NOMARK

MARK enables fieldmarking for a run with an ID-number.

NOMARK disables the default fieldmarking for a run with an ID-number when fieldmarking is enabled.

NOCONSOLE (Windows)

Suppresses opening a Command Console window. Normally QUICK opens a second Command Console window to run QTP. If QTP runs in the background, does not require user input, or does not display useful output, the second command console window may not be necessary.

NOWARN

Specifies that operating system warning messages issued during the execution of the run should not be displayed.

ON ERROR CONTINUE|TERMINATE

Specifies the action to be taken if an operating system error occurs during the execution of a run. If TERMINATE is in effect, an operating system error causes QUICK to process the error as it would for an ERROR verb. If CONTINUE is specified, an operating system error is ignored and processing continues as if the error had not occurred.

For information on the ERROR verb, see (p. 420).

Default: TERMINATE

REFRESH option

Clears and rewrites an area of the terminal memory when the screen is reactivated and QUICK is ready to prompt the user. REFRESH options are performed before, and in addition to, an automatic refresh from any CLEAR option.

ALL

Clears and rewrites the entire terminal memory.

SCREEN

Clears and rewrites the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears and rewrites the area between and including lines n to m, numbering from the first line of terminal memory. LINES n alone refreshes line n only.

RESPONSE

Prompts the QUICK user for a response after the run has executed. QUICK resumes processing after the user responds, preventing the screen from being refreshed immediately.

WAIT|NOWAIT (Windows)

The WAIT option instructs QUICK to suspend current screen processing until the run has executed, at which time control returns to the screen. The NOWAIT option specifies that screen processing continues immediately and the run executes concurrently.

Default: NOWAIT

Discussion

The RUN statement executes QTP and takes the compiled run as a parameter.

SAVE

Saves QDESIGN source statements in a permanent file.

Syntax

SAVE filespec [CLEAR]

filespec

Names a permanent file in which to save source statements.

CLEAR

Removes the contents of QDESIGN's temporary source statement save file after copying the contents to a permanent file. The save file is QKSAVE (**MPE/iX**) or qksave.qks (**OpenVMS, UNIX, Windows**).

Discussion

The SAVE statement relates to QDESIGN's temporary save file. Statements are written to this file as you enter them. The SAVE statement itself isn't included in the file. The SAVE statement creates a permanent copy of the save file at the point at which the SAVE statement was entered. You can use the saved contents as a source file for documentation and future changes, or as a working file for modification using the text editor. The saved statements can also be processed by QDESIGN with the USE statement.

Example

The following example demonstrates the correct use of the SAVE statement. In this example:

- All statements before CANCEL CLEAR are purged from QDESIGN's temporary save file, qksave.
- The SAVE statement saves all of the statements since the CANCEL CLEAR in a permanent file.

```
> SCREEN NEWORD
>
> FILE ORDERMASTER PRMARY
                   ^^^^^^^
*E* Expected: ALIAS AUDIT COUNT DELETE DESIGNER DETAIL MASTER MYVIEW NEED
NOAPPEND NODELETE NOITEMS OCCURS OPEN PRIMARY REFERENCE SECONDARY <eol>
> CANCEL CLEAR
>
> SCREEN NEWORD
>
> FILE ORDERMASTER PRIMARY
> GENERATE NOLIST NODETAIL
>
> SAVE NEW-ORD
```

SAVE copies everything in QDESIGN's temporary save file, qksave, to a permanent file. It's important to clear the temporary save file of erroneous statements with the CLEAR option of the CANCEL statement.

SCREEN

Names the screen and describes its characteristics.

Syntax

SCREEN filespec [option]...

filespec

Specifies the name of the file in which the compiled QUICK screen will be stored.

Options

SCREEN Options		
ACTION	ACTIONBAR|NOACTIONBAR	ACTIVITIES
AUTOMODIFY	AUTORETURN	AUTOUPDATE
BLOCKMODE [EXTENDED]	COMMIT ON|NOCOMMIT	FIELDMARK|NOFIELDMARK
FOR	FROM	HELP POPUP
LOCK	MENU	MESSAGE
MESSAGE POPUP	MODE|NOMODE	NOSEQUENTIAL
ON LINE	PANEL|NOPANEL	PREDISPLAY
RECEIVING	SLAVE	STOPSCREEN
TRANSACTION MODEL	USERS INCLUDE	WINDOW

ACTION [STARTUP] [LABEL string] [AT [line], column] NOACTION

ACTION enables screen Action field prompting. NOACTION suppresses Action field prompting.

NOACTION and ACTION are mutually exclusive.

Default: ACTION AT 1,8

STARTUP

Specifies the Action field as the prompt mode at startup. This option must come before the LABEL or AT options.

LABEL string

Replaces the default string.

Limit: The maximum LABEL string length is 9 characters less than the screen width.

Default label: ACTION:

AT [line], column

AT specifies where the Action field label is to be located on the screen. A line is a line relative to the current screen. It can have a value of 1 to 24.

The Action field can be placed on any line from 1 to the greatest number of lines allowed on the screen. The column value can range from 1 to the largest number enabling the label and the Action field to fit on the screen. (The Action field is 9 characters long.)

Default placement: AT 1, 8

If the Action field and label exceed the screen size, an error is issued.

ACTIONBAR [ON LINE n] [STARTUP]
NOACTIONBAR

ACTIONBAR enables the Action bar; NOACTIONBAR suppresses the Action bar.

ON LINE application line

Specifies the Action bar line location.

Default: The first line of screen data.

STARTUP

Specifies the Action bar as the prompt mode at startup.

ACTIVITIES activity [,activity]...

Specifies the allowed screen activities. The four activities ENTRY, FIND, CHANGE, and DELETE encompass the functions that the normal screen can perform. To remove an activity or activities, specify on the ACTIVITIES option only those actions that you want. All others are excluded. (There is one exception: ACTIVITIES ENTRY generates a DELETE procedure so that you can delete newly-entered records prior to updating. Since it does not generate a FIND procedure, you cannot delete existing records.)

For screens with Append processing, the ENTRY activity allows data records to be appended in Entry mode and the CHANGE activity allows detail data records to be appended in Find mode.

If CHANGE or DELETE is specified, FIND is implied. If the ENTRY activity is omitted, new data records can't be added (except when you use Append processing for DETAIL records from Find mode).

Default: ENTRY, FIND, CHANGE, DELETE

AUTOMODIFY

Runs the MODIFY procedure after a record is retrieved in Find mode. Without this option, the QUICK screen user must enter a Modify command (M) in the Action field.

AUTORETURN

Specifies that QUICK is to exit this screen as soon as the update sequence (PREUPDATE, UPDATE, and POSTUPDATE procedures) is performed. The update can be done manually or automatically with the inclusion of the AUTOUPDATE option. If an EXIT procedure is included in the screen design, it is executed before QUICK returns to the higher-level screen.

AUTOUPDATE

Updates automatically at the end of the standard entry sequence if all fields are entered with acceptable values.

BLOCKMODE[EXTENDED] (MPE/iX)

Indicates that QUICK can run this screen in Block mode. The BLOCKMODE option must be present to run a PANEL screen on a Block mode terminal.

The EXTENDED option increases by one the input size of each field on the screen when the screen is in Block mode.

For more information on panel mode, see Chapter 2, "QUICK User Interface".

COMMIT ON automatic-commit-point|NOCOMMIT

Indicates the default points at which automatic commits are executed by QUICK.

Default: COMMIT ON UPDATE

NOCOMMIT

Indicates that QUICK does not generate automatic commit actions.

automatic-commit-point

Determines the points at which an automatic commit for the COMMIT ON option occurs during screen processing for read-write (Update) transactions.

The automatic commit points are UPDATE, MODE, NEXT PRIMARY, and EXIT.

UPDATE	This option is the default for the Update and Consistency transactions. Use the ON UPDATE option to ensure that related updates (for example, updates to primary and secondary data) are grouped together, but keep individual transactions relatively short. Locally active transactions are committed: • when an Update action is completed (before the POSTUPDATE procedure) • when the screen mode changes (before the PREENTRY and PATH procedures) • when the user exits the screen (before and after the EXIT procedure)
MODE	This option is the default for the Query transaction. Use the ON MODE option to ensure that changes to a series of existing records (for example, all employees in a certain branch or all tasks in a project) are committed or rolled back as a group. Locally active transactions are committed when: • the screen mode changes (before the PREENTRY and PATH procedures) • on screen exit (before and after the EXIT procedure)
NEXT PRIMARY	Use this option if you want to group all detail records (perhaps requiring several entry or display screens) together with primary and secondary records and treat them as a unit to be committed or rolled back. Locally active transactions are committed: • when the user starts an entry sequence (before the PREENTRY procedure) • when the user retrieves the next set of primary records (before the FIND procedure • when the user exits the screen (before and after the EXIT procedure).
EXIT	Use this option when all activity done on a screen is to be treated as a single transaction. Locally active transactions are committed when the screen is exited (after the EXIT procedure).

FIELDMARK [RETAIN] [STARTUP]
NOFIELDMARK

Allows the user to select a screen item for processing by highlighting the ID or label using the cursor keys.

NOFIELDMARK requires the user to select screen items for processing by entering the ID number in the Action field.

Default: NOFIELDMARK

RETAIN

Specifies that the screen item ID is retained for Action command processing when the context reverts to the Action field or the Actionbar.

STARTUP

Specifies that fieldmarking is the prompt mode at startup.

FOR n [LINES]

Allows a screen length of n lines not including the message line. FOR and FROM options are mutually exclusive.

Limit: 1 to 23

Default: 23

FROM [application line1],column1 TO [application line2],column2

Specifies the coordinates for the screen.

The FROM coordinates set the top left-hand corner of the screen. The TO coordinates set the bottom right-hand corner of the screen.

The FROM and TO coordinates allow screens to overlay one another. The FROM and FOR options are mutually exclusive.

For an explanation of the difference between screen lines and application lines, see Chapter 5, PowerHouse Language Rules, in the *PowerHouse Rules* book.

Default: The default FROM coordinates are 1,1. The default TO coordinates are 23,80.

Limit: Can't be used with either of the ON LINE n or FOR n LINES options. Application line1 is 1-240; application line2 cannot exceed line1 by more than 23. The valid range for column1 and column2 is 1-132.

HELP POPUP [FROM [application line1], column1] [TO [application line2], column2]

Defines a pop-up window for display of the extended help information.

The window overlays the current screen; the current screen is restored when the pop-up window is closed.

A pop-up window has a single-line border that requires 2 rows and 2 columns.

FROM [application line1], column1

The top-left corner of a pop-up window.

Default: 8,5

TO [application line2], column2

The bottom-right corner of a pop-up window.

Default: 16,75

LOCK option

where option is one of:

MPE/iX:	FILE [FOR UPDATE]
	RECORD [FOR UPDATE]
	BASE
OpenVMS, UNIX, Windows:	FILE [FOR UPDATE]
	RECORD [FOR FIND|UPDATE]

Generates default LOCK and UNLOCK verbs for all files and records involved in the UPDATE procedure. If omitted, QUICK automatically locks files with the PUT verb. The FILE, RECORD and BASE(MPE/iX) options indicate the lock level.

By using the LOCK FILE|RECORD option in combination with FOR UPDATE, the screen designer ensures that all updatable files are granted exclusive locks for the duration of the UPDATE procedure. This strategy prevents changes to the files or records between updates, but severely restricts other users and programs who are trying to access the files. Based on the level you've specified for the update, an UNLOCK ALL FILE or an UNLOCK ALL RECORD statement is always generated at the end of the UPDATE procedure when the LOCK FOR UPDATE is used.

With the FOR UPDATE option, a LOCK verb is generated at the beginning of the UPDATE procedure and an UNLOCK ALL verb is generated at the end of the UPDATE procedure.

Default: An exclusive file-level lock.

Limit: The LOCK option is not valid for relational files.

MPE/iX

The LOCK RECORD option affects only IMAGE files. If the LOCK RECORD option is specified and no IMAGE database files are associated with the update, QUICK displays an error message.

OpenVMS, UNIX, Windows

By using the LOCK RECORD FOR FIND option, the screen designer can provide a secure and efficient capability for online applications that require a high degree of reliability. The LOCK verb generated at the beginning of the FIND or DETAIL FIND procedure locks all records as they are read. The success of updates is guaranteed by locking the record for the entire duration of the FIND procedure. Because QUICK unlocks all files when it prompts the user during the find/modify/update sequence, the FOR FIND option is only allowed with LOCK RECORD.

An UNLOCK statement is not generated at the end of the FIND procedure with the LOCK RECORD FOR FIND option so that the data on the screen can remain locked while it is displayed. QUICK continues to unlock each record before it reuses or clears the record's corresponding buffer.

With the FOR FIND option, a LOCK verb is generated at the beginning of the FIND procedure for each FILE|RECORD in the screen. No UNLOCK verb is generated as the records are unlocked when the next find sequence is started.

MENU

Indicates that the screen is a menu screen. If neither MENU nor SLAVE is specified, the screen is a regular data screen.

Limit: Can't be specified with the SLAVE option.

MESSAGE [ON] [LINE] n

Positions the message line on the specified application line (n).

Limit: 1 to 240 (maximum number of application lines)

Default: The line after the last line of the screen as specified by either the FOR or TO options. If neither option is used, the default is 24.

MESSAGE POPUP [severity]... [FROM [application line1], column1] [TO [application line2], column2]

Defines a pop-up window for a message display that can overlap the current screen. The current screen is restored when the window is closed.

A pop-up window has a single-line border that requires 2 lines and 2 columns.

Default: All messages are displayed on the message line. If the MESSAGE POPUP option is specified, the messages are displayed in pop-up windows if either of the following conditions is true:

1. The severity level of the message to be displayed is specified in the severity list.
2. Only messages with a severity of ERROR or SEVERE are displayed when no severity is specified.

severity

One of: INFORMATION, WARNING, ERROR, or SEVERE.

FROM application line1, column1

The top-left corner of a pop-up window.

Default: 8,5

TO application line2, column2

The bottom-right corner of a pop-up window.

Default: 16,75

MODE [LABEL string] [AT [line], column]
NOMODE

MODE enables the Mode field location and label. NOMODE suppresses the Mode field.

NOMODE and MODE are mutually exclusive.

Default: MODE AT 1,1

LABEL string

Displays another string in place of the default string.

Limit: The maximum LABEL string length is 1 character less than the width of the screen.

Default label: MODE:

AT [line], column

The Mode field can be placed on any line from 1 to the greatest number of lines allowed on the screen. The column value can range from 1 to the largest number enabling the label and the Mode field to fit on the screen. (The Mode field is 1 character long.)

Default placement: AT 1, 1

NOSEQUENTIAL

Suppresses the automatic generation of sequential access to the data records in the PRIMARY file.

ON [LINE] application line

Specifies the starting position for the screen on the terminal.

Limit: 1 to 240

Default: 1

PANEL|NOPANEL

Specifies whether or not generated procedure code will contain the options and constructs required to process QUICK screen fields in panel mode.

SCREEN statements with the PANEL or NOPANEL options override SET PANEL and SET NOPANEL statements.

PANEL

Generates the APPEND, ENTRY, MODIFY, PATH, and SELECT procedures, and includes BLOCK TRANSFER control structures in each. BLOCK EACH and BLOCK ALL options on CLUSTER statements affect the way the BLOCK TRANSFER control structures are generated in the APPEND and ENTRY procedures.

NOPANEL

Generates only the APPEND, ENTRY, and PATH procedures, and doesn't include BLOCK TRANSFER control structures in them. BLOCK EACH and BLOCK ALL options on CLUSTER statements are ignored.

Generated FIELD statements, CLUSTER statements, and procedure code do not contain the options and control structures required to process Data fields using Panel input.

Default: NOPANEL

PREDISPLAY

Displays initial values in all fields when the screen appears.

RECEIVING record|item [,record|item]...

Declares the record-structures, defined items, and temporary items that are received by this screen from a higher-level screen.

The RECEIVING list must be specified in the same order as the PASSING list on the SUBSCREEN statement (or RUN SCREEN verb) in the higher-level screen that calls this screen, although the names can be different. To be accessible, record-structures and items in the list must be declared in the screen.

Limit: A combined maximum of 16 records and items can be received.

SLAVE

Indicates that the screen is a slave screen. Slave screens don't contain any PRIMARY record-structures; rather, they only reference record-structures that are passed from higher-level screens and are declared as MASTER files. SLAVE screens are typically used when all the required data entry cannot fit on the driver screen.

The MENU and SLAVE options are mutually exclusive. If neither is specified, the screen is a regular data screen. The PREDISPLAY option is assumed for all fields on a slave screen.

Limit: Can't be specified with the MENU option.

STOPSCREEN

Specifies that control is to return to this screen if the QUICK screen user enters a Return to Stopscreen command (^^) in the Action field of a lower-level screen.

The Return to Stopscreen action stops if there is a STOPSCREEN option on a lower-level screen, or if any balancing warnings or error messages are issued on intervening screens.

TRANSACTION [MODEL] [transaction-option]

Determines the transaction model that is used for the screen.

Default: Set in the dictionary. The dictionary default is the Concurrency model. Defaults for each operation can be customized using the SYSTEM OPTIONS statement in PDL.

For more information about transaction models, see Chapter 1, "About PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

The transaction-options are CONCURRENCY, CONSISTENCY, DUAL, and OPTIMISTIC.

CONCURRENCY

Specifies that the Concurrency transaction model is used.

CONSISTENCY

Specifies that the Consistency transaction model is used.

DUAL [option]...

Allows a screen to use one transaction model for Select mode operations and a different transaction model for Entry and Find mode operations. The syntax for the options of the Dual transaction model is:

SELECT [IN] CONSISTENCY|CONCURRENCY|OPTIMISTIC
ENTRY [AND] FIND] [IN] CONSISTENCY|CONCURRENCY|OPTIMISTIC

Defaults: SELECT IN CONCURRENCY; ENTRY AND FIND IN CONSISTENCY

If the same model is specified for both Select and Entry/Find mode, a warning message is issued.

The options specified on the SCREEN statement override any defaults specified in the dictionary.

DUAL is not a valid transaction model for Select or Entry/Find mode operations.

OPTIMISTIC

Specifies that the Optimistic transaction model is used.

USERS INCLUDE ALL|class[,class]

Restricts execution of the screen to the application security classes listed.

ALL

Includes all application security classes declared in the data dictionary, including the application security class UNKNOWN.

class[,class]...

Indicates a name or names in the data dictionary for an application security class. Only the classes specified are able to execute the screen. The application security class names must be declared in the data dictionary. The application security class UNKNOWN can be listed as a class.

Limit: 1 to 64 user classes

WINDOW [ON] [LINE]n

Positions the 24-row window starting on the specified application row(n).

Limit: 1 to 240

Defaults to the ON LINE value. For more information about ON LINE, see (p. 187). For information about ON LINE and stacking screens, see (p. 190).

Discussion

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

The SCREEN statement provides a file name for the screen, and allows you to specify screen-wide options and screen positioning.

MENU Screens

Menu screens are built by adding the MENU option to the SCREEN statement, and by using COMMAND statements, SUBSCREEN statements, and designer-written procedures to link the menu screen with other screens, programs, and functions.

In general, menu screens don't act on, or relate to, data items or files. However, to pass parameters to the entire system, you can declare temporary or defined items (as well as record-structures in REFERENCE and DESIGNER files) for a menu screen. You can also declare MASTER record-structures if a menu screen appears in the middle of a series of linked screens, and you must receive and pass files through the menu screen.

By default, if no fields are declared, the ENTRY, MODIFY, and SELECT procedures for a menu screen contain only the NULL verb. Similarly, the FIND and PATH procedures on a menu screen contain only the NULL verb, and the UPDATE and DELETE procedures contain only the DISABLE control structure. For more information about the DISABLE control structure and the NULL verb, see (p. 384) and (p. 449).

To perform file retrieval and data initialization, you can provide an ENTRY procedure. This procedure is executed only if the menu screen is in Entry mode. Menu screens are placed in Entry mode when

- Entry mode is specified in the Initial mode parameter of QKGO
- the screen started in Entry mode
- the Action field command E (or the Entry mode command, if this command has been redefined in QKGO) is entered in the Action field

When a menu screen is in Entry mode, the predefined condition CORRECTMODE is true, unless you have written an ENTRY procedure for the menu screen; in this case, the predefined condition ENTRYMODE is true during processing of the ENTRY procedure.

If a menu screen is started with no mode or in Find mode (that is, if F is specified in the Initial mode parameter of QKGO, if the screen is started in Find mode, or if F is entered in the Action field), the predefined condition CHANGEMODE is true.

Since a menu screen is normally in the Change or Correction phase, the user enters ID-numbers to select the appropriate action. A mode entered in the Action field can be passed to a screen invoked from the menu if the MODE SAME option is used on the SUBSCREEN statement.

Slave Screens

There is no PRIMARY file on a slave screen. The current file is passed from the higher-level screen and received as a MASTER file.

An update on any screen updates all items on the record-structure. As a result, updating a slave screen removes the necessity of updating on the calling screen. Similarly, updating a higher-level screen removes the necessity of updating all lower-level slave screens.

REQUIRED fields should always be on the main screen, since there is no way to force the QUICK screen user to enter data on a slave screen.

When leaving a slave screen, a backout action is only invoked when data local to the screen is changed and not updated. If data in a received MASTER file is changed, the backout action is not invoked because the slave screen is considered an extension of the calling screen and the file originated there. However if a locally declared DESIGNER file is changed and not updated, backout processing will occur upon leaving the screen.

The mode on the higher-level screen determines the mode and functions allowed on the slave screen. When the slave screen is called while the higher-level screen is in the Entry sequence, QUICK executes the Entry sequence on the slave screen and then prompts in the Action field.Corrections can then be entered or an update can be performed. QUICK does not, however, allow a Find or Select action, or a repeat of the Entry sequence.

When the slave screen is called after the completion of the Entry sequence on the higher-level screen (correct phase), QUICK only allows correction to be entered or an update to be performed on the slave screen.

If the slave screen is called after retrieving data in Find or Select mode, QUICK only allows changes to be entered or an update to be performed. Using Find, Select, or Entry modes causes the screen to display an error message.

You can, however, force the Entry sequence on the slave screen when the higher-level screen is in the correct phase by using the MODE E option of the SUBSCREEN statement. For example

```
> SUBSCREEN HISTSUB1 PASSING HISTORY MODE E AUTO
```

Stacking Screens

QUICK supports up to 240 screen lines ("application lines") regardless of the terminal's actual memory capacity.

Applications addressing more than 24 screen lines (rows) require the designer to keep track of how the QUICK screens appear to the user since only 24 lines can be seen by the user at any single time.

The option, WINDOW ON LINE n, allows the designer to decide which application line (n) will be the top line displayed on the terminal. If the lines of a screen are not numbered greater than 24, QUICK assumes the window line to be 1. If a screen (or its message line) extends beyond application line 24, QUICK assumes the window line is the screen's first line. Some terminals do not support QUICK screens placed on windows in the middle of a terminal page.

In the following example, there are six screens placed on application lines 1 to 72 as follows:

```
> SCREEN A
.
.
.
> SCREEN B ON LINE 13 FOR 11 LINES
.
.
.
> SCREEN C ON LINE 25 FOR 12 LINES &
>   MESSAGE ON LINE 48
.
.
.
```

```
> SCREEN D ON LINE 37 FOR 11 LINES  &
>   WINDOW ON LINE 25
.
.
.
> SCREEN E ON LINE 37 FOR 11 LINES  &
>   WINDOW ON LINE 25
.
.
.
> SCREEN F ON LINE 49
```

The following table shows what the user sees when each screen is active:

Active Screen	What the user sees
A	Screen A alone.
B	A split screen with part of screen A on top and screen B on the bottom.
C	A split screen with screen C on top and a blank screen, screen D, or screen E on the bottom, depending on whether and which of screen D or E was called.
D	A split screen with screen C on top and screen D on the bottom.
E	A split screen with screen C on top and screen E on the bottom.
F	Screen F alone.

If, at execution-time, the terminal has extended memory capacity beyond the standard displayable 24 lines, QUICK makes use of this extra memory to avoid rewriting the background when flipping between screens. If a QUICK screen is still in terminal memory, QUICK simply adjusts the window positioning. On the other hand, if the screen is lost from terminal memory, QUICK refreshes the user's display so that what the user sees is the same as if the terminal had sufficient memory capacity.

Screens outside of the current window may not always be displayed with up-to-date values. In fact, QUICK avoids writing anything to the terminal until all required data for the currently-displayed window is available. Even on split screens the data displayed in the current window for currently inactive screens may not be up-to-date. This is especially true in balancing and in other situations where lower-level screens update data for display on higher-level screens. In a balancing situation, control counts or sums computed on a lower-level screen aren't updated on the higher-level screen until the user returns to that higher-level screen. The REFRESH option on the FIELD statement must be used to bring such fields up-to-date on return to the screen.

Application Lines and Terminal Memory

The screen stacking options of the SCREEN statement do not act alone. They act in conjunction with the Application lines parameter of QKGO, and with usable terminal memory.

Application Lines

A maximum of 240 screen lines (regardless of the usable terminal memory) can be specified at execution-time with the Application lines parameter of QKGO. (The default is 48.) The designer can take advantage of application lines by using the WINDOW option of the SCREEN statement. Since windows are 24 lines long, the number of application lines should be specified to be evenly divisible by 24. Split or overlapping windows can be used, but since writing to terminal memory is done in 24-line windows, it may be difficult to produce a smooth screen-to-screen transition.

If the designer specifies a WINDOW option with a line number greater than the number of application lines available at execution-time, an overlay or mapping is done based on the number of lines available. For example, if 72 application lines were available at execution-time, and a window was specified to start on line 97, it is mapped onto application lines starting at line 25 (97 minus 72).

The line numbers in the screen stacking options - ON LINE n, MESSAGE ON LINE n, and WINDOW ON LINE n - all refer to line numbers (n) within application lines. On all other statements, row numbers are relative to a position on the screen, or relative to line 1 of the screen. The FOR n LINES option specifies how many rows the screen contains.

Usable Terminal Memory

Each terminal recognized by QUICK has a corresponding terminal profile containing the characteristics required to control the cursor, highlighting, line drawing, and terminal memory. One of the pieces of information in this profile is the maximum number of lines of terminal memory that QUICK can use (to a maximum of 240). This number is also the maximum that can be used if the **term** program parameter, the terminal type prompt, or the data dictionary are used to specify the number of lines.

Unless otherwise specified, QUICK assumes 48 lines or the maximum number in the profile, whichever is lower (some terminals only have 24 lines of memory). If the number of lines is specified, the number used is either the lower of the lines specified or the value in the profile. For example, if the profile allows a maximum of 120 lines, and 144 lines are specified on a terminal-type prompt, 120 lines are used. Specifying more lines than the terminal actually has available leads to unpredictable display results. Specifying more terminal memory than your application needs can have a negative impact on performance.

Regardless of the number of lines of terminal memory specified, the number of lines actually used will not be greater than the number of application lines specified in QKGO. The number of lines used also depends on screen stacking and windowing options. These options may or may not make use of the total number of application lines available. The number of lines of terminal memory should be specified to be evenly divisible by 24.

When specifying the number of terminal lines to be used, you should take the type of terminal memory into consideration. If the memory is based on a fixed number of lines, even if the lines are completely filled, there is no problem; 48 lines will always be 48 lines. Some terminals base their memory on a fixed number of bytes, with the actual number of lines dependent on how full each line is. A line's contents may include many cursor and highlight control characters which, although not visible, occupy bytes and use terminal memory. A system of screens must be physically tested on terminals of this type to determine if problems will occur.

While the screen stacking options of the SCREEN statement refer to application lines, the CLEAR and REFRESH options and verbs refer to terminal memory only. Any numbers specified are mapped to lines actually used on the terminal.

Mapping from Application Lines onto Terminal Memory

A system of screens may be mapped into application lines by using the screen stacking options. When displaying screens on the terminal, application lines are mapped into terminal memory. If the number of application lines does not exceed the number of usable terminal lines, the mapping is a straightforward one-to-one correspondence. If there are more application lines than usable terminal lines, the mapping is the same as if a WINDOW option specifies a line number greater than the number of application lines. If 240 application lines were used and specified in QKGO, and only 72 lines of terminal memory were available at execution-time, the mapping is as outlined in the following table, assuming full 24 line windows with no overlapping:

Application Lines		**Terminal Lines**	
Window	**Line numbers**	**Window**	**Line Numbers**
1	1-24	1	1-24
2	25-48	2	25-48
3	49-72	3	49-72
4	73-96	1	1-24
5	97-120	2	25-48

Application Lines		Terminal Lines (Continued)	
Window	**Line numbers**	**Window**	**Line Numbers**
6	121-144	3	49-72
7	145-168	1	1-24
8	169-192	2	25-48
9	193-216	3	49-72
10	217-240	1	1-24

If a screen is already in terminal memory when it is linked to or returned to, a simple adjustment of the cursor position is all that's required to display it to the user (a "screen flip"). If the screen is not in terminal memory but is in application lines, it can be redisplayed from there. If the screen is not in terminal memory or application lines, it must be reconstructed.

Using this information, plus a knowledge of the target terminals, the designer can plan where a screen is to appear within terminal memory and what other screens it overlays. You can obtain significant gains in processing efficiency by using application lines and terminal memory to their fullest.

Action Bars

An Action bar presents a list of QUICK actions or menus in a menu bar that extends across the terminal window. You can invoke an Action command by selecting the appropriate entry in the Action bar or pull down menu as an alternative to entering the action at the Action field prompt.

With an Action bar, you can associate commands with a more descriptive label, and also show all the commands that are available to the user.

Adding an Action Bar to Your Screen Design

To create an Action bar:

1. Specify the ACTIONBAR option of the SCREEN statement.
2. Use the ACTIONMENU statement to define the menus and actions you want to place on the Action bar.
3. Use the MENUITEM statement to define the actions for each menu to be pulled down from the Action bar.

Action Bar Menus and Actions

The Action bar can contain both menus and actions. To specify a menu, enter the ACTIONMENU statement without indicating an action option, as in

```
> ACTIONMENU LABEL "EMPLOYEES"
```

To specify the action that's performed, use the ACTION option, as in

```
> ACTIONMENU LABEL "CREATE" ACTION ENTRY
```

You can specify any QUICK Action command as an ACTION option. If the action requires an ID-number or ID-number range as a parameter, you can specify these explicitly or you can obtain the ID-numbers at run time by using the MARK or PROMPT options.

If you specify PROMPT, QUICK prompts you for ID-number values with a prompt box. If you specify MARK, QUICK determines the value from the current FIELDMARK setting. If there is no current MARK and MARK has been specified, QUICK prompts for the ID-numbers.

If you don't want your Action bar to have pull-down menus, exclude MENUITEM statements from your design statements. Instead, you can specify Action commands in the Action bar.

But if you want to specify a pull-down menu, enter MENUITEM statements after the corresponding ACTIONMENU statement, as in

```
> ACTIONMENU LABEL "EMPLOYEES"
> MENUITEM LABEL "LOCATE" ACTION FIND
```

```
> MENUITEM LABEL "NEW" ACTION ENTRY
```

If you specify MENUITEM statements following the ACTIONMENU statement, QDESIGN constructs a menu that appears to "pull down" from the Action bar when the Action menu is selected.

Each MENUITEM statement creates a new entry on the menu. The menu can accommodate any size label and any number of items: the menu border adjusts to accommodate the largest menu label, and the menu scrolls to fit the number of menu items.

Generally, menus pull down from the Action bar; that is, they appear below the corresponding Action menu on the Action bar. However, if you specify that the Action bar appear on the last three lines of the terminal, the menus pop up above the Action bar.

Action Bars and Action Fields

You can specify that both an Action bar and an Action field appear on a screen. When the screen runs, only one mode is active at a time, but you can switch between the two modes. This way, you can make a limited set of commands available through an Action bar for the novice user but still enable an Action field for the more experienced user.

If both the Action bar and Action field are available, the screen initially runs Action field prompting mode by default. To specify that the Action bar should be the initial mode, add the STARTUP option to the ACTIONBAR option of the SCREEN statement.

If the Action field and Action bar appear on the same line, only the current prompting mode is visible. If they are not on the same line, both the Action bar and Action field are visible, but QUICK only prompts you at the active mode.

Console Window Size (Windows)

The size of a Command Console window can be controlled by the properties setting of that window. However, QUICK normally operates with a screen size of 80 to 132 columns by 24 lines. QUICK will control the Command Console window size so that the screen can be displayed in its entirety whenever possible.

For example, if QUICK runs an 80 by 24 screen and then a 132 by 24 screen, it will resize the Command Console window automatically when it switches between the two screens. The exception to this is when the Command Console window is running in "Full Screen" mode. In "Full Screen" mode, screens larger than 80 columns will only display 80 columns at a time.

If the user resizes the Command Console window while QUICK is running, then QUICK may not display the entire screen or it may not produce the desired output.

Examples

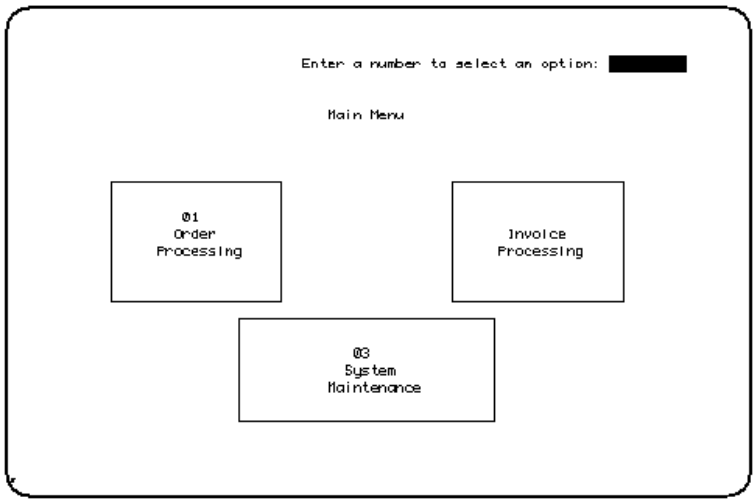
The following examples show the syntax for setting up a MENU screen, in which no FILE statements are specified. MENU screens are commonly used to call other screens, and generally they contain one SUBSCREEN statement for each screen that can be called from the menu being established.

NOMODE suppresses the display of a Mode indicator. The ACTION LABEL and AT options position the Action field and specify a label for it.

```
> SCREEN MAINMENU MENU &
>   NOMODE &
>   ACTION LABEL &
>   "Enter a number to select an option: " &
>   AT 1,33
>
> SKIP TO LINE 4
>> TITLE "Main Menu" CENTERED
>
> SKIP TO LINE 8
>
> DRAW 8,10 TO 15,30
> DRAW 8,50 TO 15,70
> DRAW 16,25 TO 22,55
>
```

```
>   SUBSCREEN ORDMAIN &
>   NOLABEL &
> ID 1 AT 10,19
>
> SUBSCREEN INVMAIN &
>   NOLABEL &
>   ID 2 AT 10,59
>
> SUBSCREEN MAINT &
>   NOLABEL &
>   ID 3 AT 18,39
>
> TITLE "Order" AT 11,18
> TITLE "Invoice" AT 11,57
>
> TITLE "Processing" AT 12,16
> TITLE "Processing" AT 12,56
>
> TITLE "System" AT 19,38
> TITLE "Maintenance" AT 20,35
>
> BUILD
```

The QUICK screen that results serves as a main menu for an entire system:



Setting Up QUICK's Panel Mode of Operation

By default, QUICK screens are not PANEL screens. Each field is treated as a distinct "block" of information. To group fields into blocks for processing, use the PANEL option of the SCREEN statement.

- PANEL causes the resulting QUICK screen to block fields based on CLUSTER statements.
- ACTIVITIES limits the screen activities; in this case, no ENTRY activity is allowed since the screen is designed to make changes to existing data records
- No ENTRY procedure is generated because of the ACTIVITIES options specified on the SCREEN statement.
- The PATH, SELECT, and MODIFY procedures contain BLOCK TRANSFER control structures. No BLOCK TRANSFER control structures are present for NOPANEL screens.
- The SELECT and MODIFY procedures are generated only for PANEL screens. However, you can code them for any QUICK screen, whether or not PANEL is specified.

```
> SCREEN MODCUST PANEL &
>    ACTIVITIES FIND, CHANGE, DELETE
>
> FILE CUSTOMERS PRIMARY
>
> SKIP TO LINE 8
```

```
> ALIGN (,25,40) >
> CLUSTER BLOCK EACH
> FIELD CUSTOMERKEY OF CUSTOMERS &
>   REQUIRED &
>   NOCHANGE
> CLUSTER
> SKIP 1
> CLUSTER BLOCK ALL
> FIELD CUSTOMERNAME OF CUSTOMERS &
>   REQUIRED &
>   NOCHANGE
> FIELD CITY OF CUSTOMERS
> FIELD STREET OF CUSTOMERS
.
.
.
> FIELD PROVSTATE OF CUSTOMERS &
>   ID SAME &
>   LABEL "Prov/State"
> FIELD POSTALZIP OF CUSTOMERS &
>   ID SAME &
>   LABEL "Pcode or

> FIELD PHONENUMBER OF CUSTOMERS &
>   ID SAME &
>   LABEL "Phone"
> CLUSTER
>
>BUILD LIST
>
> PROCEDURE PATH
> BEGIN
> BLOCK TRANSFER SEQUENCED
> BEGIN
> REQUEST CUSTOMERKEY OF CUSTOMERS
> END
> IF PROMPTOK FOR CUSTOMERKEY OF CUSTOMERS
> THEN LET PATH = 1
> IF PATH = 0
> THEN BEGIN
> BLOCK TRANSFER SEQUENCED
> BEGIN
> REQUEST CUSTOMERNAME OF CUSTOMERS
> END
> IF PROMPTOK FOR CUSTOMERNAME OF CUSTOMERS
> THEN LET PATH = 2
> END
> IF PATH = 0
> THEN ERROR "Key required."
> END
>
> PROCEDURE FIND
> BEGIN
> IF PATH = 1
> THEN GET CUSTOMERS &
>   USING CUSTOMERKEY OF CUSTOMERS
> IF PATH = 2
> THEN GET CUSTOMERS &
>   USING CUSTOMERNAME OF CUSTOMERS
> END
>
> PROCEDURE UPDATE
> BEGIN
> PUT CUSTOMERS
> END
>
> PROCEDURE DELETE
> BEGIN
```

```
>  DELETE CUSTOMERS
> END
> PROCEDURE SELECT
> BEGIN
> BLOCK TRANSFER
> BEGIN
> SELECT CUSTOMERKEY OF CUSTOMERS
> SELECT CUSTOMERNAME OF CUSTOMERS
> SELECT CITY OF CUSTOMERS
> SELECT STREET OF CUSTOMERS
> SELECT PROVSTATE OF CUSTOMERS
> SELECT POSTALZIP OF CUSTOMERS
> SELECT PHONENUMBER OF CUSTOMERS
> END
> END
.
.
.
> PROCEDURE MODIFY
> BEGIN
> BLOCK TRANSFER
> BEGIN
> DISPLAY CUSTOMERKEY OF CUSTOMERS
> END
> BLOCK TRANSFER
> BEGIN
> DISPLAY CUSTOMERNAME OF CUSTOMERS
> ACCEPT CITY OF CUSTOMERS
> ACCEPT STREET OF CUSTOMERS
> ACCEPT PROVSTATE OF CUSTOMERS
> ACCEPT POSTALZIP OF CUSTOMERS
> ACCEPT PHONENUMBER OF CUSTOMERS
> END
> END
```

This is an example of the COMMIT ON option:

```
> SCREEN EX3 &
>   TRANSACTION MODEL CONSISTENCY &
>   COMMIT ON MODE
```

This SCREEN statement results in QUICK employing a single consistency transaction that is committed at the beginning of each new screen mode.

SELECT

Applies a selection condition to retrieved data records.

Syntax

SELECT [IF] condition

condition

Establishes a condition that must be satisfied for the record-structure that is accessed by the screen. If the condition isn't satisfied, the record-structure is bypassed and the next record-structure is read.

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

Discussion

The SELECT statement is part of the data section and limits the data records accessed. The SELECT statement must follow the FILE statement to which it refers.

You cannot apply a SELECT statement to a CURSOR. To specify selection criteria when using a CURSOR, use the WHERE option or substitutions on the associated DECLARE CURSOR.

When accessing data records from the record-structure, only data records that satisfy the condition are made available to the screen. If more than one SELECT statement is associated with the record-structure, the data record must meet all selection criteria. A SELECT statement for a record-structure that is the object of a LOOKUP NOTON option is ignored when the lookup is performed. The LOOKUP NOTON option doesn't transfer data into the buffers, so there is nothing to compare.

Example

This example shows how the SELECT statement can be used to create a screen that tracks unfilled purchase orders.

"N" tells QUICK to retrieve data records from the PURCHASE record-structure that have been entered but not filled. With this design, QUICK screen users can access outstanding orders.

```
> SCREEN BACKORD
>
> FILE PURCHASE PRIMARY
>   SELECT IF SHIPPED = "N"
>
> FILE INVITEMS SECONDARY
>
> FIELD CUSTNAME
> FIELD CUSTNAME DISPLAY
> FIELD ORDERPARTS
> FIELD DATEORDERED
> FIELD EXPECTEDSHIPDATE LABEL "Exp. Ship. Date"
>
> BUILD
```

SET

Changes default settings for a QDESIGN session.

Syntax

SET DEFAULT|[option]...

DEFAULT

Resets all SET options to the following default values:

COMPILE

DETAIL GENERATE USE

LIST GENERATE LAYOUT USE

NESTING 50

NODETAIL PROCEDURES

NOLIST PROCEDURES

NOMENUKEYS

NOPANEL

NOPRINT

VERIFY DELETE SUMMARY

NOAUTONEXT

NOVERIFY ERRORS

WARNINGS

WRAPAROUND

The SET DEFAULT statement doesn't reset the DICTIONARY option.

SET DEFAULT resets the SET LIST|NOLIST option to SET NOLIST PROCEDURES SQL.

Options

SET Options		
ASSUMED	COMPILE|SYNTAX	DATABASE
DETAIL|NODETAIL	DICTIONARY	DOWNSHIFT|UPSHIFT|NOSHIFT
LIST|NOLIST	MENUKEYS|NOMENUKEYS	NESTING n
PANEL|NOPANEL	PRINT|NOPRINT	SAVE CLEAR
VERIFY|NOVERIFY	WARNINGS|NOWARNINGS	WRAPAROUND|NOWRAPAROUND

ASSUMED record-structure

Sets the assumed record-structure.

QUICK uses the assumed record-structure as the default record-structure name during the generation of FIELD statements.

If there is no SET ASSUMED statement, the assumed record-structure name is set to blank at the start of a screen design, and to the name of the primary record-structure when the FILE statement for the PRIMARY file is entered.

To change the assumed record-structure, the SET ASSUMED statement must follow the FILE statement for the PRIMARY file.

COMPILE|SYNTAX

COMPILE allows screens to be compiled and executed. SYNTAX allows QDESIGN to parse statements, but doesn't allow QDESIGN to execute the screen or compile it into a permanent file. The BUILD and GO statements are disabled when the SYNTAX option is specified.

Default: COMPILE

DATABASE database-name

For SQL support, each SQL statement requires a name to attach to the database. The database name must exist as a logical name in the current dictionary.

The name can also be set when loading the dictionary, or in the resource file, or by using the IN database option of SQL statements.

For more information about setting the database, see Chapter 1, "About PowerHouse and Relational Databases", in the *PowerHouse and Relational Databases* book.

DETAIL|NODETAIL [GENERATE] [PROCEDURES] [USE]

DETAIL and NODETAIL specify whether or not QDESIGN writes detailed statements to the source statement save file.

Although DETAIL performs the opposite function of NODETAIL, the two are not mutually exclusive unless they include identical options. An entry of either the SET DETAIL or SET NODETAIL statement without any options sets the DETAIL or NODETAIL options to GENERATE PROCEDURES USE.

Default: DETAIL GENERATE USE, NODETAIL PROCEDURES

GENERATE

Writes (if DETAIL was specified) or doesn't write (if NODETAIL was specified) the results of the GENERATE statement to the source statement save file. When NODETAIL GENERATE is specified, only the GENERATE statement is written to the source statement save file.

PROCEDURES

Writes (if DETAIL was specified) or doesn't write (if NODETAIL was specified) the procedures created by the BUILD statement to the source statement save file. When NODETAIL PROCEDURES is specified, only the BUILD statement is written to the source statement save file.

NODETAIL can be overridden by specifying BUILD DETAIL when creating the compiled screen.

USE

Writes (if DETAIL was specified) or doesn't write (if NODETAIL was specified) the contents of source statement files referenced by USE statements to the temporary save file: QKSAVE (**MPE/iX**) or qksave.qks (**OpenVMS, UNIX, Windows**).

When NODETAIL USE is specified, only the USE statement is written to the source statement save file.

DICTIONARY filespec [TYPE PHD|PDC]

Names the data dictionary used for the current session. By default, QDESIGN looks for a dictionary named phd.

Limits: The DICTIONARY option isn't saved in a compiled screen. The DICTIONARY option is valid only before the SCREEN statement or after the BUILD statement.

filespec

Specifies the name of a file that contains the PowerHouse dictionary to be used for the current QUICK session.

Defaults: PHD (**MPE/iX, OpenVMS**) or phd.pdc (**UNIX, Windows**)

[TYPE PHD|PDC] (OpenVMS)

Specifies the default dictionary type. If the TYPE option is specified, it applies to all subsequent SET DICTIONARY statements in the session.

When searching for a dictionary, PowerHouse limits searches to the dictionary type specified by the TYPE option. If the TYPE option is not specified, PowerHouse searches first for a PHD dictionary, then a PDC dictionary.

Default: PHD

DOWNSHIFT|UPSHIFT|NOSHIFT

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

These options allow dictionaries to be created with case-sensitive entity names. For system-wide access to mixed, lowercase or uppercase identifiers, you can specify the SHIFT option in the SYSTEM OPTION statement.

LIST|NOLIST [GENERATE] [LAYOUT] [PROCEDURES] [SQL] [TRANSACTION] [USE]

LIST and NOLIST control what is listed on your terminal.

Although LIST performs the opposite function of NOLIST, the two are not mutually exclusive unless they include identical options.

An entry of either SET LIST or SET NOLIST without any options sets the default.

Default: LIST GENERATE LAYOUT USE; NOLIST PROCEDURES

GENERATE

Lists (if LIST was specified) or doesn't list (if NOLIST was specified) the results of the GENERATE statement to the list device.

LAYOUT

Lists (if LIST was specified) or doesn't list (if NOLIST was specified) the sample screen layout to the list device.

PROCEDURES

Lists (if LIST was specified) or doesn't list (if NOLIST was specified) the procedures created by the BUILD statement to the list device.

SQL

Controls the listing of SQL statements. It shows the SQL requests sent from PowerHouse to the database, including the effects of any substitutions.

Default: SET NOLIST SQL

TRANSACTION

Displays the transaction model used by the screen, all of the transactions defined in a screen, and all of the file and transaction associations. The transaction associations include the isolation levels for each database.

Default: SET NOLIST TRANSACTION

USE

Lists (if LIST was specified) or doesn't list (if NOLIST was specified) the source statements contained in a USE file.

MENUKEYS|NOMENUKEYS

MENUKEYS assigns default short-cut menu keys to Action bar and drop-down menu items that have labels. PowerHouse scans each label, and assigns a menu key to the first alphanumeric character (7-bit ASCII) that isn't assigned to another menu key in the Action bar or in the same drop-down menu. If a unique character cannot be found, then PowerHouse issues the following warning message:

```
*W* A non-unique MENUKEYS has been assigned to this ACTIONMENU or MENUITEM.
```

You can override any of the defaults using the MENUKEY or NOMENUKEY options of the ACTIONMENU and MENUITEM statements. Menu keys are highlighted with an underline by default. You can change the highlighting by using the HILITE statement.

Menu keys may enhance the ease-of-use of screens with Action bars and/or drop-down menus. Menu keys are indicated by a highlighted character (underlined by default) in the label of an Action bar or drop-down menu item; when the key is pressed, the menu item is invoked. This offers an alternative to using cursor keys or a mouse to select a menu item.

NOMENUKEYS specifies that no default menu keys are assigned to Action bar and drop-down menu items. However, you can still add menu keys where required by using the MENUKEYS option of the ACTIONMENU statement and the MENUITEM statement.

Default: NOMENUKEYS

NESTING n

Sets the number of entries (n) in a table used by the parser to track control structures. This option is needed only when QDESIGN issues a message that the tables need more space.

Default: 50 entries

PANEL|NOPANEL

Specifies whether or not generated procedure code contains the options and control structures required to process Data fields in Panel mode.

SCREEN statements with the PANEL or NOPANEL options override SET PANEL and SET NOPANEL statements.

Default: NOPANEL

Limit: The PANEL|NOPANEL option is valid only before the SCREEN statement or after the BUILD statement.

PANEL

Generates the APPEND, ENTRY, MODIFY, PATH, and SELECT procedures, and includes BLOCK TRANSFER control structures in each. BLOCK EACH and BLOCK ALL options on CLUSTER statements affect the way the BLOCK TRANSFER control structures are generated in the APPEND and ENTRY procedures.

NOPANEL

Generates only the APPEND, ENTRY, and PATH procedures, and doesn't include BLOCK TRANSFER control structures in them. BLOCK EACH and BLOCK ALL options on CLUSTER statements are ignored.

Generated FIELD statements, CLUSTER statements, and procedure code do not contain the options and control structures required to process Data fields in Panel mode.

PRINT|NOPRINT

PRINT sends the source listing to the default printer; NOPRINT doesn't.

MPE/iX:	The source listing is sent to the designated file SYSPRINT.
OpenVMS:	The source listing is sent to the designated file SYSPRINT. If this logical is not defined then the system default is used. SET PRINT is ignored in batch jobs.

UNIX, Windows:	The default printer is obtained from the value of the environment variable PH_PRINTER. If this variable is not set, the system default is used.

Default: NOPRINT

SAVE CLEAR

Clears QDESIGN's source statement save file at the point where the SET SAVE CLEAR statement is entered.

VERIFY|NOVERIFY [DELETE] [ERRORS] [SUMMARY]

VERIFY enables requests for authorization to proceed with processing. NOVERIFY disables requests for authorization to proceed with processing.

Although VERIFY performs the opposite function of NOVERIFY, they are not mutually exclusive unless they include identical options. An entry of either the SET VERIFY or SET NOVERIFY statement without any options sets the VERIFY or NOVERIFY options to DELETE ERRORS SUMMARY.

DELETE

If NOVERIFY was specified, authorization is not requested before deleting an existing file.

MPE/iX, UNIX, Windows:	If VERIFY was specified, authorization is requested to delete an existing file and replace it with a new file of the same name generated by a BUILD or SAVE statement.
OpenVMS:	If VERIFY was specified, authorization is requested to create a new version of an existing file generated by a BUILD or SAVE statement.

ERRORS

Issues (if VERIFY was specified) or doesn't issue (if NOVERIFY was specified) an "expected" list when errors are encountered in a file processed by a USE statement, and waits for a carriage return to continue processing or a user break to end processing.

SUMMARY

Requests (if VERIFY was specified) or doesn't request (if NOVERIFY was specified) authorization to proceed when a summary of errors and warnings is reported following the BUILD statement at the end of the screen design.

Default: VERIFY DELETE SUMMARY, NOVERIFY ERRORS

WARNINGS|NOWARNINGS

WARNINGS issues warnings as required; NOWARNINGS suppresses warnings.

Default: WARNINGS

WRAPAROUND|NOWRAPAROUND

Determines whether or not QDESIGN should issue a carriage return and line feed at the end of each line on a screen.

Use WRAPAROUND for terminals that automatically issue a carriage return and line feed after column 80 when displaying the sample layout screen. Use NOWRAPAROUND for terminals that don't automatically issue a carriage return and line feed after column 80 when displaying the sample layout screen.

Default: WRAPAROUND

Discussion

The SET statement overrides the default options that are normally in effect during a QDESIGN session.

Most SET options remain in effect from one screen design to another. The SET options are used by QDESIGN and are not saved in a compiled screen.

Setting Options in the Resource File

Some options can be specified in the resource file. The options on the SET statement override those in the resource file. For more information, see Chapter 1, "Running PowerHouse", in the *PowerHouse Rules* book.

Example

The following example shows the use of the SET statement to specify that only procedural code be listed. SET NOLIST GENERATE LAYOUT suppresses the generation of FIELD statements and the screen layout. SET LIST PROCEDURES produces a listing of the procedural code generated by the BUILD statement.

```
> SET NOLIST GENERATE LAYOUT
> SET LIST PROCEDURES
>
> SCREEN ADMITTED
>
> FILE PATIENTS PRIMARY
> FILE HOSPITAL SECONDARY
>
> GENERATE
> BUILD
HOSPITAL accessed via MEDINSNO.
>
> PROCEDURE ENTRY
> BEGIN
> BLOCK TRANSFER
>   BEGIN
>     ACCEPT MEDINSNO OF PATIENTS
>     ACCEPT LASTNAME OF PATIENTS
.
.
.
```

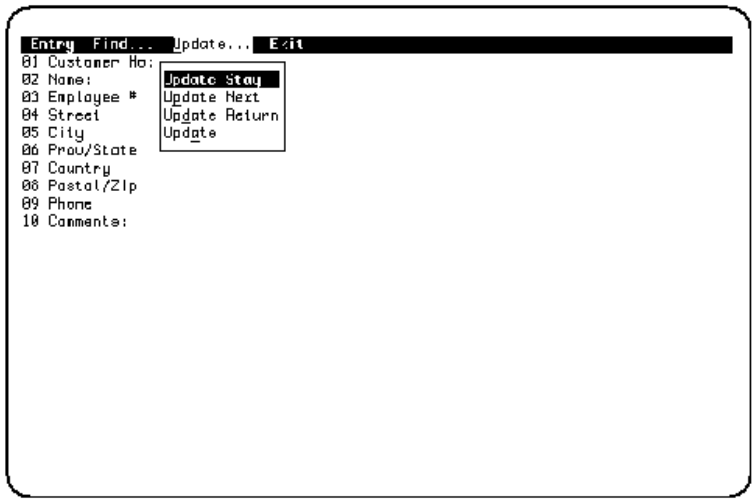
Assigning Default Menu Keys

In the following example, default menu keys are assigned to each Action bar and drop-down menu item. The underlined letter in each menu item is the menu key for that menu item:

```
> SET MENUKEYS
> SCREEN CUSTCAP ACTIONBAR
> ACTIONMENU LABEL "ENTRY" ACTION ENTRY
> ACTIONMENU LABEL "FIND..."
>   MENUITEM LABEL "FIND" ACTION FIND
>   MENUITEM LABEL "FIND NEXT" ACTION NEXT DATA
> ACTIONMENU LABEL "UPDATE..."
>   MENUITEM LABEL "UPDATE STAY" ACTION UPDATE STAY
>   MENUITEM LABEL "UPDATE NEXT" ACTION UPDATE NEXT
>   MENUITEM LABEL "UPDATE RETURN" ACTION UPDATE RETURN
>   MENUITEM LABEL "UPDATE" ACTION UPDATE
> ACTIONMENU LABEL "EXIT" ACTION RETURN
>
> FILE CUSTOMERS
> FIELD CUSTOMER OF CUSTOMERS REQUIRED &
>    NOCHANGE LOOKUP NOTON CUSTOMERS
> FIELD CUSTNAME OF CUSTOMERS REQUIRED NOCHANGE
> FIELD EMPLOYNO OF CUSTOMERS REQUIRED NOCHANGE
> FIELD STREET OF CUSTOMERS
> FIELD CITY OF CUSTOMERS
```

```
> FIELD PROVSTATE OF CUSTOMERS
> FIELD COUNTRY OF CUSTOMERS
> FIELD POSTALZIP OF CUSTOMERS
> FIELD PHONE OF CUSTOMERS
> FIELD REMARKS OF CUSTOMERS
> GO
```

The resulting screen looks like this:



Displaying Transactions

The following example shows the transactions used by the screen STOCKMT:

```
> SET NOLIST GENERATE LAYOUT PROCEDURE
> SET LIST TRANSACTION
> SCREEN STOCKMT TRANSACTION MODEL CONCURRENCY
> FILE STOCKS PRIMARY
> GENERATE
> BUILD
Transaction model Concurrency.
LOGICAL transaction QUERY, Special Inherited
    Commit on MODE
    Used by STOCKS for QUERY
LOGICAL transaction UPDATE, Special Inherited
    Commit on UPDATE
    Used by STOCKS for PROCESS UPDATE
    CONSISTENCY
```

SHOW

Displays information about available record-structures and/or items in the data dictionary.

Syntax

SHOW DATABASES|FILES|ITEMS

DATABASES

Lists the tables or views from databases that are declared with either FILE or DATABASE statements in PDL.

FILES

Lists the names of record structures, tables, or views from files and databases that are declared with FILE or DATABASE statements in PDL.

ITEMS

Displays the names of all items accessed with FILE or CURSOR statements including the temporary and defined items. The files available include

AUDIT	DELETE	DESIGNER
DETAIL	MASTER	PRIMARY
REFERENCE	SECONDARY	

Indexed items are identified by asterisks (*). If there are indexes made up of multiple segments, only the first segment is identified with an asterisk.

Substructured items are identified by periods (.) up to the fourth substructured level; for items substructured at levels 5 to 15 (the maximum), the display format is .05 to .15. In this way, items substructured from the fifth to the fifteenth level are identified explicitly by number rather than by a nested format.

Discussion

The SHOW statement displays the names of record-structures and items in the data dictionary. In QDESIGN, picture information is displayed for temporary and defined items. Because the picture information isn't available until the user issues the FIELD statement, these pictures are based only on datatype information provided in the TEMPORARY or DEFINE statements themselves.

In QDESIGN, all files and items are displayed regardless of the security specified for those files and items in the data dictionary.

Redefinitions are identified by underscores (_).

The maximum picture size allowed in the SHOW ITEMS display is 15. Picture overflow is identified by three trailing periods; pictures that are over 15 characters in length are displayed up to the twelfth character followed by three trailing periods (...).

Example

The following example demonstrates how you can view the databases, files and items in your data dictionary. Substructures are preceded by periods (as with the items RECORDTYPE and ACCOUNTNUMBER).

```
> SET DICTIONARY ORDERENT
> SHOW DATABASES
BILLINGS                          IN EMPBASE
BILLINGS_AUDIT                    IN EMPBASE
BRANCHES                          IN EMPBASE
CUSTOMER                          IN EMPBASE
```

```
CUSTOMER_OLD                        IN EMPBASE
CUSTOMER_PROJECT                    IN EMPBASE
DIVISIONS                           IN EMPBASE
EMPLOYEES                           IN EMPBASE
.
.
.
> DECLARE CURSOR_ONE CURSOR FOR &
>   SELECT * FROM BRANCHES
> SCREEN BRANCHC
> CURSOR CURSOR_ONE PRIMARY KEY BRANCH
> SHOW ITEMS
                          INPUT OUTPUT
CURSOR_ONE                TYPE SCALE SCALE DEC PICTURE
  BRANCHES.BRANCH         CHARX(2)
  BRANCHES.BRANCH_NAME    CHAR                X(20)
  BRANCHES.BRANCH_MANAGER VARCHAR             X(20)

> SET DICTIONARY ORDERENT
> SHOW FILES
ADJUSTMENTS
CUSTOMERS
SUPPLIERS
INVENTORYAUDIT
INVOICEDETAIL
INVOICEMASTER
NEXTCODE
ORDERDETAIL
ORDERMASTER
PARTS
PARTNOTES
PARTSUPPLIERS
SALESYTD
> SCREEN MODCUST PANEL
> FILE CUSTOMERS PRIMARY
>
> SHOW ITEMS
CUSTOMERS          TYPE   SCALE   SCALE  DEC  PICTURE
 *CUSTOMERKEY      CHAR                       X(5)
  .RECORDTYPE      CHAR                       X(1)
  .ACCOUNTNUMBER   NUM                        "^^^^"
  CUSTOMERNAME     CHAR                       X(20)
```

SKIP

Skips lines to a specific line or to an alignment group.

Syntax

SKIP [n|TO [LINE] m|TO GROUP g]

n

Sets the number of lines (n) to be skipped. The SKIP statement positions the cursor at the beginning of the next line, and then skips the specified number of lines.

TO [LINE] m

Skips to the specified line (m) relative to the first line of the screen.

TO GROUP g

Skips to the specified alignment group (g).

Discussion

The SKIP statement is part of the layout section of the screen design. It is related to the first CLUSTER, COMMAND, FIELD, SUBSCREEN, or TITLE statement that follows it. The SKIP statement positions the object of the statement that follows it at a line number or an alignment group or both. The SKIP statement and the SKIP n option position objects relative to the current screen position.

How the SKIP Statement Works

The current screen position is normally the highest position (left-to-right, top-to-bottom) occupied on the screen. However, the SKIP TO LINE (m) option resets the current screen position and positions the object relative to the first line of the screen. The SKIP statement with no options specified skips to the next line. The SKIP statement does nothing if it is placed so that it relates to the beginning of a screen line.

Example

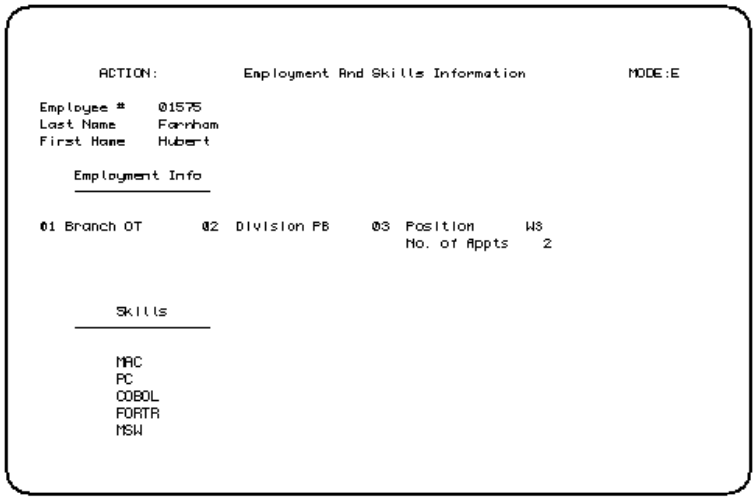
The following example demonstrates how to use the SKIP statement to create a screen with a format that represents a hard copy form. In this example:

- The SKIP 1 statement skips 1 line to position fields on line 3.
- The SKIP TO 7 statement positions the title on line 7 and omits the keyword LINE.
- The SKIP 2 statement leaves two blank lines on the screen after the first group of fields.
- The SKIP TO GROUP 3 statement positions NOOFAPPTS within the third alignment group under the field POSITION.

```
> SCREEN EMPDTL &
>  MODE AT 1,70 FIELDMARK
>
> FILE EMPLOY1 PRIMARY
> FILE SKILLS DETAIL OCCURS 5
>
> TITLE "Employment And Skills Information" AT ,25
>
> SKIP 1
>
> ALIGN (,1,15)
> FIELD EMPLOYEE OF EMPLOY1 REQUIRED NOCHANGE &
> LOOKUP NOTON EMPLOY1
> FIELD LASTNAME OF EMPLOY1 REQUIRED NOCHANGE
> FIELD FIRSTNAME OF EMPLOY1
>
```

```
> SKIP TO 7
>
> TITLE "Employment Info" AT ,5
> DRAW FROM 8,5 TO 8,20
>
> SKIP 2
>
> ALIGN (1,4,11) (20,24,33) (40,44,58)
> FIELD BRANCH OF EMPLOY1 REQUIRED NOCHANGE
> FIELD DIVISION OF EMPLOY1 REQUIRED NOCHANGE
> FIELD POSITION OF EMPLOY1 REQUIRED NOCHANGE
>
> SKIP TO GROUP 3
>
> FIELD NOOFAPPTS OF EMPLOY1 ID SAME
>
> SKIP TO LINE 15
> TITLE "Skills" AT ,10
> DRAW FROM 16,5 TO 16,20
>
> SKIP 2
>
> ALIGN (,,10)
> CLUSTER OCCURS WITH SKILLS
> FIELD SKILL OF SKILLS
> CLUSTER
>
```

The resulting screen looks like this:



SUBSCREEN

Invokes a lower-level screen.

Syntax

SUBSCREEN filespec|{ITEM item} [option]...

filespec

Names a file containing the compiled screen to be called. Normally QUICK only opens the file the first time the statement is processed.

ITEM item

Indicates that the subscreen's file specification is defined in an item.

Limit: You cannot call subscreens named ITEM unless you precede the file specification with a percent sign (%).

item

Names the item in which the subscreen's file specification is defined. The item can be either a record item, a temporary item, or a defined item.

Limit: The item type must be CHARACTER or VARCHAR.

Options

SUBSCREEN Options

AUTO	CLEAR ALL|SCREEN	HIDDEN
ID|NOID	[ENTRY] IF	INPUT B|C|SAME
LABEL|NOLABEL	MARK|NOMARK	MODE
ON ERROR	PASSING	REFRESH
WINDOW WIDTH		

AUTO

Automatically invokes the named screen when the standard entry sequence reaches this statement, provided the ENTRY procedure was generated by QDESIGN.

CLEAR ALL|SCREEN|{[LINES] n [TO m]}

Clears an area of the terminal memory before the screen is called. Cleared lines are refreshed automatically when the current screen becomes active again and QUICK is ready to prompt the user.

ALL

Clears the entire terminal memory.

SCREEN

Clears the area taken by the current calling screen.

[LINES] n [TO m]

Clears the area between and including lines n to m numbering from the first line of terminal memory. LINE n alone clears only line n. LINE is for documentation only.

HIDDEN

Suppresses the screen ID display, but the user can reference the field using ID-numbers.

ID SAME
ID n [AT [row],column]
ID NEXT [AT [row],column]
ID AT [row],column]
NOID

Declares the ID-number for the subscreen and its position.

ID SAME

Instructs QDESIGN to omit the ID-number on the subscreen. To access the subscreen, use the ID-number of the previous field.

ID n

Explicitly specifies an ID-number.

Limit: 1 to 99

ID NEXT

Uses the next ID-number in sequence.

AT [row],column

Positions the first digit of the ID-number at the specified row and column relative to the starting line of the screen. If the row is missing, the current line is assumed.

NOID

States that no ID-number is assigned to this subscreen and the subscreen can't be referenced from the Action field.

[ENTRY] IF condition

Invokes this subscreen in the standard entry sequence only if this condition is satisfied, provided the ENTRY procedure was generated by QDESIGN. QDESIGN generates an identical IF control structure in the default ENTRY procedure. If the condition is satisfied, the **auto** program parameter is assumed.

For PANEL screens, subscreens aren't invoked until the entire block is transmitted to QUICK for processing. The RUN SCREEN verb in the ENTRY procedure is processed only after an entry has been made for each ACCEPT verb in the ENTRY procedure, and the QUICK screen user has pressed [Enter].

Limit: The IF option is evaluated only during the standard ENTRY sequence; otherwise, it is ignored.

INPUT B|C|SAME (MPE/iX)

Specifies the input mode the subscreen is to be in when it appears. The terminal is always returned to the original mode on return from the called screen.

B

Starts the subscreen in Block mode if the subscreen can be run in Block mode.

C

Starts the subscreen in character mode.

SAME

Starts the subscreen in the same input mode as the calling screen.

LABEL [string] [AT [row],column]|NOLABEL

Declares the label and its position.

string AT [row],column

Indicates the subscreen label, and, optionally, the position of the label on the screen.

The AT option positions the first character of the label at the specified row and column relative to the starting line of the screen. If the row isn't specified, the current line is assumed.

Default: The subscreen name.

NOLABEL

Specifies that no label appears for the subscreen.

MARK|NOMARK

MARK enables fieldmarking for a subscreen with an ID.

NOMARK disables the default fieldmarking for a subscreen with an ID when fieldmarking is enabled.

MODE E|F|S|NULL|SAME|GHOST

Specifies which mode the subscreen is in when it appears.

E|F|S

Indicates that the subscreen is in one of the following modes when it first appears:

- Entry mode (E)
- Find mode (F)
- Select mode (S)

NULL

Indicates no mode. QUICK prompts for a mode at the Action field when the screen appears.

SAME

Indicates that the subscreen is in the same mode as the current screen when the screen named in the SUBSCREEN statement is invoked.

GHOST

Indicates that the subscreen being called is a "ghost" screen. This causes QUICK to skip the refreshing of the calling screen when returning from the subscreen call.

When QUICK runs a subscreen with the GHOST option, the default mode is used.

The GHOST option should only be used to call a ghost screen; that is, a screen that does all its work in the INITIALIZE procedure with no terminal output. If the GHOST option is used on a subscreen that is not a ghost screen, results will be unpredictable and screen corruption may occur.

Default: NULL. If the subscreen is invoked during the standard Entry sequence, the default is E (Entry mode).

ON ERROR CONTINUE|TERMINATE

Determines whether processing on the calling screen continues or terminates if an error which the user had no opportunity to correct, occurs on the subscreen. This option only has an effect when an error on a subscreen is not displayed to the user on that subscreen.

CONTINUE

The execution of the calling screen continues, regardless of the fact that an error which the user had no opportunity to correct, occurred on the subscreen.

TERMINATE

When an error which the user had no opportunity to correct, occurs on the subscreen, processing on the calling screen terminates as if the SUBSCREEN statement failed.

Default: TERMINATE

Prior to 7.33.C (**UNIX**), 7.10E1 (**OpenVMS**) and 8.09 (**MPE/iX**), the behavior was equivalent to CONTINUE.

PASSING record-structure|item [,record-structure|item]...

Specifies which of the current screen's existing record-structures, defined items, and temporary items are passed to the subscreen. Individual record items cannot be passed to the subscreen. Entity names in the list must be separated by commas.

Items must match, on the basis of identical item attributes, with the items named in the RECEIVING option of the lower-level SCREEN statement. The names themselves may differ.

Passing a defined item allows you to use a higher-level screen expression on the lower-level screen without having to repeat the expression. No value is passed. Defined items on higher-level screens can only be passed to defined items on lower-level screens; temporary items on higher-level screens can only be passed to temporary items on lower-level screens.

Limit: A combined maximum of 16 records and items.

REFRESH ALL|SCREEN|{[LINES] n [TO m]}

Clears and rewrites an area of the terminal memory when the screen becomes active again and QUICK is ready to prompt the user. REFRESH options are performed before, and in addition to, an automatic refresh from any CLEAR option.

ALL

Clears and rewrites the entire terminal memory.

SCREEN

Clears and rewrites the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears and rewrites the area between and including the lines n to m numbering from the first line of the terminal memory. LINES n alone refreshes only line n. LINES is for documentation only.

WINDOW WIDTH CONSTANT|DEFAULT WHEN CALLING|RETURNING

Overrides the default behavior and sets the terminal to the correct screen width (80 or 132 columns).

WHEN CALLING

Keeps the current screen width of the screen when calling a subscreen.

WHEN RETURNING

Keeps the current screen width of the subscreen when returning from that subscreen.

Discussion

The SUBSCREEN statement is part of the layout section of the screen design. It allows the screen user to invoke a related, lower-level screen.

Limit: The combined maximum number of SUBSCREEN statements, RUN SCREEN and RUN COMMAND verbs is 256 per screen; if you exceed this limit, QDESIGN issues an error message.

Dynamic Screen Calls in QDESIGN and QUICK

You can call subscreens by using the SUBSCREEN statement, the THREAD statement, the RUN SCREEN verb, or the RUN THREAD verb. Each of these statements/verbs works in either of two ways:

- You provide the subscreen's file specification directly in the statement or verb syntax. Thus, the same subscreen is called every time the statement or verb is executed.
- You use the ITEM keyword to point the statement or verb to an item that contains the subscreen's file specification. Thus, a different subscreen might be called each time depending on what file specification the item contains at the moment of execution. If you use the PASSING option, the parameters passed must be in the same order for each screen referencing the item.

The latter method is known as dynamic screen calling because you control which subscreens are called based on run-time variables. This allows you to build context-sensitive applications in which different users may see different screens based on these variables.

Examples

The following example creates a MENU screen from which several "part maintenance" subscreens can be accessed. In this example:

- MODE E invokes both the subscreens ADDPART and ADDVAR in Entry mode.
- MODE F invokes the subscreens MODPART and PARTLIST in Find mode.

```
> SCREEN PARTMAIN MENU
>
> TITLE "Parts Maintenance" CENTERED AT 4,1
>
> ALIGN (20,25,)
> SKIP TO LINE 8
>
> SUBSCREEN ADDPART   &
>   MODE E  &
>   LABEL "Add a New Part"
>
> SUBSCREEN ADDVAR   &
>   MODE E  &
>   LABEL "Add a New Part Variant"
>
> SUBSCREEN MODPART  &
>   MODE F &
>   LABEL "Change or Delete a Part"
>
> SKIP 1
>
> SUBSCREEN PARTLIST &
>   MODE F &
>   LABEL "Check Inventory Level for a Part"
>
> SKIP 1
>
> SUBSCREEN PARTRPT  &
>   ID 20  &
>   LABEL "Generate Parts Summary Reports"
>
> SKIP 2
>
> BUILD
```

Passing Records Between Subscreens

The following EMPLOYEES screen invokes a subscreen that lists employee skills. In this example

- PASSING indicates that EMPLOYEES is passed to the SKILLS screen.
- MODE SAME indicates that SKILLS is invoked in the same mode as that of EMPLOYEES.

```
> SCREEN STAFF
```

```
> FILE EMPLOYEES
>
> TITLE "S T A F F   S C R E E N" at 1,30
> SKIP 1
> FIELD EMPLOYEENUMBER OF EMPLOYEES &
>   REQUIRED NOCHANGE &
>   LOOKUP NOTON EMPLOYEES &
>   LABEL "EMPLOYEE NUMBER"
> ALIGN (1,4,21) (,,35)
> FIELD FIRSTNAME OF EMPLOYEES LABEL "NAME"
> FIELD LASTNAME OF EMPLOYEES REQUIRED
> ALIGN (16,20,45)
> SKIP 1
>
> SUBSCREEN SKILLS &
>  LABEL "S K I L L S   S C R E E N" &
>  PASSING EMPLOYEES MODE SAME
```

The lower-level screen must include a RECEIVING statement to allow the record-structure EMPLOYEES to be passed to it. The EMPLOYEES record-structure must also be declared as the MASTER file in the lower-level screen.

```
> SCREEN SKILLS RECEIVING EMPLOYEES
> FILE EMPLOYEES MASTER
> FILE SKILLS PRIMARY OCCURS 6
.
.
.
> BUILD
```

In the following example, the conditional-expression in item SUB1 tests the user's application security class, resulting in a file specification for either a restricted-access screen or a full-access screen, as appropriate:

```
> SCREEN STAFF
>
> DEFINE SUB1 CHAR*31 =    &
>    "Employee_All" IF MATCHUSER ("MANAGER")   &
>    ELSE "Employee_Restricted"
.
.
.
> SUBSCREEN ITEM SUB1 LABEL "Employee_Info"
```

TARGET

Calculates the record number in a direct file for storing a newly-created record.

Syntax

TARGET numeric-expression

numeric-expression

A numeric expression that, when evaluated, results in a record number.

Discussion

The TARGET statement is part of the data section of your screen design. The first record number in a direct file is 0 (**MPE/iX, UNIX, Windows**) or 1 (**OpenVMS**).

The TARGET statement causes an AT option to be generated on the PUT verb from the file in the standard UPDATE procedure.

The TARGET statement must immediately follow the FILE statement of the direct file to which it refers. TARGET is relevant only if the record status is new. If the statement is missing, new records are appended to the end of the file.

TEMPORARY

Creates a temporary item that is not defined in the data dictionary.

Syntax

TEMPORARY name [type[*n] [type-option]] [option]...

Limit: A maximum of 1023 defined and temporary items can be declared in a screen design.

name

Names the temporary item.

Limit: Must begin with a letter and can't exceed 64 characters. Fields relating to temporary items can't be used to enter selection criteria in Select mode.

type

Establishes the physical format of the temporary item.

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

Default: NUMERIC

*n

Specifies the number of characters or digits that can be stored in the temporary item.

type-option

Indicates the set of options that further characterize the item datatype.

The type-options are CENTURY INCLUDED|EXCLUDED, NUMERIC, SIGNED|UNSIGNED, and SIZE.

CENTURY INCLUDED|EXCLUDED

Indicates whether or not the date will contain a century prefix.

Limit: Valid for temporary items of type DATE, JDATE and PHDATE.

Default: For DATE items, the default is determined by the SYSTEM OPTIONS statement in the data dictionary. For PHDATE and JDATE items, the default is CENTURY EXCLUDED.

NUMERIC

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

Limit: Valid only for ZONED datatypes.

SIGNED|UNSIGNED

Indicates that the datatypes INTEGER, PACKED, and ZONED are SIGNED or UNSIGNED. When the SIGNED option is used with INTEGER, negative values can be stored. A datatype INTEGER with the UNSIGNED option can't store negative values. The datatypes PACKED and ZONED can store positive or negative numbers, whether or not the SIGNED or UNSIGNED option is specified.

Limit: Valid only for INTEGER, PACKED, and ZONED datatypes.

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

SIZE m [BYTES]

Specifies a storage size (m) in bytes.

Use the SIZE m BYTES option when the default size isn't the size that's required for the item.

Options

The TEMPORARY options are INITIAL, OCCURS, RESET, and SUM.

INITIAL conditional-expression

Assigns the temporary item a value that is calculated by this expression when the temporary item is initialized.

For information about conditional-expressions in PowerHouse, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Limit: This option has no effect if the item was received from a higher-level screen, unless the MYVIEW option of the FILE statement is in effect.

OCCURS n [TIMES]
OCCURS WITH [ITEM] item|[FILE] record-structure

Repeats the temporary item on the screen.

n [TIMES]

Repeats the temporary item the specified number of times (n) on the screen.

Limit: 1 to 255

WITH [ITEM] item|[FILE] record-structure

Repeats the temporary item as many times as the named data record or item repeats on the screen.

Limit: The RESET option is not valid with the OCCURS WITH option.

RESET [AT MODE|STARTUP]

Resets the temporary item to initial values. By default, a temporary item is initialized when record buffers are initialized.

Limit: A RESET option on a higher-level screen has no effect on a temporary item that's passed to a lower-level screen while the item is being used on the lower-level screen.

AT MODE

Causes initialization when Entry mode (E) or Find mode (F) or Select mode (S) initialization occurs.

AT STARTUP

Causes initialization only when the screen is entered from a higher-level screen.

Default: If you do not specify AT MODE or AT STARTUP, a temporary item is initialized when primary record buffers are initialized.

Limit: The RESET AT STARTUP and RESET AT MODE option are not valid with the OCCURS WITH option. They are valid with the OCCURS n option. A RESET option on a higher-level screen has no effect on a temporary item that’s passed to a lower-level screen while the item is being used on the lower-level screen.

SUM [NEGATIVE] [INTO] item2 [WHEN POSITIVE|NEGATIVE] [,[NEGATIVE] [INTO] item3 [WHEN POSITIVE|NEGATIVE]]...

Adds or subtracts all values entered for the temporary item and maintains the total in the one or more items. This sum is automatically

- reduced when values are deleted
- incremented when values are entered
- adjusted when the value is changed

NEGATIVE

Reverses the incrementing and reducing activities.

INTO

Used only for documentation.

item2, item3, ...

Specifies the items into which the summing is performed.

WHEN POSITIVE|NEGATIVE

Specifies that summing is performed only if the values are positive or only if the values are negative.

Discussion

The TEMPORARY statement is part of the data section of your screen design. It creates and defines a temporary item that doesn't exist in the data dictionary and applies only to this screen (although it can be passed to lower-level screens). A TEMPORARY statement can be placed anywhere in the data section, as long as it doesn't reference items in options that are not yet declared.

Initialization of Temporary Items

Temporary items are initialized to zero or blank if no initial values are provided. By default, temporary item buffers associated with the screen are initialized when the record buffers are marked for initialization, except when a data record retrieval fails. A temporary item is associated with a record-structure in a PRIMARY file unless it is specifically related to a data record or item using the OCCURS WITH option.

Passing Temporary Items to Subscreens

When a temporary item is passed to a lower-level screen, the RESET options specified when the temporary item was originally declared have no effect. If nothing is specified when the temporary item is received, it is never reset by QUICK on the lower-level screen. Any RESET option specified on the lower-level screen acts as specified.

Resetting Temporary Items

Numeric and date temporary items are initialized to zeroes and character temporary items to blanks unless the INITIAL option is used. Temporary items are initialized as follows.

Option	Timing
Not Specified or RESET	Initialization occurs when the primary record buffers are marked for initialization. This occurs during the Entry initialization phase, Find initialization phase and the retrieval initialization phase. During the Retrieval initialization phase, if the data record retrieval fails, the temporary item is not reset. A temporary item is associated with a record structure in a PRIMARY file unless it is specifically related to a data record or item using the OCCURS WITH option.
RESET AT MODE	Initialization only occurs when the user explicitly chooses Entry, Find, or Select mode.
RESET AT STARTUP	Initialization only occurs at screen startup, prior to the optional INITIALIZE procedure.

Example

The following screen uses a temporary item to store the total value of all billings charged to a project for a given employee:

```
> SCREEN PROJBILL
>
> TEMPORARY TOTBILLINGS NUMERIC *8
>
> FILE PROJECTS PRIMARY
> FILE BILLINGS DETAIL OCCURS 6
Item PROJNO initialized (fixed) to PROJNO of PROJECTS
>   ITEM BILLING SUM INTO TOTBILLINGS
> FILE BILLINGS ALIAS BILLDES DESIGNER OPEN 1
> FIELD TOTBILLINGS
>
> SKIP 1
>
> FIELD PROJNO OF PROJECTS
> FIELD PROJNAME OF PROJECTS
> FIELD PROJMGR OF PROJECTS
> FIELD PROJBUDG OF PROJECTS
>
> SKIP 1
>
> TITLE "Employee" AT ,4
> TITLE "Month" AT ,15
> TITLE "Billing" AT ,25
> TITLE "Employee" AT ,44
> TITLE "Month" AT ,55
> TITLE "Billing" AT ,65
>
> ALIGN (1,,4) (,,15) (,,25)
> SKIP 1
>
> CLUSTER OCCURS WITH BILLINGS FOR 1,39
>   FIELD EMPNUM OF BILLINGS &
>     REQUIRED  &
>     NOCHANGE
>   FIELD MONTH OF BILLINGS
>   FIELD BILLING OF BILLINGS
> CLUSTER
>
> PROCEDURE POSTFIND
>   BEGIN
>     WHILE RETRIEVING BILLDES &
>       VIA PROJNO &
>       USING PROJNO OF PROJECTS
>     LET TOTBILLINGS = TOTBILLINGS + BILLINGS &
>       OF BILLDES
>   END
> BUILD
```

THREAD

Specifies a screen thread.

Syntax

THREAD filespec|{ITEM item} [option]...

filespec

Names the root screen of a thread.

ITEM item

Indicates that the root screen's file specification is defined in an item.

Limit: You cannot call root screens named ITEM unless you precede the file specification with a percent sign (%).

item

Names the item in which the root screen's file specification is defined. The item can be either a record item, a temporary item, or a defined item.

Limit: The item type must be CHARACTER or VARCHAR.

Options

THREAD Options		
AUTO	HIDDEN	ID|NOID
[ENTRY] IF	INPUT	LABEL|NOLABEL
MARK|NOMARK	MODE	SHARED
WINDOW WIDTH		

AUTO

Automatically invokes the named screen when the standard entry sequence reaches this statement, provided the ENTRY procedure was generated by QDESIGN.

HIDDEN

Suppresses the screen ID display, but the user can reference the field using the ID-number.

ID SAME
ID n [AT [row],column]
ID NEXT [AT [row],column]
NOID

Declares the ID-number for the thread and its position.

ID SAME

Instructs QDESIGN to omit the ID-number on the thread. To access the thread, use the ID-number of the previous field.

ID n

Explicitly specifies an ID-number.

Limit: 1 to 99

ID NEXT

Uses the next ID-number in sequence.

AT [row],column

Positions the first digit of the ID-number at the specified row and column relative to the starting line of the screen. If the row is missing, the current line is assumed.

NOID

States that no ID-number is assigned to this thread and the thread can't be referenced from the Action field.

[ENTRY] IF condition

Invokes this thread in the standard entry sequence only if this condition is satisfied, provided the ENTRY procedure was generated by QDESIGN. QDESIGN generates an identical IF control structure in the default ENTRY procedure. If the condition is satisfied, the **auto** program parameter is assumed.

For PANEL screens, threads aren't invoked until the entire block is transmitted to QUICK for processing. The RUN THREAD verb in the ENTRY procedure is processed only after an entry has been made for each ACCEPT verb in the ENTRY procedure, and the QUICK screen user has pressed [Enter].

Limit: The IF option is evaluated only during the standard ENTRY sequence; otherwise, it is ignored.

INPUT B|C (MPE/iX)

Specifies the input mode the thread is to be in when it appears. (Although the SAME sub-option is accepted syntactically, it has no meaning in the context of threads.)

B

Starts the thread in Block mode if the thread can be run in Block mode

C

Starts the thread in character mode.

Default: Unless overridden by other means, such as in QKGO or the SCREEN statement, when a thread is first opened, it will be opened in character mode. When toggling back to a thread, it will be in the same input mode that it was in when it was left.

LABEL [string] [AT [row],column]|NOLABEL

Declares the label and its position.

string AT [row],column

Indicates the thread label, and, optionally, the position of the label on the screen.

The AT option positions the first character of the label at the specified row and column relative to the starting line of the screen. If the row isn't specified, the current line is assumed.

Default string: The thread name. If you are using the ITEM keyword, no default label is assigned. It is recommended that you include the string whenever you use the ITEM keyword syntax, and whenever the file specification includes a path.

NOLABEL

Specifies that no label appears for the thread.

MARK|NOMARK

MARK enables fieldmarking for a thread with an ID.

NOMARK disables the default fieldmarking for a thread with an ID when fieldmarking is enabled.

MODE E|F|S|NULL|SAME

Specifies which mode the thread is in when it appears.

E|F|S

Indicates that the thread is in one of the following modes when it first appears:

- Entry mode (E)
- Find mode (F)
- Select mode (S)

NULL

Indicates no mode. QUICK prompts for a mode at the Action field when the screen appears.

SAME

Indicates that the thread is in the same mode as the current screen when the screen named in the THREAD statement is invoked.

Default: NULL. If the thread is invoked during the standard Entry sequence, the default is E (Entry mode).

SHARED

If SHARED is specified and the thread already exists, then the client activates the existing thread and makes it current.

WINDOW WIDTH CONSTANT|DEFAULT WHEN CALLING

Overrides the default behavior and sets the terminal to the correct screen width (80 or 132 columns).

Discussion

By default, a maximum of three threads can be specified in a screen design. By using the "Max number of threads" execution-time parameter in QKGO, the limit can be increased to a maximum of seven threads.

Screen Threads

Multiple screen thread processing lets an application have more than one screen hierarchy active at one time. Screen threads can be used on terminals in the host environment; they also let designers and users take advantage of the Microsoft Windows Multiple Document Interface (MDI).

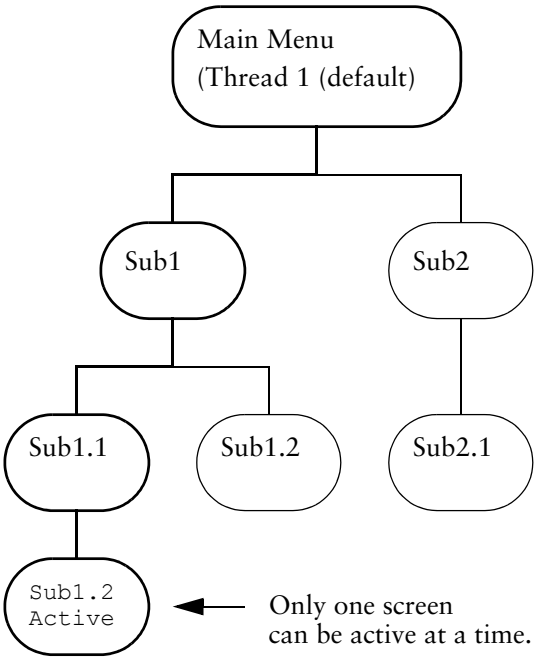
The QUICK Action command to toggle between threads is T, or the QKGO defined function key ([F1_T]) on HP terminals.

Multiple screen thread processing is available to PowerHouse applications and PowerHouse Windows RunTime applications.

Single Active Screen Hierarchy

If a screen hierarchy does not contain threading techniques, then only one screen can be active at a time. When a subscreen is the active screen, it takes control of processing. To return to the invoking screen, the user must exit from the subscreen.

In the following screen hierarchy, each called screen forces the previous screen to become inactive first. At any one time, there can only be one active screen in a screen hierarchy.



Multiple Active Screens Hierarchy

Using multiple screen threads, the designer can build applications that contain more than one active screen. The user can move between threads without closing screens. For example, the user could leave in the middle of entering data in one screen to find data in another and then return to the first screen.

By default, only three threads can be open simultaneously. This number can be increased to a maximum of seven in QKGO, by using the "Max. number of threads" execution-time parameter in QKGO. If the user tries to open a thread above the current allowable limit, a message is issued either asking to increase the limit, or indicating that the maximum (7) has been reached.

The limit on simultaneous open threads is not a limit on the number of THREAD statements that an application can contain. It limits the number of threads to which you can toggle.

Moving among Threads

To move among threads, use the Toggle Thread (T) Action command or the QKGO standard fixed function key. The maximum number of threads allowed in one session is controlled by the "Max number of threads" execution-time parameter in QKGO. Threads are kept track of in a circular list in the order that they are accessed.

Using the system of threads shown previously, the following is an example of a possible sequence of user actions and results:

Sequence of User Actions	Result
Start QUICK application: Main Menu as the first screen	Main Menu thread appears
Toggle Thread command (T)	Remain at Main Menu (since no other thread is open)
Select a menu option to start the Sub1 thread	Sub1 thread appears
Toggle Thread command (T)	Main Menu appears
Toggle Thread command (T)	Sub1 thread appears
Toggle Thread command (T)	Main Menu thread appears

Sequence of User Actions	Result
Select a menu option to start the Sub2 thread	Sub2 thread appears
Toggle Thread command (T)	Main Menu thread appears
Toggle Thread command (T)	Sub1 thread appears
Toggle Thread command (T)	Sub2 thread appears
Toggle Thread command (T)	Main Menu thread appears

Dynamic Screen Calls in QDESIGN and QUICK

You can call subscreens by using the SUBSCREEN statement, the THREAD statement, the RUN SCREEN verb, or the RUN THREAD verb. Each of these statements/verbs works in either of two ways:

- You provide the subscreen's file specification directly in the statement or verb syntax. Thus, the same subscreen is called every time the statement or verb is executed.
- You use the ITEM keyword to point the statement or verb to an item that contains the subscreen's file specification. Thus, a different subscreen might be called each time depending on what file specification the item contains at the moment of execution.

The latter method is known as dynamic screen calling because you control which subscreens are called based on run-time variables. This allows you to build context-sensitive applications in which different users may see different screens based on these variables.

Example

In the following example, the conditional-expression in item SUB1 tests the user's application security class, resulting in a file specification for either a restricted-access screen or a full-access screen, as appropriate:

```
> SCREEN STAFF
>
> DEFINE SUB1 CHAR*31 =    &
>     "Employee_All" IF MATCHUSER ("MANAGER")   &
>   ELSE "Employee_Restricted"
.
.
.
> THREAD ITEM SUB1 LABEL "Employee_Info"
```

TITLE

Positions text on the screen.

Syntax

TITLE string [AT [row], column] [CENTERED|CENTRED]

string

Indicates the text to be positioned.

AT [row],column

Positions the first character of the string at the specified row and column relative to the starting line of the screen. If the row is missing, the current line is assumed.

CENTERED|CENTRED

Centers the string on the current 80-column line.

Discussion

The TITLE statement is part of the layout section of a screen design. It positions character strings (titles, column headings, or other text) on the screen.

The following is a list of qualifications to the TITLE statement:

- If both the AT and CENTERED options are used, the string is centered on the column specified.
- If neither option is used, the first character of the string is positioned at the current alignment position for ID-numbers.
- If only the CENTERED option is used, and the current alignment position for ID-numbers is 1, the title is centered within the screen.
- If only the CENTERED option is used, and the current alignment position for ID-numbers is greater than 1, QUICK attempts to center the title over the ID-number position.
- If there isn't enough room to center the title to the left of the ID position, the title runs from the left of the current ID-number position (ignoring the CENTERED option), and a warning is issued.

Example

This menu screen displays each option in an individual box. The QUICK screen user's selection invokes one of the three subscreens that are accessible from this MENU screen. In this example:

- The CENTERED statement positions the title "Main Menu" in the center of the current line.
- The TITLE statements set up descriptive titles for the subscreens that are invoked on this screen.

```
> SCREEN MAINMENU MENU &
>    NOMODE &
>    ACTION LABEL &
>    "Enter a number to select an option: " &
>    AT 1,33
>
> SKIP TO LINE 4
>
> TITLE "Main Menu" CENTERED
>
> SKIP TO LINE 8
>
> DRAW 8,10 TO 15,30
> DRAW 8,50 TO 15,70
> DRAW 16,25 TO 22,55
```

```
>
> SUBSCREEN ORDMAIN &
> NOLABEL &
> ID 1 AT 10,19
>
> SUBSCREEN INVMAIN &
> NOLABEL &
> ID 2 AT 10,59
>
> SUBSCREEN MAINT &
> NOLABEL &
> ID 3 AT 18,39
>
> TITLE "Order" AT 11,18
> TITLE "Invoice" AT 11,57
> TITLE "Processing" AT 12,16
> TITLE "Processing" AT 12,56
> TITLE "System" AT 19,38
> TITLE "Maintenance" AT 20,35
>
> BUILD
```

TRANSACTION

Defines transactions used for relational files.

Syntax

TRANSACTION name INHERITED
[COMMIT ON automatic-commit-point | NOCOMMIT]

or

TRANSACTION name
[COMMIT ON automatic-commit-point | NOCOMMIT] [options]

name

A unique name used to identify logical transactions.

INHERITED

Indicates that the properties of a transaction have been defined on an ancestor screen and are known on the current screen only at execution-time. If the INHERITED option is used on the TRANSACTION statement, the only other options accepted are COMMIT ON or NOCOMMIT to indicate the automatic commit points to indicate the save points for the transaction on the current screen.

If no commit timing is specified on the TRANSACTION statement, QUICK will use the commit timing specified on the SCREEN statement, the default being COMMIT AT UPDATE. The INHERITED option cannot pass down the commit timing for the transaction from the previous screen.

The purpose of the INHERITED option is to allow QUICK to verify that transaction names referenced on a screen are valid. For each transaction referenced in a screen, the transaction must be one of the following:

- a PowerHouse predefined transaction (such as Query)
- defined on the screen using the TRANSACTION statement, or
- defined on an ancestor screen. You can optionally reference this transaction on the current screen using the INHERITED option on the TRANSACTION statement.

If at screen load time no ancestor screen has actually defined the transaction, the screen fails to load.

COMMIT ON automatic-commit-point

The COMMIT ON option is used to indicate the default points at which automatic commits are executed by QUICK.

Defaults: For query transactions: COMMIT ON MODE. For query transactions on subscreens that receive MASTER files: NOCOMMIT. For all other transactions, the default is taken from the SCREEN statement's commit point. For the SCREEN statement, the default is COMMIT ON UPDATE.

automatic-commit-point

Determines the points at which an automatic commit for the COMMIT ON option occurs during screen processing for Consistency and Concurrently Read_Write (Update) transactions.

The automatic commit points are UPDATE, NEXT PRIMARY, MODE, and EXIT.

UPDATE	This option is the default for the Update and Consistency transactions. Use the ON UPDATE option to ensure that related updates (for example, updates to primary and secondary data) are grouped together, but keep individual transactions relatively short. Locally active transactions are committed: when an Update action is completed (before the POSTUPDATE procedure); when the screen mode changes (before the PREENTRY and PATH procedures), when the user exits the screen (before and after the EXIT procedure).

NEXT PRIMARY	Use this option if you want to group all detail records (perhaps requiring several entry or display screens) together with primary and secondary records and treat them as a unit to be committed or rolled back. Locally active transactions are committed: when the user starts an entry sequence (before the PREENTRY procedure); when the user retrieves the next set of primary records (before the FIND procedure); and when the user exits the screen (before and after the EXIT procedure).
MODE	This option is the default for the Query transaction. Use the ON MODE option to ensure that changes to a series of existing records (for example, all employees in a certain branch or all tasks in a project) are committed or rolled back as a group. Locally active transactions are committed when: the screen mode changes (before the PREENTRY and PATH procedures); and on screen exit (before and after the EXIT procedure).
EXIT	Use this option when all activity done on a screen is to be treated as a single transaction. Locally active transactions are committed when the screen is exited (after the EXIT procedure).

NOCOMMIT

Indicates that QUICK does not generate automatic commit actions.

Options

TRANSACTION Options		
CONSTRAINTS	isolation-level	PRIORITY n
READ ONLY|READ WRITE	RESERVING FOR	WAIT|NOWAIT

CONSTRAINTS [type [,type]...] DEFERRED

Allows specific types of constraints to be deferred while others are checked immediately. The types of constraint checking that can be specified are:

- ALL
- CHECK
- REFERENTIAL
- UNIQUE

Default: ALL. If constraints are not deferred, they are checked every time a value is inserted, altered or deleted.

Limit: Valid only for ALLBASE/SQL transactions.

isolation-level

Lets the designer specify the degree to which this transaction is to be protected from the effects of concurrent transactions.

If a database doesn't support a specified isolation level, PowerHouse uses the next available higher isolation level without issuing a run-time warning. If a higher level is unavailable, PowerHouse uses the highest available lower level and issues a run-time warning.

The support available for the various isolation level options offered in QDESIGN depends on the support provided by the underlying database software.

For a discussion of isolation levels for specific databases, see the *PowerHouse and Relational Databases* book.

The isolation levels are listed below from lowest to highest:

READ UNCOMMITTED	A very low level of isolation that allows a transaction to see all changes made by other transactions, whether committed or not. Also known as a "dirty read."
READ COMMITTED	A transaction can read any data that has been committed by any transaction as of the time the read is done.
STABLE CURSOR	While a transaction has addressability to a record (that is, has just fetched it), no other transaction is allowed to change or delete it.
REPEATABLE READ	Any data that has been read during a transaction can be re-read at any point within that transaction with identical results.
PHANTOM PROTECTION	A transaction does not see new records, or "phantoms", that did not exist when the transaction started.
SERIALIZABLE	The results of the execution of a group of concurrent transactions must be the same as would be achieved executing those same transactions serially in some order.

Defaults: The default isolation level for transactions (other than inherited transactions or QUICK's predefined transactions) depends on the screen's transaction model. For Concurrency, the default isolation level is REPEATABLE READ for Update transactions and READ COMMITTED for Query transactions. For Consistency, the default isolation level is SERIALIZABLE.

PRIORITY n

Lets you specify the transaction priority for an ALLBASE/SQL transaction.

n

An integer in the range 0 to 255.

Limit: Valid for ALLBASE/SQL only.

READ ONLY|READ WRITE

Determines the type of activities that can be performed by this transaction. It also affects what type of transaction is started in the underlying database system.

Defaults: READ ONLY for Query. READ WRITE for all other transactions

RESERVING FOR [SHARED|PROTECTED|EXCLUSIVE] READ|WRITE {table [IN database]}...

Lets you specify database specific reserving on a table-by-table basis for a particular transaction.

SHARED|PROTECTED|EXCLUSIVE

SHARED lets others work with the same table(s). PROTECTED lets others read the table you are using; they cannot have write access. EXCLUSIVE prevents others from reading records from the table(s) included in your transaction.

Default: SHARED

READ|WRITE

READ lets you only read data from the reserved tables; WRITE lets you insert, update or delete data in the table.

Default: READ for read-only transactions. WRITE for read/write transactions. The defaults are determined from the READ ONLY|READ WRITE option.

Limit: Not supported for ALLBASE/SQL or Sybase databases.

WAIT|NOWAIT

Allows you to specify whether the transaction should wait in the case of lock or resource conflicts. The default is determined by the **dbwait** program parameter in effect at run time in QUICK (by default, DBWAIT).

If the option is WAIT, then a resource or lock conflict causes the transaction to wait, subject to other system parameter and time-out settings.

If the option is NOWAIT, then a resource or lock conflict causes the transaction to end with an error condition.

Discussion

There are three basic uses for the TRANSACTION statement:

- To declare and define the attributes of new designer-defined transactions
- To override the attributes of predefined transactions
- To indicate that a designer-defined transaction is INHERITED from a higher screen, but that the attributes of the transaction have been declared on another screen and are in effect when this screen is run. QDESIGN will accept the following:

```
> SCREEN PARENT
> TRANSACTION GLOBAL_READ READ ONLY...
> TRANSACTION GLOBAL_WRITE READ WRITE...
> TRANSACTION LOCAL READ WRITE
> SUBSCREEN CHILD

> SCREEN CHILD
> FILE A TRANSACTION GLOBAL_WRITE FOR PROCESS UPDATE
```

In QDESIGN, the attributes for a transaction are determined as follows:

1. The attributes are set to default values.
2. If the transaction is defined in the dictionary, then the attributes specified in the dictionary are applied and override any default attributes.
3. If there is a transaction defined for the screen, then the attributes specified on the QDESIGN TRANSACTION statement are applied and override any attributes defined previously.

QDESIGN's predefined transactions are a special case of INHERITED transactions, because references to their names are valid but their actual attributes may have been defined on another screen (and are not known until run time). In contrast, designer-defined transactions require a complete transaction definition in the screen ancestry.

In screen CHILD, below, although transaction LOCAL has the same attributes as transaction LOCAL in screen PARENT, it is still a separate transaction because the INHERITED option is not specified.

```
> SCREEN PARENT
> TRANSACTION GLOBAL_READ READ ONLY...
> TRANSACTION GLOBAL_WRITE READ WRITE...
> TRANSACTION LOCAL READ WRITE
> SUBSCREEN CHILD
> SCREEN CHILD
> TRANSACTION GLOBAL_READ INHERITED
> TRANSACTION LOCAL READ WRITE
> FILE a TRANSACTION GLOBAL_WRITE &
>  FOR PROCESS, UPDATE
```

It assumes that GLOBAL_WRITE is INHERITED. If the CHILD screen is the top level screen, or a definition for GLOBAL_WRITE is not found in any of the CHILD screen's parents, QUICK fails to load the screen. Predefined transactions are handled differently. If a definition for a predefined transaction is required but not found, in either the current screen or its parents when QUICK is loading the screen, a transaction is automatically created with all the default associations.

A QUICK transaction may be shared by many different tables or cursors, even those drawn from different databases.

A database transaction is activated as required whenever an attempt is made to read from or write to a relational file. The first attempt to activate a database transaction triggers QUICK to attach to the database unless the **subdict=nodelay** program parameter is in effect. A separate database attach and database transaction is used wherever needed. When possible (for database products that support this feature), attaches and transactions are shared and/or reused.

The Default Transactions

The QDESIGN default transactions are predefined. They are:

```
TRANSACTION CONSISTENCY READ WRITE SERIALIZABLE
TRANSACTION QUERY READ ONLY READ COMMITTED
TRANSACTION UPDATE READ WRITE REPEATABLE READ
```

The Effect of the PDL TRANSACTION Statement on QDESIGN

Options specified on the TRANSACTION statement in PDL are used in QDESIGN for transactions of the same name.

Overriding the Predefined Transactions

If you are satisfied with the attributes of QDESIGN's predefined transactions (Query, Update, Consistency) then there is no need to include TRANSACTION statements for these transactions. These predefined names can still be referenced on FILE and CURSOR statements, on SQL DML statements, or in control structures without being explicitly defined on each screen.

However, you are free to override any of the attributes of QDESIGN's predefined transactions by including a TRANSACTION statement with the name of the predefined transaction along with the attribute(s) to be changed. All unspecified attributes remain unchanged. For example, the following statement changes the isolation level of the Query transaction to serializable:

```
> TRANSACTION QUERY SERIALIZABLE
```

Only the isolation level is affected; all other attributes are retained. This causes a new Query transaction to be created. It will not be the same one used in a higher level screen.

Reserving on a Table-By-Table Basis

Tables to be reserved are specified using the "table IN database" convention, as it is used on the FILE or CURSOR statements. This allows you to include tables in the RESERVING list that are not defined on the screen where the transaction is declared. For example, the designer can reserve tables that are used on a subscreen.

If a transaction is started automatically as a result of an access or write to a database, those tables in the reserving list from a different database will not be reserved. If a transaction is started explicitly using the START TRANSACTION verb, then all tables from all databases in the reserving list will be reserved.

Reserving options are specified using keywords that match options available in the underlying database software. For example, Oracle Rdb supports options such as [PROTECTED] READ or [PROTECTED] WRITE.

The RESERVING option is especially useful for high-contention applications such as those which involve reservation of inventory or seats. It also provides a means of ensuring deadlock-free transactions since all tables are locked before the transaction does any other work. (Consider however, the impact that any table locking has on concurrent database users since there is always a trade-off to be considered. Remember also that table locks cannot be released until the transaction terminates.) The RESERVING option also provides a means of guaranteeing successful serializable updates to the reserved tables.

USE

Processes QDESIGN source statements contained in a file.

Syntax

USE filespec [DETAIL|NODETAIL] [LIST|NOLIST]

filespec

Names a file that contains QDESIGN source statements you want to use.

DETAIL|NODETAIL

DETAIL writes the contents of the file being used rather than just the USE statement itself to QDESIGN's source statement save file. NODETAIL writes just the USE statement alone, rather than the contents of the file being used. The source statement save file is qksave (**MPE/iX**) or qksave.qks (**OpenVMS, UNIX, Windows**).

The NODETAIL option can be used to reference a group of statements without including them in the source statement save file.

This prevents the repetition of the same statements in many files.

Default: As specified in the SET statement, which defaults to SET DETAIL.

LIST|NOLIST

LIST displays the statements as they are read from the source file; NOLIST doesn't.

Default: As specified in the SET statement, which defaults to SET LIST.

Discussion

The USE statement instructs QDESIGN to read the named file for statement input. QDESIGN reads and interprets each statement as if it had been entered from the terminal.

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

Nesting USE Statements

A file referenced in a USE statement can itself contain other USE statements. USE statements can be nested to a maximum of 20 levels.

Example

The following example shows how you can save time by placing a number of common layout statements in a permanent file.

The contents of the file HEADER are listed after the USE statement. QDESIGN processes these statements as though they'd been entered at the keyboard during the current session.

```
> SCREEN MODCUST
>
> FILE CUSTOMERS PRIMARY
>
> USE HEADER
> HILITE TITLE LINEDRAWING INVERSE
> DRAW 3,16 TO 6,64
> TITLE " F U T U R E  I N D U S T R I E S  I n c. " &
>    AT 4,17
> SKIP 3
> HILITE LINEDRAWING DEFAULT
>
.
```

.
.

Chapter 4: QDESIGN Procedures Overview

Overview

This chapter provides an overview of QDESIGN's procedures, as well as the verbs and control structures that make up a procedure. The topics covered in this chapter include

- QDESIGN's default and designer-written procedures
- how to generate source versions of QDESIGN's default procedures
- how to order your QDESIGN procedures effectively
- how to use verbs and control structures in procedures
- how to test processing status using predefined conditions in QDESIGN

Default Procedures and Designer-Written Procedures

Procedural processing is governed by QDESIGN procedures. QDESIGN procedures are, in effect, high-level user exits or points of intervention in QUICK processing. You can let QDESIGN construct default procedures or you can specify your own procedures directly. You can also override some default procedures and allow QDESIGN to construct the rest.

Because of the complexity of QDESIGN procedural code, you should always examine the default procedures before writing procedures that replace them. In many instances, rather than modifying generated procedures, you should modify your QUICK screen using non-procedural methods or use preprocedures and postprocedures. You can also use named or numbered designer procedures.

Default Procedures

To make screen design easy and flexible, QDESIGN constructs default versions of the basic set of required procedures. Default procedures are based on the specifications in the data and layout sections of the screen design. These default procedures are sufficient for the screen to perform its standard functions. However, you can modify them to control screen processing activities more directly.

To help you design your own procedures, you can set QDESIGN to construct the source statements of the default procedures. You can then edit these default procedures to suit the needs of your application.

Modifying Default Procedures

If you write your own version of a default procedure, QDESIGN will not override your version with a generated default procedure. Designer-written procedures prevent the construction of the corresponding default procedure. QDESIGN still constructs default procedures for those procedures that you haven't written yourself.

QDESIGN generates default versions of the following procedures:

APPEND	DELETE	DETAIL DELETE
DETAIL FIND	ENTRY	FIND
MODIFY	PATH	SELECT
UPDATE		

Note: The MODIFY and SELECT procedures are generated only when the PANEL option has been specified in either the SET or SCREEN statement. However, both the MODIFY and SELECT procedures can be designer-written for any screen, regardless of the PANEL|NOPANEL setting.

Obtaining a Source Version of Default Procedures

You can use two options of the BUILD statement to access the QDESIGN default procedures:

- The LIST option displays the source form of the constructed procedures.
- The DETAIL option causes QDESIGN's generated procedures to be included in the QDESIGN temporary save file.

You can use these two options together. For example, if you specify the statement

```
> BUILD LIST DETAIL
```

in a given screen design, QDESIGN lists all the default procedures that are generated and writes all of the procedures to the QDESIGN temporary save file.

Saving QDESIGN's Generated Procedures

Use the SAVE statement to save the statements that QDESIGN generates in a source statement file. You can then modify the saved statements using the editor and rerun QDESIGN, where you can read the modified file with a USE statement that specifies the name of the saved source statement file.

Keep the source code only for the default procedures that are modified. When QDESIGN detects a designer-written default procedure, it neither updates the procedure to reflect the current screen design statements nor replaces it.

It is very easy for these procedures to get out of synchronization with the current screen design statements, resulting in "double maintenance".

Designer-Written Procedures

In addition to the basic required procedures, there are several optional supplementary procedures for which no defaults are created. If you specify any supplementary procedures, they are used at predefined exit points in QUICK's normal processing to perform specialized editing and data manipulation.

Modified or designer-written procedures must follow layout statements (such as the FIELD and SUBSCREEN statements) and immediately precede the BUILD statement.

You can specify the following supplementary procedures in QDESIGN:

BACKOUT	DESIGNER	DETAIL POSTFIND
EDIT field	EXIT	INITIALIZE
INPUT field	INTERNAL	OUTPUT field
POSTFIND	POSTPATH	POSTSCROLL
POSTUPDATE	PREENTRY	PRESCROLL
PREUPDATE	PROCESS field	

Procedure Sequence Guidelines

When specifying your own procedures, you can make processing more efficient and easier to follow by specifying the procedures in a logical sequence. The sequence and groupings recommended when writing your own procedures are:

1.	INPUT	16.	PRESCROLL
2.	EDIT	17.	POSTSCROLL
3.	PROCESS		
4.	OUTPUT		
5.	APPEND	18	PREUPDATE
6.	PREENTRY	19.	UPDATE
7.	ENTRY	20.	POSTUPDATE
8.	MODIFY	21.	BACKOUT
9.	PATH	22.	DELETE
10.	SELECT	23.	DETAIL DELETE
11.	POSTPATH		
12.	FIND	24.	INITIALIZE
13.	POSTFIND	25.	EXIT
14.	DETAIL FIND		
15.	DETAIL POSTFIND	26.	DESIGNER

Follow these general rules when editing and writing procedures:

- When you use multiple field procedures (INPUT, EDIT, PROCESS, or OUTPUT), group them by field name in the order that the fields are processed by the ENTRY procedure.
- To use multiple DESIGNER procedures, add all of them to the end of the procedures list.
- Locate INTERNAL procedures close to and before the procedures that invoke them.
- Use all other procedures only once per screen.

For more information, see Chapter 7, "QDESIGN Procedures".

QDESIGN Verbs and Control Structures

QDESIGN verbs are used together with control structures to control processing in QDESIGN procedures.

QDESIGN Verbs

QDESIGN generates verbs automatically in default procedures. For example, FIELD statements in a screen design cause QDESIGN to generate

- ACCEPT or DISPLAY verbs in the default ENTRY procedure
- ACCEPT verbs in the default MODIFY procedure
- SELECT verbs in the default SELECT procedure
- REQUEST verbs (if the fields represent index segments) in the default PATH procedure

FILE statements cause QDESIGN to generate

- DELETE verbs in the default DELETE and DETAIL DELETE procedures
- PUT verbs in the default UPDATE procedure

The way you enter QDESIGN statements when creating a QUICK screen determines how verbs are generated in the default procedures.

Whenever possible, modify your design statements so that they generate the procedures, verbs, and constructs you need rather than modifying QDESIGN's procedures directly.

For detailed information about specific QDESIGN verbs, see Chapter 8, "QDESIGN Verbs and Control Structures".

QDESIGN Control Structures

QDESIGN control structures determine the processing flow of a procedure. With control structures, you can create blocks, loops, and conditional branches.

For detailed information about specific QDESIGN control structures, see Chapter 8, "QDESIGN Verbs and Control Structures".

Writing Procedures

Writing a procedure is a straightforward process. Simply enter the keyword procedure followed by the type of procedure required, as in:

```
> PROCEDURE PREUPDATE
```

On the line that follows the PROCEDURE keyword line, specify a procedural statement to control processing in QUICK.

Procedural Statements

A procedural statement can be one of the following

- a verb
- a compound statement
- a conditional statement
- a repetitive statement

Compound, conditional, and repetitive statements are implemented with control structures. The control structures are:

BEGIN...END	BLOCK TRANSFER	DISABLE
FOR	IF	WHILE
WHILE RETRIEVING		

Verb Statements

A verb statement consists of a verb plus its object, as in

```
> ACCEPT EMPLOYEE
```

Some verbs, such as the RETURN verb, can stand alone without an object. Others, such as the GET verb, have options in addition to an object.

For detailed information about QDESIGN verbs and how to use them, see Chapter 8, "QDESIGN Verbs and Control Structures".

You must be careful when using certain verbs in certain procedures. Verb and procedure compatibility is discussed for each procedure on (p. 239).

Compound Statements

A compound statement consists of two or more statements.

All compound statements must start with the keyword BEGIN on its own line and finish with the keyword END on its own line. The statements between the BEGIN and END keywords make up the compound statement.

You can nest compound statements within one another, as long as the BEGIN and END control structures are balanced.

Conditional Statements

Conditional statements take the form

IF condition
THEN procedural-statement
[ELSE procedural-statement]

The THEN and ELSE keywords must each begin on a separate line. The ELSE clause is optional.

You can create nested conditions by including subordinate IF control structures within the THEN or ELSE clauses of a higher-level IF control structure.

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

Repetitive Statements

A repetitive statement is executed for a fixed number of times or for each occurrence of a record or item on the screen. This type of repetitive processing is controlled by the FOR control structure.

Alternatively, you can handle repetitive processing based on the number of data records that QUICK retrieves. This type of repetitive processing is controlled by the WHILE RETRIEVING control structure.

For more information about repetitive processing, see the "FOR" and "WHILE RETRIEVING" verbs on (p. 425) and (p. 495), respectively.

Verb and Procedure Compatibility

Verbs (compatibility with procedures APPEND to INTERNAL)	APPEND	BACKOUT	DELETE	DESIGNER	DETAIL DELETE	DETAIL FIND	DETAIL POSTFIND	EDIT	ENTRY	EXIT	FIND	INITIALIZE	INPUT	INTERNAL
ACCEPT		2 11				3		4 5		2 11	3	2	4 5	*
BREAK	W	W	W	W	W	W	W	W	W	W	W	W	W	W
[SQL] CALL														*
CLEAR														*
CLOSE														*
[SQL] CLOSE														*
COMMIT														*
DELETE										8		8		*
[SQL] DELETE														*
DISPLAY		9	9			9	9	5		9	9	9	5	*
DO BLOB														*
DO EXTERNAL														*
DO INTERNAL														*
EDIT		11						4 5		11			5	*
ERROR		11								11				*
[SQL] FETCH														*
GET		11								11		25		*
INFORMATION														*

Verbs (compatibility with procedures APPEND to INTERNAL)	APPEND	BACKOUT	DELETE	DESIGNER	DETAIL DELETE	DETAIL FIND	DETAIL POSTFIND	EDIT	ENTRY	EXIT	FIND	INITIALIZE	INPUT	INTERNAL
[SQL] INSERT														*
LET						3				2	3	2		*
LOCK														*
MEMOLOG														*
NULL														*
[SQL] OPEN														*
PERFORM APPEND	E	E	E	E	E	E	E	E		E	E	E	E	E
PROMPT		2 11				3		5		2 11	3	2	4 5	*
PUSH										15				*
PUT	16	16	16	16	16	16	16	16	16	16	16	26	16	*
REFRESH														*
REQUEST	18	18	18	18	18	18	18	18	18	18	18	18	4 18	*
RETURN		4 17								17				*
ROLLBACK														*
RUN COMMAND														*
RUN SCREEN								5					5	*
RUN THREAD														*
SELECT	22	22	22	22	22			22	22	22		22	4 22	*
SEVERE		4 11								11				*
START														*
STARTLOG														*
STOPLOG														*
UNLOCK														*
[SQL] UPDATE														*
WARNING														*

Note: Explanations for the numbers, letters and symbols used in the table can be found on (p. 242).

Verbs (compatibility with procedures MODIFY to UPDATE)	MODIFY	OUTPUT	PATH	POSTFIND	POSTPATH	POSTSCROLL	POSTUPDATE	PREENTRY	PRESCROLL	PREUPDATE	PROCESS	SELECT	UPDATE
ACCEPT		4 5	3 6		3 6	24	2				4	3 21	1 7
BREAK	W	W	W	W	W	W	W	W	W	W	W	W	W
[SQL] CALL													
CLEAR													1
CLOSE													
[SQL] CLOSE													
COMMIT													
DELETE			8		8		8					8	
[SQL] DELETE													
DISPLAY		4 5	9	9	9	24	9			9		9	9
DO BLOB													
DO EXTERNAL													10
DO INTERNAL													
EDIT		5				23	20				4		7
ERROR		12				23	20				13		7
[SQL] FETCH													
GET		12	25		25	23	20					25	7
INFORMATION													1
[SQL] INSERT													
LET		14	3		3		2					3	
LOCK													
MEMOLOG													
NULL													
[SQL] OPEN													
PERFORM APPEND		E	E	E	E	E	E	E	E	E	E	E	E
PROMPT		4 5	3 6		3 6	24	2					3 21	1 7
PUSH													
PUT	16	16	16	16	16	16	16	16	16	16	16	16	
REFRESH													1
REQUEST	18	4 18		18		18	18	18	18	18	18	18 21	18
RETURN													1 17
ROLLBACK													
RUN COMMAND													1
RUN SCREEN		5											1 19

Verbs (compatibility with procedures MODIFY to UPDATE)	MODIFY	OUTPUT	PATH	POSTFIND	POSTPATH	POSTSCROLL	POSTUPDATE	PREENTRY	PRESCROLL	PREUPDATE	PROCESS	SELECT	UPDATE
RUN THREAD													1 19
SELECT	22	4 22	6		6	22	22	22	22	22	22		22
SEVERE							20						7
START													
STARTLOG													
STOPLOG													
UNLOCK													1
[SQL] UPDATE													
WARNING													

Explanations

1. Locks may be removed through terminal I/O.
2. Entered or changed data may be lost.
3. ALTEREDRECORD cannot be set to true when the predefined condition FINDMODE is true.
4. May result in an infinite loop.
5. Value in FIELDTEXT and/or FIELDVALUE may be lost.
6. Only REQUEST verb puts value in request buffer.
7. Error can force rollback.
8. Mark for deletion may be lost.
9. Display may be lost by clearing screen or overwritten by automatic display.
10. Lock conflicts can result in deadlock.
11. Error will not stop backout or exit.
12. Error results in incomplete processing.
13. Error results in incorrect processing; data has already been accepted.
14. Data must be put into FIELDTEXT to be displayed.
15. A screen's pending commands are lost on exit. The commands in the PSIB will be lost.
16. Automatic rollback is not done for non-relational files.
17. Prevents normal processing completion.
18. REQUEST verb is specific to retrieval method determination.
19. Rollback information is lost upon leaving screen.
20. Error will not cause rollback.
21. Only SELECT verb puts value in selection buffer.
22. SELECT verb is specific to selection determination.
23. Error will not stop scrolling.
24. May cause additional scrolling.
25. Retrieved data may be lost due to subsequent initialization.

Explanations	
26.	Automatic rollback is done for non-relational files if RECOVERABLE is used.
E	Results in a syntax error in QDESIGN.
W	Only allowed with a FOR, WHILE, or WHILE RETRIEVING control structure.
*	Verbs should comply with the invoking procedure's restrictions.

Testing Processing Status Using Predefined Conditions

QUICK provides several predefined conditions that you can test to determine the current status of processing. These predefined conditions are logical conditions that can be used in IF options or IF control structures.

QUICK provides several predefined conditions that allow you to test

- record status
- retrieval status
- user response
- processing modes

Testing Record Status

For each data record that's used on a screen, QUICK keeps track of the current record status.

Record status has three components:

- New (has never been put on file) or Old (exists on file)
- Changed (at least one value in the buffer has been changed) or Unchanged (for New data records, the values are all initial; for Old data records, the values are identical to what is on file)
- Deleted (marked for deletion but not removed from the file or database) or Not Deleted (not marked for deletion)

The current status of any record buffer can be determined using the predefined conditions NEWRECORD, ALTEREDRECORD, and DELETEDRECORD. For example, if the record status is New, then the predefined condition NEWRECORD is true, regardless of whether the status is Changed or Deleted. Similarly, if the record status is Changed, then the predefined condition ALTEREDRECORD is true.

The following tables illustrate the way in which QUICK sets the record status at different points during Entry and Find mode processing.

Status of Predefined Conditions in Entry Mode Processing

Processing Point	NEWRECORD	ALTEREDRECORD	DELETEDRECORD
at Initialization	True	False	False
after field entry	True	True	False
Update[1] in Action field	False	False	False
DELETE verb encountered	True	True or False	True

[1]*Includes U, UR, US, UN*

Status of Predefined Conditions in Find Mode Processing

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
Update[1] in Action field (data changed)	False	False	False
Update[1] in Action field (data deleted)	False	False	True

[1]*Includes U, UR, US, UN*

The predefined conditions listed in the preceding tables can be qualified by the record-structure name, as in

```
> IF NEWRECORD OF EMPLOYEES
>   THEN . . .
```

At initialization, the status of a record buffer is New, Unchanged, Undeleted. However, as soon as the QUICK screen user enters a value in Entry mode, the status becomes New, Changed, Undeleted. (***Note:*** For PANEL MODE screens, the record status doesn't change until the current block is transmitted to QUICK by pressing [Return].) When the data record is put on file, but before the buffer is re-initialized, the status is Old, Unchanged, Undeleted because it is reflecting values on file.

When the user retrieves a data record in Find mode, the immediate record status is Old, Unchanged, Undeleted. If the user changes a value, the status becomes Old, Changed, Undeleted. If the changed data record is updated, the status again becomes Old, Unchanged, Undeleted.

When the user marks a data record for deletion by entering the Delete (D) Action field command, the record status becomes Old, Changed, Deleted. When the data record is actually deleted by one of the update Action field commands, the status becomes Old, Unchanged, Deleted. Items in data records marked for deletion can't be referenced on the screen.

Record status also affects updating. The actions taken by a PUT verb depend on the status of the data record. For more information about PUT verb processing, see (p. 455).

Testing Record Retrieval Status

When QUICK performs a record retrieval, the predefined condition ACCESSOK is set to true or false depending on whether the retrieval succeeds or fails. For optional retrievals, this predefined condition can be tested, as in

```
> PROCEDURE EDIT POSITION
>  BEGIN
>    GET FILE POSITIONS USING FIELDTEXT OPTIONAL
>    IF NOT ACCESSOK
>     THEN ERROR = "NO POSITION ON FILE FOR CODE " &
>          + FIELDTEXT
>    END
```

Since much of QUICK's processing is based on the existence of a primary data record, the retrieval of data records of the PRIMARY file is assumed to be required. Specifying optional retrieval and testing the ACCESSOK predefined condition can lead to unpredictable results.

Testing User Response Status

When the user is prompted for entry by an accept, prompt, request, or SELECT verb, the predefined condition PROMPTOK is set to true or false, depending on whether the user enters a value or a null response. This setting can be tested, as in

```
> IF NOT PROMPTOK FOR EMPLOYEE OF EMPLOYEES
>   THEN . . .
```

Using the PROMPTOK predefined condition is discussed in more detail in the discussion sections of the field processing verbs ACCEPT, PROMPT, REQUEST, EDIT, and DISPLAY in Chapter 8, "QDESIGN Verbs and Control Structures".

Testing Processing Modes

Some procedures such as UPDATE, DESIGNER, and the field processing procedures may have to act differently, depending on the point in QUICK's processing at which they are performed.

You can test the current processing point using the following predefined conditions:

ENTRYMODE	means the screen is in the standard entry sequence or appending data.
CORRECTMODE	means the screen is in Entry mode and has completed the standard entry sequence.
FINDMODE	means the screen is performing Find mode initialization, retrieving records, or displaying retrieved data. It is also true during the initialization phase of the standard entry sequence, as this prevents ITEM INITIAL statements from changing record status.
CHANGEMODE	means the screen is in Find mode, data has been displayed, and the screen is prepared to accept changes from the user. The predefined condition, CHANGEMODE, is also true after the user enters an Update Stay command.
SELECTMODE	means the screen is performing SELECT mode initialization, retrieving records, or displaying retrieved data.

For example, your test might look like this:

```
> IF FINDMODE
>   THEN DO INTERNAL CLEANUP
>   ELSE ERROR "Valid only in Find Mode"
```

Testing Entered Values in Designer-Written Field Processing Procedures

QDESIGN has two useful predefined items that you can use to test the value entered into a field in field processing procedures. These predefined items are FIELDVALUE and FIELDTEXT.

FIELDTEXT contains the most recent set of characters in a field. Use FIELDTEXT in field processing procedures (such as an EDIT procedure) to examine the contents of the field.

FIELDVALUE always contains the most recent entry in a NUMERIC or DATE field. The contents of FIELDVALUE are derived from FIELDTEXT after the optional INPUT procedure is performed on the field. The storage format of FIELDVALUE is determined by the item datatype for the item that's associated with the current field.

There is only one FIELDTEXT and one FIELDVALUE item per QDESIGN session.

Restrict the use of FIELDTEXT and FIELDVALUE to field processing procedures (INPUT, EDIT, OUTPUT). If you use FIELDTEXT and FIELDVALUE in other contexts, unpredictable results can occur.

For more information about the use of FIELDTEXT and FIELDVALUE, see the ACCEPT verb on (p. 364).

Chapter 5: QUICK's Processing Modes

Overview

This chapter discusses QUICK's three main modes of operation: Entry, Find, and Select. The chapter ends with a brief discussion of Append mode processing, which is an extension of both Find and Entry mode.

Understanding the Relationship Between QDESIGN and QUICK

Before you create systems of QUICK screens with QDESIGN, you must understand how your screen designs affect the way the screens function once they are created and in use.

Each command a QUICK screen user enters in the Action field on a given QUICK screen causes QUICK to perform a predetermined series of processing steps, and to execute one or more of the procedures defined in QDESIGN for that screen. Similarly, pressing one function key when the cursor is positioned on a data field causes QUICK to perform a predetermined series of processing steps.

Understanding QUICK's Processing Modes

QUICK's operation is guided by the compiled screen definitions that are created by the BUILD statement in QDESIGN.

The compiled screen definition consists of four parts:

- the screen background (the fixed display portion of the screen)
- tables of record-structures, items, and indexes
- the screen foreground (fields)
- procedural code that defines the activities to be performed

QUICK operates in two basic modes: Entry and Find. Select mode is an extension of Find mode.

When a QUICK screen is activated, its background is immediately displayed. The user signals the desired processing mode by entering one of the following commands in the Action field:

- E for Entry mode
- F for Find mode
- S for Select mode

Once in a processing mode, actual processing is controlled by a combination of

- procedural processing
- automatic housekeeping functions (performed by QUICK based on the tables in the compiled screen)

During each phase of processing, QUICK places entered and retrieved data in temporary storage areas called buffers. QUICK uses different buffers, depending on whether the QUICK screen user is performing a Find, a Select, or an Entry operation.

The sections that follow describe the sequence of events in Entry, Find and Select mode processing. In addition, the final section of this chapter discusses Append mode processing, which allows you to enter data records in either Entry or Find mode.

Entry Mode Processing

In Entry mode, QUICK screen users can create, validate, and store data in files. Entry mode consists of a series of four phases that are repeated for each set of data created:

- Initialization phase
- Entry phase
- Correction phase
- Update phase

The Correction phase is optional. You can bypass this phase by using the AUTOUPDATE option of the SCREEN statement. QUICK then proceeds directly from the Entry phase to the Update phase.

The Initialization Phase

In this phase, temporary items are initialized, record buffers are marked for initialization, and the foreground portion of the screen is cleared and/or set to initial predisplayed values. The initialization phase is performed automatically by QUICK based on specifications in the data and layout sections of the screen design. There is no procedure for this phase.

The Entry Phase

The Entry phase is controlled by

- the PREENTRY procedure (if one is specified)
- the ENTRY procedure
- the APPEND procedure (if the screen includes a DETAIL file or a repeating PRIMARY file)

Normally, QUICK prompts the user through the fields or blocks of fields on the screen. The PREENTRY, ENTRY, and APPEND procedures validate values as they're entered and then places them in the buffers. If procedures contain BLOCK TRANSFER constructs, QUICK validates groups of fields delimited by the BLOCK TRANSFER control structures simultaneously.

Lookups and Calculations in the Entry Phase

The Entry phase can involve lookups and retrieval of related data from record-structures in Reference files, as well as calculation of data values based on the data entered or retrieved.

The Default ENTRY Procedure

QDESIGN constructs a default version of the ENTRY procedure based on the field and other layout section statements and their options. If the PANEL option is specified for either the SET or the SCREEN statement, then the generated ENTRY procedure automatically contains BLOCK TRANSFER control structures. In addition, if the screen design includes a detail record-structure or a primary record-structure with multiple occurrences, an APPEND procedure is also constructed. You can modify these procedures using their source code.

The Correction Phase

In this phase, the user can make corrections to the data created in the Entry phase. There are two ways QUICK screen users can make corrections to field entries:

- with the M (Modify) command
- with numbered DESIGNER procedures

The MODIFY Command

The M (Modify) command is valid only when a MODIFY procedure exists for the screen. The MODIFY procedure is automatically generated for PANEL screens. You can write your own MODIFY procedure for NOPANEL screens. When the MODIFY procedure is initiated, QUICK prompts for corrections at specific fields that are referenced in the MODIFY procedure. QUICK determines which fields the user can change (based on field options such as NOCHANGE, NOCORRECT, and DISPLAY) and enables these fields for input.

If AUTOUPDATE is specified on the SCREEN statement, QUICK skips the Correction phase and goes to the Update phase.

Numbered DESIGNER Procedures

QUICK generates internal numbered DESIGNER procedures for each field for which there is an ID-number. The generated numbered DESIGNER procedures only contain ACCEPT verbs for the fields associated with that ID as long as the fields don't have the NOCORRECT, or DISPLAY options.

You can't access the default DESIGNER procedures. However, you can write your own numbered DESIGNER procedures to override the default numbered DESIGNER procedures. In such cases, the numbered DESIGNER procedures perform whatever steps they've been customized to perform.

Numbered DESIGNER procedures are similar to the ENTRY procedure; the difference is that numbered DESIGNER procedures affect individual fields rather than groups of fields. When the QUICK screen user enters the number of a DESIGNER procedure in the Action field, QUICK prompts for corrections at the field with that ID-number.

The field ID-number doesn't have to be visible on the screen in order for the associated numbered DESIGNER procedure to be in effect. Fields with the HIDDEN option don't have a displayed ID-number; they do, however, have associated ID-numbers (which are hidden from view) and associated DESIGNER procedures.

Deleting Newly Entered Records in the Correction Phase

Newly entered data records can be deleted in the Correction phase (that is, prior to an update) by the Delete command. Because newly created data records don't yet exist in a file, the Delete command used in the Correction phase simply prevents the records from being placed in a file. This activity is controlled by the DELETE and the DETAIL DELETE procedures. QDESIGN constructs default versions based on the FILE statements; these default procedures are available as source code. Both procedures mark records for deletion.

The Update Phase

After the data has been entered and corrected, the user initiates the Update phase by entering one Update command. This phase is controlled by three procedures and some automatic housekeeping.

The first procedure is the optional PREUPDATE procedure. This user exit allows you to specify infield editing or other processing that depends on having all data from all the fields on the screen available. Balance checks are performed following the Preupdate phase if they were specified in the data section, and if no errors have been issued by the PREUPDATE procedure.

If no errors are detected, control is passed to the second procedure, the UPDATE procedure, after balance checking. The default procedure is constructed by QDESIGN from the FILE statements and from associated ITEM and TARGET statements. The source version of the UPDATE procedure is available.

If errors are detected at any point in the UPDATE procedure, all updates previously performed in that procedure are rolled back and the user is returned to the Correction phase. For more information about rollbacks, see (p. 357).

The third procedure is the optional POSTUPDATE procedure that performs processing following the successful completion of an UPDATE procedure. After executing the POSTUPDATE procedure, QUICK verifies that all data records (except DESIGNER file records) that have been changed have also been updated. QUICK warns the user if any data records failed the test.

If the user enters an Update Stay command, the files are updated and then the screen reverts to the Change phase for additional modifications to the same set of data. For more information about the Change phase, see (p. 251).

Following these four stages of Entry mode processing, QUICK returns to the Initialization phase for the next set of data.

Find Mode Processing

In Find mode, QUICK screen users can retrieve, view, alter, or delete existing data records. Find mode consists of three separate processing phases:

- Initialization phase
- Path Determination phase
- Retrieval Cycle phase

The Initialization Phase

When a QUICK screen user enters Find mode, QUICK clears the screen foreground, initializes temporary items, and marks record buffers for initialization. This process is done automatically from specifications and options in the data and layout sections of the screen design. There is no procedure for this phase.

Path Determination Phase

This phase establishes which data is to be retrieved and how it is to be retrieved. The Path Determination phase is controlled by the PATH procedure, a default version of which is constructed by QDESIGN based on FILE and associated ACCESS statements and the availability of fields for segments. The default procedure is available in source form.

The PATH procedure establishes a dialogue with the screen user to determine which index and which index values are to be used for data retrieval. By default, QUICK prompts the user at each field that's associated with a segment in an index of the primary record-structure. Based on the user's responses, both the access path and its index value are established. Once established, these can't be changed during the retrieval cycle.

The optional POSTPATH procedure is executed following the successful completion of the PATH procedure. You can include this procedure to perform processing before the record-retrieval cycle begins.

On some lower-level screens in a data hierarchy, the criteria for retrieval is fixed and based on the current values in the MASTER file record that is passed to that screen. In these cases, QUICK constructs a simplified PATH procedure that sets PATH to 1.

The Retrieval Cycle Phase

The Retrieval Cycle phase is itself broken down into several phases, each of which is repeated until QUICK has retrieved all the data records that meet the retrieval criteria of the Path Determination phase, or until the user interrupts the process. The phases of the Retrieval Cycle phase are:

- Retrieval Initialization phase
- Data Retrieval phase
- Display Data phase
- Change phase
- Update phase

Retrieval Initialization Phase

QUICK clears the screen foreground, initializes temporary items to their default values, and marks record buffers for initialization. This phase is performed automatically by QUICK, based on the data and layout sections of the design and on the retrieval criteria established in the PATH procedure.

Data Retrieval Phase

During this phase, QUICK retrieves the data record or data records for primary and secondary record-structures, as well as for detail record-structures, if present. The FIND procedure controls this activity. The FIND procedure retrieves one screenload of data based on the retrieval criteria specified in the PATH procedure. If included, a POSTFIND procedure is executed after successful completion of the FIND procedure. If a detail record-structure is included in the screen design, the DETAIL FIND procedure is executed to retrieve its data records. If included, a DETAIL POSTFIND procedure is executed after the successful completion of the DETAIL FIND procedure. Default versions of the FIND and DETAIL FIND procedures are constructed by QDESIGN based on FILE and associated ACCESS statements, and on the availability of fields for segments. The default procedures are available in source form.

Display Data Phase

After QUICK retrieves data, the screen foreground is displayed to the user. This is done automatically based on the data and layout sections of the screen design.

If the AUTOMODIFY option is specified on the SCREEN statement, the MODIFY procedure is initiated and the screen enters the Change phase automatically after QUICK retrieves data.

Change Phase

The screen user can make changes to the displayed data. This activity is the same as the Correction phase in Entry mode (p. 248).

Numbered DESIGNER procedures may perform slightly differently in the Correction phase of Entry mode or the Change phase of Find mode because of mode-dependent options of the FIELD statements. Data records can be deleted in the Change phase (as in the Correction phase). However, because the deleted data records were retrieved and therefore exist in a file, actual physical deletion of the data records doesn't occur until the Update phase.

Update Phase

The screen user signals the end of the Change phase by entering an Update command. The Update phase is identical to the Entry mode Update phase, except that existing data records are updated (new data records aren't added) and marked data records are physically deleted.

Notes on Find Mode

The PATH and FIND procedures are closely related. If you modify or explicitly code either of these procedures in a QUICK screen design, you must code the other procedure to match it. However, modifying these procedures is not recommended. For detailed information about the default FIND and PATH procedures, see (p. 318) and (p. 333).

Find mode allows the user to browse through data records by pressing [Return] to see the next record.

If more data records exist than can be retrieved and shown in one Find sequence, the retrieval cycle is repeated each time the user presses [Return] until no more data records meeting the retrieval criteria are found. When this happens, QUICK displays a screen with a blank foreground indicating the end of the retrieval sequence.

If no changes are made to a set of retrieved data records, the user can press [Return] to retrieve and display the next set of data records. An update isn't required if nothing is changed. If an update is performed, only data records that have been changed are actually updated.

When the user enters the Update command or presses one Update function key, the files are updated and the next set of data records is presented. When you include a DETAIL record-structure in the screen design, the Update command also updates the DETAIL file and any data occurring with it, and causes the next set of detail records to be presented. The primary file data record remains on the screen.

When the user enters the UN (Update Next) Action field command on a screen that includes a DETAIL record-structure, that file and the PRIMARY file are updated, and the next primary file record and its related detail file records are shown.

When the user enters the US (Update Stay) Action field command in Find mode, QUICK updates the screen and reverts to the Change phase of processing for additional actions on the same set of data.

When the user enters the UR (Update Return) Action field command, all files on the screen are updated, and QUICK returns to the immediately higher-level screen. If no higher-level screen exists, QUICK returns to the point from which it was invoked (usually the operating system).

Select Mode Processing

Select mode processing is identical to Find mode processing, with one additional phase. Following the execution of the PATH procedure, QUICK performs the following steps:

- If a SELECT procedure exists for the screen, QUICK executes it.
- If a SELECT procedure doesn't exist for the screen, QUICK prompts the user for field ID-numbers. Once ID-numbers are entered, QUICK prompts for additional record selection criteria at each field for which an ID-number was entered, with the exception of relational blobs.

The values entered in Select mode supplement rather than override the values entered in response to the PATH procedure. Any non-null data values entered into fields in response to the prompts are treated as selection values.

To use empty fields as part of the selection criteria, the user enters one blank for character items, and one zero for numeric or date items. If the values of the items in retrieved data records don't match the specified selection values, the data records are bypassed. Selection values can't be entered into temporary or defined item fields. The POSTPATH procedure (if present) is executed following the prompting for selection values.

Append Mode Processing

Append mode processing streamlines data entry by allowing the QUICK screen user to append data records while in either Entry or Find mode. It also gives QUICK screen designers the option of setting up a one-to-many relationship between record-structures on a single screen.

Implementing and using Append mode processing involves repeating PRIMARY or DETAIL record-structures, several procedures, and Action commands.

Procedures and Verbs Used in Append Mode Processing

APPEND	A procedure generated by QDESIGN to control Append mode processing.
DETAIL FIND	A procedure generated by QDESIGN to control the retrieval of data records from the file and the files occurring with the detail file.
DETAIL DELETE	A procedure generated by QDESIGN to control the marking for deletion of specific data records from detail files and the files occurring with the detail files.

DETAIL POSTFIND	A designer-written procedure that performs processing after the successful completion of a DETAIL FIND procedure.
PERFORM APPEND	The verb generated by QDESIGN in the ENTRY procedure, or in a FOR MISSING loop in the MODIFY procedure. PERFORM APPEND controls the execution of the APPEND procedure in the entry sequence.

Action Field Commands Used in Append Mode Processing

A	Append	The Action field command that initiates Append mode processing.
N	Next full data	The Action field command that, in Find mode, bypasses retrieval of any further detail records associated with the current primary file record, and begins retrieval of the next primary file record.
UN	Update Next	The Action field command that, in Find mode, updates the current screen, bypasses retrieval of any further detail records associated with the current primary file record, and begins retrieval of the next primary file record and its associated detail records. In Entry mode, it updates the current screen, and prompts for a new primary file record.

Append Mode Processing and Primary Record-Structures

Append mode processing on a repeating primary record-structure allows you, in CHANGEMODE or CORRECTMODE, to add data records to empty occurrences on a screen without clearing data records currently displayed on the screen.

For example, you can combine different modes as follows:

- To retrieve a group of data records, use the F (Find) Action field command.
- To add new data records, use the A (Append) Action field command.
- To update, use any of the update function keys or one of four update Action field commands (U, UR, UN, US).

To terminate Append mode processing, enter the Skip All (//) command.

When all occurrences are full, QUICK prompts in the Action field for the next instruction. Append mode processing is like Entry mode; at the end of the entry sequence, the user can make corrections to entered data before updating.

Append Mode Processing and Detail Record-Structures

Using detail record-structures provides an easy way to establish one-to-many relationships between a primary record-structure and related subordinate record-structures. If only two record-structures are involved, you can avoid creating higher- and lower-level screens linked by a SUBSCREEN statement.

Notes on Append Mode Processing

The ACTIVITIES option of the SCREEN statement in QDESIGN affects Append mode processing. The ENTRY activity option allows Append processing in Entry mode, while the CHANGE activity option allows Append mode processing in Find mode and Select mode.

If a screen has a repeating primary record-structure and the only activities are FIND and CHANGE activity options, Append mode processing isn't allowed. Data entered into primary file records using Append mode processing is considered New, Unchanged.

Chapter 6: Customizing QUICK with QKGO

Overview

QKGO is the PowerHouse utility that lets you:

- modify QUICK execution-time parameters
- rename QUICK action and data commands
- assign QUICK action and data commands to function keys using the terminal interface configuration
- define function keys through DFKs

An alternative to much of the functionality in QKGO is the QUICK Initialization file, or QKI (p. 290), a text file which you can create with a text editor.

QKGO: The QUICK Execution-Time Parameter File-Set

A QKGO file-set establishes QUICK's basic working parameters, such as

- the first screen called by QUICK
- the default mode of the first screen
- execution-time table sizes
- the length of the optional time-out period
- the symbols used for all QUICK commands
- the default ASCII line-drawing symbols
- terminal interface configuration (TIC)
- dynamic function keys (DFKs)

Each time QUICK is started, it looks for a QKGO file-set. QUICK first checks for the **auto** program parameter. If **auto** isn't specified:

MPE/iX:	QUICK looks for the QKGO files in the current group, or for a file equation for QKGO.
OpenVMS:	QUICK looks for the logical name QKGO. If not found, it looks for the logical name PH_DEFAULT_QKGO.
UNIX, Windows:	QUICK uses the environment variable QKGO.

You don't need to have an active QKGO file-set in order to run QUICK. Default values are assumed for all parameters when a QKGO file-set doesn't exist. The default values for the table-size parameters are fairly low in order to avoid using unnecessary memory space. However, you may be able to reduce memory usage further and increase performance by changing some parameter values using QKGO.

PowerHouse provides default terminal interface configurations for many terminals. The default QKGO file-set is automatically made available when QKGO is specified.

Alternatives to Custom QKGO file-sets

A QKGO file-set isn't required if the only changes you intend are to

- name the first screen that QUICK calls ("First screen" field)

- name your dictionary ("Dictionary File" field)

The **auto** program parameter provides the same functionality as the "First screen" field in the Construction and Maintenance Screen. Alternatively, you can use:

MPE/iX:	`FILE QKGO = MAINMENU`
OpenVMS:	`DEFINE QKGO SCREEN1.QKC`

UNIX:	`setenv QKGO mainmenu (for the C shell)`
	`QKGO=mainmenu`
	
	`export QKGO (for Bourne or Korn shell)`

Windows:	Create a system environment variable **QKGO** whose value is `[drive:\][directory\]...mainmenu`.

Similarly, the **dictionary** program parameter provides the same functionality as the "Dictionary File" field in the Construction and Maintenance Screen. Alternatively, you can use the following to set the PHD designated file:

MPE/iX:	`FILE PHD = MPEIXPHD`
OpenVMS:	`SETDICT PHD`

UNIX:	`setenv PHD unixdict (for the C shell)`
	`PHD=unixdict`
	
	`export PHD (for Bourne or Korn shell)`
	`setdict phd.pdc`

Windows:	Edit the system environment variable **PHD** to point to your dictionary: `set PHD=windict`

F or more information about program parameters, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

The QUICK initialization file (p. 290) is an alternative to four of the main screens within QKGO: Execution Time Parameters, Action Field Commands, Action and Data Field Commands, and Data Field Commands. The QKI file is a text file that you can create with a text editor.

An alternativeto the Terminal Interface Configuration screen within QKGO is the TIC text file(p. 278).

Starting QKGO

To start QKGO, enter the following command:

MPE/iX:	`SETQKGO`
OpenVMS:	`QKGOMAINTENANCE or QKGOMAINT`
UNIX:	`qkgo`
Windows:	From the Start Menu, select PowerHouse *<version>* and click on QKGO. A QKView version is available by clicking on QKGOQKView.

The main screen of QKGO, the Construction and Maintenance Screen, is displayed.

Exiting QKGO

To terminate QKGO and save any changes made during your session, enter the Update Return (UR) command in the Enter command field of the Construction and Maintenance Screen.

If you do not want to save your changes, enter the Return (^) command in the Enter command field of the Construction and Maintenance Screen.

If you make changes during your session and do not issue an update command before issuing a Return command, QKGO prompts you with the warning message:

```
Data has been changed but not updated. Repeat the action if this is ok.
```

Choosing Options on QKGO Screens

QKGO screens behave similarly to other QUICK screens. One notable difference is that the Action field is often labeled "Enter command". You navigate by entering a subscreen ID-number or a valid Action command in the Enter command field.

To change field values, enter the field's ID-number in the field labeled "Enter command", then, when you are prompted, enter the new value.

MPE/iX, UNIX, Windows:	Values that are different from the default are displayed in halftone typeface.
OpenVMS:	Values that are different from the default are displayed in bold typeface.
MPE/IX, UNIX, Windows:	The function keys on QKGO screens for HP and HP-compatible terminals and Windows consoles are dynamic. The labels indicate the function of each key. You can use these keys to select appropriate actions for the screen you are using. The QKGO menu screen employs two shift levels. You use the SHIFT function key to shift between levels.

Getting Help

You can access online help messages by entering the Help command (?) in any field. It gives a brief description of what's expected in that field. The Extended Help command (??) displays a more detailed explanation.

OpenVMS: [PF2] and [GOLD/PF2] can also be used for Help and Extended Help.

Performing Lookups in Fields on QKGO Screens

Another form of help is available within many of the options in the Terminal Interface Configuration (TIC) subscreen system. PowerHouse allows you to access lookup information for a specific field.

To do this, enter an equal sign (=) in the field that you want information on and press [Return].

For example, you can enter an equal sign in the Logical Key Name field in the System Commands subscreen (in the TIC Screen). This calls the Logical Key Names screen and displays all the valid mnemonic codes. You can select a mnemonic by entering the id-number that precedes the key name.

The Construction and Maintenance Screen

You create and maintain QKGO file-sets with a system of QUICK screens that provide an interface to QUICK's operating parameters. The Construction and Maintenance Screen is the main screen of QKGO.

Parameter values and operator commands are divided into subscreens, which are activated by entering the preceding id-number. The options on this screen are:

QKGO file	When you choose Entry mode, you are prompted for the name of the new QKGO file-set. When you choose Find mode, you are prompted for the name of an existing QKGO file-set.
Initialize from	When you enter COPY in the Enter command field, you are prompted in this field for an existing QKGO file-set from which to copy the settings.
01 First screen	The name of the first screen to be displayed when QUICK is initiated. On entry, QKGO does not verify the existence of the screen. If this is the only parameter that you intend to customize in your QKGO file-set, see (p. 255). Limit: Must be a valid file specification.
02 Dictionary File	The name of the dictionary to be accessed when QUICK is initiated. On entry, QKGO does not verify the existence of the dictionary. If the First Screen and Dictionary File parameters are the only parameters that you intend to customize in your QKGO file-set, see (p. 255). Limit: Must be a valid file specification.
03 Execution-time parameter values	Use this screen to change the values of QUICK runtime parameters. For more information, see (p. 261).
04 Action field commands	Use this screen to rename QUICK Action field commands. For more information, see (p. 268).
05 Action and Data field commands	Use this screen to rename commands common to both the action and data fields. For more information, see (p. 269)
06 Data field commands	Use this screen to rename data field commands, and to modify the generic retrieval character and the pattern match character. For more information, see (p. 270).
07 Dynamic Function Keys	Use this set of screens to control dynamic function key (DFK) parameters in your application. For more information, see (p. 271).
08 Terminal interface configuration (TIC)	Accesses a set of screens that lets you define and maintain the terminal keys that are used to control various QUICK User Interface features. Function keys, keypad keys and escape sequences (control characters) can be mapped to execute QUICK commands and designer procedures. For more information, see (p. 278).

Specifying QKGO File-Sets

You must specify a QKGO file-set before you choose any subscreen options or procedures available on the main screen. To specify a QKGO file-set, you can create a new file, or you can modify or convert an existing QKGO file-set.

Creating QKGO File-Sets

To create a new QKGO file-set, use the Enter (E) command in the Enter command field.

You are prompted in the "QKGO file" field. Enter a name for your QKGO file-set. Once the file "filename" has been created, the following message is displayed:

```
New QKGO file has been created with default values.
```

OpenVMS, UNIX, Windows: You do not need to specify a file extension. QKGO automatically appends the default extension to each physical filename in the file-set.

To make the file permanent, enter the Update Stay (US) command in the Enter command field. If you leave the screen before updating, the newly created QKGO file-set is deleted.

You can now choose any of the options or procedures on the screen.

Copying and Converting QKGO File-Sets

To copy the values from an existing QKGO file-set to a new QKGO file-set:

1. Enter the Entry Mode (E) command in the Enter command field.
2. When prompted in the QKGO file field, enter a name for your new QKGO file-set.
3. Enter COPY in the Enter command field.
4. When QKGO prompts you in the Initialize from field, enter the name of the existing source file.

   OpenVMS: If you are copying or upgrading a 7.10 or 8.xx QKGO file, the following message is displayed:

   ```
   Convert an existing 7.10 QKGO file? [Y/N]
   ```

   Enter Y if the file is a 7.10 QKGO file. The default is N.
5. QKGO asks if you want to copy the TICs.Enter Yes if you want to copy terminal information from the existing file. Enter No to assume new defaults.

In addition to the file specified in the QKGO file field, QKGO creates data files, as described in the section, "Physical QKGO File-sets" on (p. 260).

To make the files permanent, enter the Update Stay (US) command in the Enter command field. If you leave the screen before updating, the newly created QKGO file-sets are deleted.

You can now choose any of the options or procedures on the screen.

Modifying or Deleting a QKGO File-Set

To modify or delete an existing QKGO file-set, enter the Find (F) command in the Enter command field.

QKGO prompts you to enter the name of the file-set in the "QKGO file" field. Once retrieved, the QKGO file-set can be modified or deleted.

To delete the file-sets, enter the Delete (D) command in the Enter command field. If you exit QKGO without updating, the QKGO file-sets will not be deleted.

Physical QKGO File-Sets

QKGO file-sets comprise the following file(s):

Platform	File	Description
MPE/iX:	filename	This file always exists in QKGO file-sets.
UNIX, Windows, OpenVMS:	filename.qkg	
MPE/iX:	filenameb	This context binding file exists in QKGO file-sets only if terminal (TIC) information is specified.
UNIX, Windows:	filenameb.dat and filenameb.idx	
OpenVMS:	filnameb.dat	
MPE/iX:	filenamek	This key sequence file exists in QKGO file-sets only if terminal (TIC) information is specified.
UNIX, Windows:	filenamek.dat and filenamek.idx	
OpenVMS:	filenamek.dat	
MPE/iX:	filenamet	These terminal-group files exist in QKGO file-sets only if terminal (TIC) information is specified.
UNIX, Windows:	filenamet.dat and filenamet.idx	
OpenVMS:	filenamet.dat	

When copying a QKGO file-set between directories, be sure to copy all the files in the file-set.

Changing Values in the Subscreens

The current values from the QKGO file-set are displayed when the following screens are called:

03 Execution-Time Parameter Values

04 Action Field Commands

05 Action and Data Field Commands

06 Data Field Commands.

MPE/iX, UNIX, Windows:	Values that are different from the default are displayed in halftone typeface.
OpenVMS:	Values that are different from the default are displayed in bold typeface.

To change a value, select the field ID-number and enter the new value. Only displayable characters are allowed. In the Action Field Commands, Data Field Commands, and Action and Data Field Commands screens, you can change a value back to the default by entering <DEF> in the field.

The Execution-Time Parameter Values Screen

The Execution-Time Parameter Values screen allows the values of certain QUICK run-time parameters to be adjusted. Use the screen to change QUICK's operating characteristics.

To access the screen, enter option 03 "Execution-Time Parameter Values" on the QKGO Construction and Maintenance Screen.

This table lists the execution-time parameters, their range of values, and their default values. Refer to this table to determine which values to optimize to reflect your production requirements.

Parameter	Purpose	(Min./Default/Max.)
Application lines	Sets the number of lines of simulated terminal memory for stacking screens.	(24/48/240)
Block Mode Retries (**MPE/iX**)	Controls the number of times QUICK retries a block mode read (if original read fails).	(0/5/15)
Common area size	Sets the maximum space (in words) allocated for use by DO EXTERNAL subroutines.	(1/1/30000) **MPE/iX, UNIX, Windows** (0/0/30000) **OpenVMS**
Do Ext Save/Restore *	Specifies whether the terminal sets/resets when DO EXTERNAL subroutines are executed from QUICK. A value of Y means the setting/resetting takes place; a value of N means it doesn't.	(/N/)
Driver (**OpenVMS**)	Specifies the DO EXTERNAL subroutine driver program name.	/QKDRIVER/
Error Recall	Controls the redisplay of invalid data on the field when a data edit fails. This parameter may be overridden for a specified field with the ERRORRECALL or NOERRORECALL options on the FIELD statement. The options are Y or N.	/N/
Expression size	Sets the maximum space (in words) reserved for evaluating expressions and function results.	(20/400/8192)
Ext. subroutines (**MPE/iX, UNIX, Windows**)	Establishes the maximum number of external subroutines QUICK can call. Each unique external subroutine reference uses a table entry. If the value is too low, QUICK issues an error message. Using too high a value wastes memory.	(0/1/50)

Parameter	Purpose	(Min./Default/Max.)
Field terminators	Controls data input. The options are full field (F), autonext (A), audible autonext (B), or manual (M). The full field option (F) ensures that the data entered by the user does not exceed the size of the data item. The terminal sounds a warning when the data exceeds the field size. No more data is accepted at that point. The user can then edit or delete the data in that field. The autonext option (A) automatically moves the cursor to the field once input exceeds a field size. The audible autonext option (B) behaves like the autonext option but causes the terminal to sound a warning when the cursor moves to the next field. The manual option (M) disables all field terminator options and requires the user to press [Return] to terminate field input. This option does not apply to the Action field, the REQUEST verb, or the PROMPT verb.	(/M/) **MPE/iX, UNIX, Windows** (/F/) **OpenVMS**
Horizontal lines	Sets the default ASCII character to be used for horizontal lines on a terminal that doesn't support the line-drawing character set.	
HP terminal has LDW (**MPE/iX, UNIX**)	If terminals support line drawing, choose Y. This way, you don't have to specify the line drawing character set at execution time.	(/N/)
Initial mode	Specifies that the first screen entry is to start in one of these modes: Entry, Find, or Select (E, F, or S).	(/blank/)
Input Mode (**MPE/iX**)	Specifies the assumed terminal input mode (BLOCK or CHARACTER).	(/C/)
Line Intersections	Sets the default ASCII character to be used for crossing lines on a terminal that doesn't support the line-drawing character set.	
Lock Attempts (**MPE/iX, UNIX, Windows**)	Sets the number of lock attempts allowed. If the number of allowable lock attempts is exceeded, QUICK displays a message.	(1/16/50)
Lock Message Wait (**OpenVMS**)	Sets the time delay, in seconds, between a lock request for file access and the display of the message "The file is busy. Please wait..." This message is displayed until the maximum lock request specification has been exhausted. If the lock request is not granted by this time, the message "Unable to complete the requested action at this time" is displayed. The value entered here must be less than that entered for the LOCK REQUEST WAIT parameter. If no message is required, enter a value of 0.	(0/5/254)

Parameter	Purpose	(Min./Default/Max.)
Lock Request Wait (**OpenVMS**)	Sets the maximum amount of time for which a lock request is queued.	(1/30/255)
Lock Retry Interval (**MPE/iX, UNIX, Windows**)	Sets the number of seconds QUICK waits between locking attempts. A value of 0 means there is no pause between lock attempts.	(0/2/60)
Lock Unconditional (**MPE/iX, UNIX, Windows**)	Y indicates the first lock attempted will be unconditional. N indicates the first lock attempted will be conditional.	(/N/)
Max number of threads	Sets the number of threads that can be active in one session.	(1/3/7)
Max. Paged Memory (**OpenVMS**)	Sets the maximum amount of memory (in 512-byte pages) that can be used for QUICK's internal paging system. (i.e. update, backout and inactive screens)	(0/2048/16384)
Rollback Buffer*	Sets the size of the primary memory buffer(in multiples of 128 words) used to store rollback information.	(128/1024/32767)
Rollback Clear* (**OpenVMS, UNIX, Windows**)	The default setting of Yes uses rollback pending. A setting of No causes an immediate rollback while keeping screen buffers under certain conditions.	(/Y/)
Rollback Time-out*	Specifies the number of seconds QUICK waits for input during a rollback pending state before it terminates blocking transactions. A value of 0 means QUICK doesn't time out.	(1/0/65535)
Run Cmd Save/ Restore*	Specifies whether the terminal sets/resets when COMMANDs or RUN COMMANDs are executed from QUICK. A value of Y means the setting/resetting takes place; a value of N means it doesn't.	(/Y/)
Screen levels*	Sets the maximum number of levels in the screen hierarchy. QUICK keeps at most this number of screen levels open and mapped onto terminal memory, even if they are not active screens.	(1/5/15)
Screen Section (**OpenVMS**)	Determines availability of sections for screen and dictionary file sharing. Sharing is only available for .PDC dictionaries. The options are PRIVATE(P), GROUP(G), and SYSTEM(S). The SYSTEM option indicates that a section is available to all processes in the system; it is recommended if the application or dictionary is to be available to all users in the system. PowerHouse must be installed with SYSGBL to use the SYSTEM option. The GROUP option indicates that a group section can be shared only by processes that are executed from within the same group. The PRIVATE option indicates which private section is available to the process.	(P/G/S)

Parameter	Purpose	(Min./Default/Max.)
Screen table*	Sets the maximum number of screens that QUICK can keep track of concurrently.	(1/15/50)
Secondary Blocks* (**OpenVMS**)	Refers to the number of blocks that QUICK can maintain to write to extra data segments and/or temporary files. Each block is the same as the rollback buffer.	(0/32/1000)
Selection Size* (**OpenVMS**)	Sets the maximum space (in words) per screen reserved for storing retrieval selection conditions.	(0/128/2000)
Terminal buffer*	Sets the maximum number of characters (one ASCII character equals one byte) in the terminal input/output buffers.	(80/512/4000)
Terminal Time-out	Sets the number of seconds QUICK waits for input before terminating. A value of 0 means QUICK doesn't time out.	(30/0/65535)
Upshift actions	Upshift user entries in the Action field. The options are Y or N.	(/Y/)
Vertical lines	Sets the default ASCII character to be used for vertical lines on a terminal that doesn't support the line-drawing character set.	

* *These parameters are described in greater detail later in this section.*

Using QKGO to Adjust Execution-Time Parameters

Many of the parameters listed in the previous table are used to set table or work area sizes. The value must be large enough to allow the application screens to execute. Larger values waste memory; however, if performance is a concern, you can increase the values at the expense of memory. Performance parameters must be selected based on your system requirements and resource usage.

Screen Tables and Work Area Parameters

Application lines

Establishes the number of lines of simulated terminal memory used for stacking screens. A screen is written out to application lines only if it's to be overwritten by the screen image QUICK maintains in primary memory. The value should be the highest line number used in the stacking and windowing options of the SCREEN statement. Although reducing this parameter below the default does save some memory space, the resulting slowdown due to screen reloading and repainting is usually not worth the savings. If the value is too low, screens are mapped onto the lines available.

Block Mode Retries (MPE/iX)

Set this parameter to control the number of times that QUICK attempts to re-read data from a block mode screen if a read should fail. This is usually caused by communication link problems. The default is 5, the maximum is 15, and 0 disables retries. QUICK displays retry messages and a count for each time it attempts to re-read the data. You must press [Enter] to resend the data.

Expression size

Establishes the size of the work area used for the evaluation of expressions and function results. If the value is too low, QUICK issues an error message. You should estimate the size by using the most complex expressions and functions in your application system. Adjust the value upwards until evaluation succeeds. Using too high a value wastes memory.

Rollback Buffer

Establishes the size of the buffer used to store information needed to rollback updates if an error occurs in the UPDATE procedure. If memory is available, the value of this parameter should be made as large as possible. The minimum value would be the length of the largest record, rounded up to the nearest 128 words.

The Rollback Buffer may not be large enough to hold all the information required. This may happen if you are deleting a large number of records with a delete file statement. In this case, the contents are written to secondary blocks. A low value for this parameter may affect performance because a large amount of processing time is required to write rollback information to these secondary blocks.

Rollback Clear (OpenVMS, UNIX, Windows)

The value of this parameter affects the behavior of screens when an error associated with a database transaction occurs during the Update phase.

If N is specified and the following conditions are met, the database transaction(s) is rolled back immediately and the data remains displayed on the screen:

- an error occurs during the Update phase
- transaction(s) active in the Update phase did not begin before the start of the Update phase

If Y is specified or if the above conditions are not met, QUICK enters Rollback Pending when an error is encountered.

Rollback Time-out

Allows you to limit the amount of time QUICK maintains blocking relational transactions while users try to determine the cause of a database error. This is useful in multi-user environments where one user's blocking transactions prevent other users from accessing the database. The Rollback Time-out is used when QUICK enters a Rollback Pending state. A value of 0 means there is no time-out.

The Rollback Time-out value is used twice. The initial time-out is used when the original database error is displayed. Typically, the error is displayed and prompts the user for authorization to proceed. If the time-out period expires, QUICK proceeds as if the user had responded, leaving the screen in a state to correct the error. If the time-out period expires, QUICK immediately rolls back any transactions and returns the user to the Action field of the first screen that started any of the rolled-back transactions.

The Rollback Time-out value must be less than the Terminal Time-out value so that the rollback is done before the terminal times out. If you attempt to enter a Rollback Time-out value that is greater than the Terminal Time-out value, QKGO issues the following message:

```
Rollback time-out should be less than terminal time-out.
```

Screen levels

Establishes the number of entries in a table that store information about active screens. The maximum number of screens that can be active concurrently is the largest number of levels in the screen hierarchy (that is, the longest leg). If the value is too low, QUICK issues an error message. Using too high a value wastes memory. This parameter also controls the maximum number of screen levels passed into memory for that user.

Screen table

Establishes the number of entries in a table used to store information about screens that are loaded, but that may no longer be active. When QUICK loads a screen, it copies the screen tables into memory (if space is available). Reloading from memory is much faster than reloading from the screen file on disk. The screen table allows QUICK to track whether a screen is available in memory. The minimum value is the maximum number of active screens (that is, screen levels).

Secondary Blocks (OpenVMS)

Determines the number of secondary blocks that are tracked. The Secondary Blocks parameter establishes the number of entries in a table used to store location information about blocks of rollback information and screen information. These secondary blocks may reside in virtual memory or in temporary files if the maximum paged memory allotment is exceeded. You may have to increase the value of this parameter if large amounts of rollback information must be stored or if there are many individual screens in the application. A secondary block must be added for each screen in the application, plus enough space to ensure that an update of any expected size can be copied into rollback buffer secondary blocks. If QUICK can't store rollback information, it issues an error message stating that page space is insufficient and abandons any update in progress, rolling back while complete information is still available.

Selection Size (OpenVMS)

Determines the size of a work area called the selection buffer. Unlike the other tables and work areas, there is a selection buffer in each screen. The selection buffer is used in Find mode and Select mode to store values accepted by the REQUEST verb for use during subsequent record retrieval.

The selection buffer must be slightly larger than the total length of the values that are to be stored, including selection values entered in Select mode. The size must be determined carefully, since every screen in the application system has its own area and uses up memory. Rarely do users need to select records based on more than one or two fields, so adding the lengths of the longest key field and the two other longest fields on the screen gives a reasonable value for that screen. By doing this calculation for all the screens in your application and taking the largest value (plus a few words), most requirements are met while using up as little memory as possible. If the value is too low, QUICK issues a warning message and ignores the selection value.

Terminal buffer

Establishes the size of the buffer used when input is read from the terminal and output is displayed on the terminal. The size of the buffer shouldn't be reduced below 512 bytes (the default) since performance may suffer. If memory is available, screen display performance may be improved by raising the terminal buffer value to the maximum. If you want to enter long text fields, this value must be as long as the longest text field.

OpenVMS: The terminal buffer must be at least 128 bytes less than the value of the system parameter MAXBUF. Furthermore, the terminal buffer should not exceed the process buffer quotas (BYTLM).

External Subroutines

Common area size

Establishes the size of an area in primary memory that's used by external subroutines. The address of the area is passed to an external subroutine if the PASSING option of the DO EXTERNAL verb is used. Results are unpredictable if the subroutine declares and uses an area larger than defined by this parameter. For more information about the DO EXTERNAL verb, see (p. 390) (**MPE/iX**), (p. 398) (**OpenVMS**), (p. 406) (**UNIX**), or **(p. 412)**(**Windows**).

Do Ext Save/Restore

Terminal settings are associated with your hardware, and exist outside of PowerHouse. Examples of terminal settings are WIDTH and WRAP|NOWRAP.

In this discussion, default settings refer to the terminal settings present before QUICK started, whereas QUICK settings refer to the terminal settings present in QUICK.

For operations that require terminal I/O, this parameter controls whether the terminal is initialized to the default settings for the duration of the external subroutine or subprocess, and then reset to the QUICK settings once the external subroutine or subprocess is complete. While some operations require this setting/resetting to maintain proper screen display characteristics, substantial time savings can be gained by overriding this setting/resetting when it is not required.

Do Ext Save/Restore defaults to N (No), which means that the terminal setting/resetting does not take place. This is because DO EXTERNAL subroutines normally don't require terminal I/O.

Run Cmd Save/Restore

For operations that require terminal I/O, this parameter controls whether the terminal is initialized to the default settings for the duration of the command or subprocess, and then reset to the QUICK settings once the command or subprocess is complete. While some operations require this setting/resetting to maintain proper screen display characteristics, substantial time savings can be gained by overriding this setting/resetting when it is not required.

Run Cmd Save/Restore defaults to Y (Yes), which means that the terminal setting/resetting does take place. This is because COMMANDs and RUN COMMANDs frequently require terminal I/O. However, if in your application they don't require terminal I/O, set this parameter to N (No) to avoid the extra overhead.

The Action Field Commands Screen

The Action Field Commands screen is used to rename QUICK Action field commands. To access the screen, enter option 04 "Action Field Commands" on the QKGO Construction and Maintenance Screen.

Action field commands can only be replaced with displayable characters. To map commands to terminal keys, use the screens accessed from the Terminal Interface Configuration Screen.

To change a value, select the field ID number and enter the new value. To change a value back to the default, enter <DEF> in the field.

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

The following characters may also be mapped in this screen:

10 Field mark character

A character indicating that the parameter for the command is the currently marked field.

12 Id char

On multi-record screens, a single record can be deleted by entering D followed by the Id character (-) and the Id number of the first field of the record in the Action field. To change the character that QUICK recognizes as the Id character, enter a new character followed by a space.

13 Id range

Use the Id range character (default /) to indicate a range of fields that are to be accessed sequentially. To change the character that QUICK recognizes as the Id range, enter a new character followed by a space.

23 Set soft keys

Set soft key labels on the screen. If you want to change the character sequence that QUICK currently recognizes (such as to the previous data command, \) enter the character (\) followed by a trailing space. A maximum of four characters can be used. Soft key labels are only applicable to terminal types that support them (for example, HP terminals).

The Action and Data Field Commands Screen

The Action and Data Field Commands screen is used to rename commands common to both the action and data fields. To access the screen, enter option 05 "Action and Data Field Commands" on the QKGO Construction and Maintenance Screen.

The Extended help and Help commands and the separator character can only be replaced with displayable characters. To map commands to terminal keys, use the screens accessed from the Terminal Interface Configuration Screen.

To change a value, select the field ID number and enter the new value. To change a value back to the default, enter <DEF> in the field.

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

The separator character may also be mapped in this screen:

03 Separator character

The separator character is used for separating several action or data field entries for rapid fire input. To change the character that QUICK recognizes, enter a new character followed by a space.

The Data Field Commands Screen

Use the Data Field Commands screen to rename data field commands, and to modify the generic retrieval character and the pattern match character.

To access the screen, enter option 06 "Data Fields Commands" on the QKGO Construction and Maintenance Screen.

To map commands to terminal keys, use the screens accessed from the Terminal Interface Configuration Screen. For more information about QUICK screen commands, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

To change a value, select the field ID number and enter the new value. To change a value back to the default, enter <DEF> in the field.

The following characters may also be mapped in this screen:

04 Generic search character

Use the character or characters preceding the Generic character in a data field as a partial value for selection. To change the character that QUICK recognizes as the Generic search character, enter a new character followed by a space. The QKGO generic search character overrides any generic retrieval character defined in the dictionary.

05 Pattern match character

Screen users can use pattern matching in Select mode on QUICK Screens to retrieve records. There are two formats depending on whether upper or lower case distinctions matter.

If upper and lower case distinctions do not matter, the screen user enters a pattern match character, by default, a percent sign (%) followed by the pattern string. To have upper and lower case characters selected exactly as specified in the pattern, enter two pattern match characters followed by the pattern string.

To avoid confusion in the rare instance when using the first format for a pattern string that itself begins with the % character, enter %!% followed by the rest of the pattern string.

To change the character that QUICK recognizes as the Pattern match character, enter a new character followed by a space.

08 Value selection list

Pop up a value selection list.

10 Enter Null Value

Sets the character which allows a null value to be entered.

11 Reverse toggle character (OpenVMS)

Changes the position of the cursor to the other end of the field and inverts the order of the characters that make up the string. Data entry is from right to left rather than the normal left to right.

The Dynamic Function Keys Screens

You can use the Dynamic Function Keys set of screens to:

- change dynamic function key parameters
- define dynamic function keys

To access the main DFK screen, enter Option 07 "Dynamic Function Keys" in the Action field of the Construction and Maintenance screen.

In your application, you may want to associate a set of commands with a single key or key combination. This can be achieved by using dynamic function keys. Dynamic function keys (DFKs) can be programmed in the following ways:

- using the KEY statement in QDESIGN
- using a set of screens in QKGO

When you define a DFK in QKGO, it is an application-wide default definition. A DFK defined in QDESIGN is associated with a screen. For more information about programming DFKs in QDESIGN, see (p. 156).

To change the value of a parameter in this screen, type its ID number.

OpenVMS:	When you define a DFK in either QDESIGN or QKGO, there is a default mapping of DFK1 to GOLD1, DFK2 to GOLD2, and so on.
MPE/iX, UNIX, Windows:	Values that are different from the default are displayed in halftone typeface.
OpenVMS:	Values that are different from the default are displayed in bold typeface.

Function Key Support Mode (MPE/iX, UNIX, Windows)

Specifies the type of function key support in QUICK. The available options are Disabled, Fixed Standard, and Dynamic.

If you choose Disabled, then function keys and labels are disabled, upon initialization, for the duration of the QUICK session.

If you choose Fixed Standard, then QUICK ignores your key definitions and uses the terminal's standard function key operation sets. If active, labels are displayed only when a user enters the KL command in QUICK. If the terminal has "loaded" function keys (function keys that are loaded with a character sequence using the terminal's firmware or some other method), then QUICK processes the character sequence as though it were entered by the keyboard.

If you choose Dynamic, then QUICK uses dynamic function key definitions assigned by QDESIGN KEY statements and QKGO. If active, QUICK labels are cleared during initialization and then they display the labels you define for each context and shift level.

Default: Fixed Standard

Function Keys

Specifies the maximum number of DFKs available for definition. Increasing the number of DFKs slightly increases QUICK's execution-time memory requirements.

Minimum: 1

Maximum: 32

Default: 8

Shift Levels

Specifies the maximum number of DFK shift levels that can be used. Increasing the number of shift levels slightly increases QUICK's execution-time memory requirements. For more information about shift levels, see (p. 156).

Minimum: 1

Maximum: 8

Default: 2

Labels Active (MPE/iX, UNIX, Windows)

Specifies whether or not QUICK displays function key labels on the terminal screen (for terminals that support function keys labels). If you specify No, you can reduce the amount of memory that QUICK uses, and the time QUICK takes to display a screen and change shift levels.

Default: Y

Bank Labels (MPE/iX, UNIX, Windows)

Specifies whether QUICK uses banked or unbanked labels to identify DFKs. If you specify unbanked labels, label strings may occupy both the top and bottom lines of the function key label, depending upon their length.

In order to provide label information that is context-specific in Field mode, QUICK must redisplay unbanked labels when the context changes. If QUICK uses banked labels, then each function key label simultaneously displays a pair of label strings, one on each line. The label string on the top line refers to the Action field context; the string on the bottom line refers to the Data field context.

Default: N

Lock Function Keys

Specifies whether or not QUICK disables terminal keys that allow the user to switch the mode of function keys from QUICK functions to local functions. Lock Function Keys are only applicable on terminal types that support them (for example, HP terminals).

Default: N

Save Function Keys (MPE/iX)

Specifies whether previously defined terminal function key definitions are kept. When N (No) is specified, previous function key definitions that are overridden by QUICK are lost upon exiting the product. Specifying Y (Yes) saves the definitions before running QUICK and restores them when exiting the product.

Default: N

Cancel All Block Transfer (MPE/iX)

Specifies whether BLOCKTRANSFER is disabled in Block Mode for DFK's. If you specify Y (yes), no BLOCKTRANSFERS are processed when you press the function keys. If your application runs over a network with data transmission costs, this option can reduce the characters transmitted. However, your operator will have to enter more keystrokes.

Default: N

Action Field Commands

EDIT	Calls the Edit DFK Definitions screen, which allows you to define or modify application-wide DFK definitions.

The Edit DFK Definitions Screen

Use the Edit DFK Definitions screen to make application-wide dynamic function key definitions. To access the screen, enter EDIT in the Action field on the Dynamic Function Keys screen.

A key defined in QKGO establishes the default function key for the application. Assigning DFK definitions using QKGO is similar to using the QDESIGN KEY statement. For more information, see (p. 156).

Actual definitions are entered via the DFK Definition Entry subscreen. When in Entry mode, you will automatically be transferred to this screen. You must specify:

- a unique key number
- a shift level (to a maximum of 8)
- a context of "Action", "Data", or "Action and Data"

To access a specific definition, type DEF-nn, where 'nn' is the ID number of the definition you want.

You must update your definition changes from this screen before you return to the main screen.

Key

Specifies the function key number to which the definition will apply. You map these logical function keys to physical (terminal) function keys by using the Terminal Interface Configuration screen.

OpenVMS: By default DFK1 is mapped to GOLD1, DFK2 to GOLD2, and so on.

Minimum: 1

Maximum: 32

Default: 1

Level

Specifies the shift level at which the definition will apply. DFKs can be defined to change the key shift level, allowing you to define more functions per key.

Minimum: 1

Maximum: 8

Default: 1

Context

Specifies in what context the DFK definition is operative. Choose either Action, Data, or Action and Data context.

Default: Action

Block mode Transfer (MPE/iX)

In Block mode, QUICK may do one of the following when a function key is hit:

(Y) Transfer screen data to internal buffers prior to performing actions.

(N) No data transfer. Perform the specified action without examining the screen.

Default: Y

Screen Label (MPE/iX, UNIX, Windows)

Specifies whether QUICK displays or does not display the label string regardless of the context. Specifying Y is useful with unbanked labels.

Default: N (the label is context-sensitive)

Label Hilites (MPE/iX, UNIX, Windows)

A group of four case-sensitive fields used to specify label highlighting on terminals that support highlighting. Each of the four letters represents a highlighting option: blinking (B), half intensity (H), inverse video (I), and underlining (U).

Options are active when entered in uppercase, and inactive when entered in lowercase. To activate an option, change the appropriate letter to the equivalent in uppercase. Lowercase deactivates the option. Any combination of case is valid in these options.

Default: bHIu (not blinking, half intensity, inverse video, no underlining)

Windows: Blinking and underlining can be set but are not supported in the Console.

Label (MPE/iX, UNIX, Windows)

Specifies that the label string to be displayed for the DFK on terminals that support function key labels.

Maximum: 16 characters for banked; 8 characters for unbanked.

Action Field Commands

DEF	Enter DEF-nn in the Action field (nn is the ID-number of a definition) to call the DFK Definition Entry screen. This screen is used to enter or edit the actions associated with the defined DFK.

The DFK Definition Entry Screen

Use this screen to enter the action for each dynamic function key defined on the Edit DFK Definitions screen.

To access this screen, enter DEF in the Action field on the Edit DFK Definitions screen. The screen allows you to edit function key definitions. Definitions appear in a form similar to a standard QDESIGN command list, that is, a list of commands separated by commas. A pair of square brackets ([ ]), called the cursor box, allows you to edit or append to the command list. An ellipsis (...) before or after the command list means that only part of the definition is displayed.

The cursor box is different from the real cursor. Use the cursor box to define your editing position. Then use the real cursor to enter data in another area, such as the Action field.

You can move the cursor box backward through a command list using the Previous (PREV) Action field command, and forward using the Next (NEXT) Action field command.

Defining the Dynamic Function Key's Action

To define the DFK's action, enter one or more command option codes in the Action field. The command options appear in the command list on the second line. Most of the command option codes look like QUICK screen commands with an apostrophe at the end. For more information, see the table on the next page.

Entering a Command Option at the end of a List

To enter a command option at the end of a list, position the empty cursor box at the end of the line and enter a command option code in the Action field. The command option appears in the position held by the cursor box and the box shifts one position to the right.

Inserting a Command Option into a List

To insert a command option into a list, position the cursor box over the command option that you want the new command option to precede, and then enter the new command option code in the Action field.

Deleting a Command Option

To delete a command option, position the cursor box over the command option to be deleted, and enter the Delete (DEL) Action field command. The command option disappears and the cursor box shifts one position to the right. To change a command option, delete the old one and then enter the new command option code.

When more command options have been entered than can fit in the window, an ellipsis appears before or after the list, which indicates that other command options precede or follow.

After entering or editing information on the Edit DFK Definitions Screen or on the DFK Definitions Entry Screen, enter an update command in the Action field of the Edit screen.

Command	Command Option Code	Action/Data field Equivalent
ACTIONBAR	BAR'	BAR in Action field
ACTIONFIELD	FLD'	
APPEND	A'	A in Action field
BACKOUT	BO'	^ in Data field
BACKUP	BU'	\ in Data field
DELETE	D'	D in Action field
DUPLICATE	DUP'	_ in Data field
ENTRY MODE	E'	E in Action field

Command	Command Option Code	Action/Data field Equivalent
EXTENDED HELP	EH'	?? command
FIELDMARK	MRK'	MARK in Action field
FIND MODE	F'	F in Action field
FIRST RECORD	FR'	FR in Action field GOLD_F (**OpenVMS**)
HELP	H'	? command
INFORMATION	I'	I command
LAST RECORD	LR'	LR in Action field GOLD_L (**OpenVMS**)
LIST	L'	L command
LIST ALL	LA'	L@ command
MODIFY	M'	M in Action field
NEXT	N'	N in Action field
NEXT DATA	ND'	<cr> in Action field
NEXT FIELD	NF'	<cr> in Data field
NEXT RECORD	NR'	NR in Action Field GOLD_CURSOR_DOWN (**OpenVMS**)
NULL	NUL'	key does nothing
PAGE DOWN	PD'	PD-n in Action field GOLD_J in Action field (**OpenVMS**)
PAGE UP	PU'	PU-n in Action field GOLD_K in Action field (**OpenVMS**)
POPUP FIELD	POP'	+ in a Data field
PREVIOUS DATA	PD'	\ in Action field
PREVIOUS RECORD	PR'	PR in Action field GOLD_CURSOR_UP (**OpenVMS**)
REFRESH	RF'	<ctrl G>
REFRESH ALL	RFA'	<ctrl W> <ctrl G><ctrl G> (**OpenVMS**)
RESTORE KEYS	K'	K in Action field
RESTORE LABELS	KL'	KL in Action field
RETURN	R'	^ in Action field
RETURN TO STOP	RS'	^^ in Action field

Command	Command Option Code	Action/Data field Equivalent
SCROLL DOWN	SD'	SD-n in Action field GOLD_D in Action field (**OpenVMS**)
SCROLL UP	SU'	SU-n in Action field GOLD_U in Action field (**OpenVMS**)
SELECT MODE	S'	S in Action field
SELECTBOX	SBX'	# in a Data field
SEPARATOR	SPR'	SEP char in Action field
SHIFT	SH'	
SHIFT TO	SHT'	
SKIP ALL	SA'	// in Data field
SKIP CLUSTER	SC'	/ in Data field
SKIP TO	ST'	/n in Data field
TOGGLE THREAD	T'	T command
UPDATE	U'	U in Action field
UPDATE NEXT	UN'	UN in Action field
UPDATE RETURN	UR'	UR in Action field
UPDATE STAY	US'	US in Action field

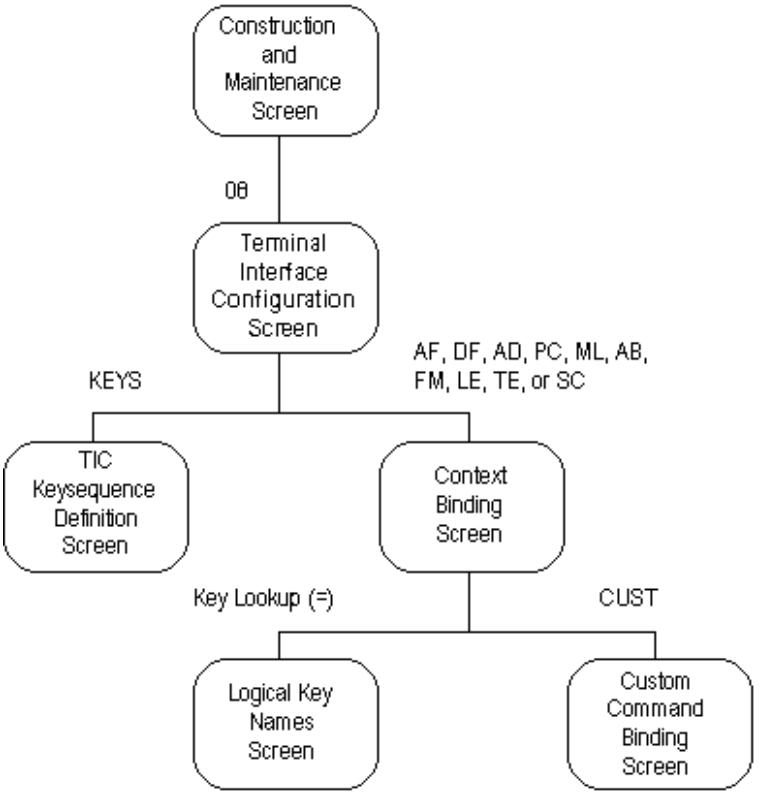
The Terminal Interface Configuration Screen

Use the Terminal Interface Configuration (TIC) Screen to configure the keyboard of various terminal types for specific applications. To access the screen, enter option 08, "Terminal Interface Configuration (TIC)" in the Enter command field of the Construction and Maintenance Screen.

From this screen, QUICK commands, designer procedures, and logical function keys can be mapped to terminal function keys, keypad keys, key combinations, and control characters.

The TIC System of Screens

For a selected terminal type, you can map QUICK commands and screen user commands using context binding subscreens. These subscreens are activated by entering the preceding two-letter context name. The KEYS command calls the TIC Keysequence Definition Screen so that you change logical key names. This lookup screen displays all the defined logical key names for a selected terminal.



Before you can modify any of the context binding screens, you must select a terminal group type from the Terminal Group section.

Modifying an Existing Terminal Interface Configuration

In most cases, you only need to modify one of the pre-loaded TICs. To modify an existing TIC, choose Find mode and enter the terminal type that you wish to configure.

When you modify the TIC for a specific terminal type, you modify all terminal types that are linked to that terminal interface configuration.

If you created a new QKGO file-set on the main screen (QKGO Construction and Maintenance Screen), there will be a default set of terminal interface configuration groups for VT100 and VT200 series keyboards (VT300 and VT400 series keyboards are grouped under VT200 series keyboards).

Creating your own Terminal Interface Configuration Group

To create a new TIC, choose Entry mode and enter the terminal type that you wish to configure.

QKGO will prompt you for an existing QKGO file-set in the "Copy from QKGO file" field to copy the terminal definition from. When a QKGO file-set is created from an existing QKGO file-set, the default terminal interface configuration is added to the QKGO file-set being created. This configuration can then be modified.

You can use Append mode to add terminal types to the displayed group.

The Command Binding Screens

The command binding screens are used to associate QUICK commands and screen operator commands with the keys of a keyboard. To call a command binding screen, enter one of the following commands in the Terminal Interface Configuration screen:

Command	Context Binding Screen
AB	Action Bar Commands
AD	Action and Data Commands
AF	Action Field Commands
DF	Data Field Commands
FM	Field Marking Commands
LE	Line Edit Commands
ML	Menu/List Commands
PC	Popup Commands
SC	System Commands
TE	Text Edit Commands

For example, if you select a VT200 terminal and enter

```
AF
```

then QUICK displays the Action Field Commands screen.

Each command binding screen is used to associate keyboard keys with QUICK commands. For each QUICK command, you can specify as many bindings as you like, but you cannot bind one key to more than one action. You can't add any new QUICK commands, but you can add DESIGNER procedure commands that you defined on the Custom Commands Mapping subscreen of the Action Field commands and the Action and Data Field Commands context binding screens.

To specify additional bindings, use either Append or Entry mode. You can either type in the field, or you can enter '=' or '*' in the field and choose the appropriate commands or key names from the pop-up screen.

MPE/iX, UNIX, Windows

In the command binding screens, key mnemonic codes take the form

- keyname (SHIFT_TAB)
- CNTL-X for control keys
- F1 through F8 for function keys
- F1_keyname for multi-key sequences

OpenVMS

In the command binding screens, key mnemonic codes take the form:

- KEY_PAD_x for key pad keys
- CNTL-X for control keys
- GOLD, PF2-PF4, F5-F20 for function keys
- GOLD_keyname for multi-key sequences

To delete a specific key mapping, enter a space in the field.

OpenVMS:

The following tables list the key sequences that cannot be defined in the TIC screen:

Control Sequence	Reason
control_C	user break
control_M	use the logical key [RETURN] for control_M
control_Q	reserved for use by the operating system
control_S	reserved for use by the operating system
control_T	reserved for use by the operating system
control_X	reserved for use by the operating system
control_Y	reserved for use by the operating system

VT Series Function Keys	Reason
F1	reserved for use by the operating system
F2	reserved for use by the operating system
F3	reserved for use by the operating system
F4	reserved for use by the operating system
F5	reserved for use by the operating system
F15	use the logical key [HELP] for F15
F16	use the logical key [DO] for F16

Command Bindings and Function Key Modes (MPE/iX, UNIX, Windows)

The function key mode is indicated in the top-left corner of the command binding screens. Function keys F1 through F8 are unavailable in Fixed Standard function key mode; however, F1 may be used as the first key in a key sequence such as F1_M. Key sequences that start with F1 are unavailable in Dynamic function key mode.

You can bind a command to a key that is unavailable in the current function key mode, but the binding is inactive. A warning message is issued, and the key mnemonic code is displayed in halftone typeface. The command binding becomes active only when you choose a function key mode in which the key is available.

Action Field Commands

KEYS	New keys can be specified by entering KEYS in the TIC screen. The TIC Keysequence Definition Screen, which is used to maintain the logical key names and their escape sequences is displayed.
LIST	This command produces a listing of the complete set of key bindings for a TIC. The report will be stored in the file called QKGOTIC (**MPE/iX**) or qkgotic.txt (**OpenVMS, UNIX, Windows**).
RESO (UNIX, Windows)	Calls a procedure that creates a flat TIC resource file for a specified terminal type. The TIC resource file contains similar information to the three indexed files of a QKGO file-set (filenameb.idx, filenamek.idx, and filenamet.idx). Whenever you make modifications to an existing QKGO file-set, you must recreate the corresponding TIC resource file so that it contains the latest information. TIC resource files are useful when a QKGO file-set resides on another host and the indexed files cannot be accessed by the C-ISAM library system. In this instance, you can redirect QUICK to get the information from the TIC resource file by setting the AXTICRS environment variable to reference it, as in `setenv AXTICRS \    $PH_QKGO_LOCATION/resource/my_tics` TIC resource files may also improve performance because they're faster to read than indexed files.

The Color Display Attributes Screen (OpenVMS)

You use the Color Display Attributes screen to change the color characteristics for a given terminal interface configuration.

This screen is accessed by selecting CD on the Terminal Interface Configuration screen.

Mnemonics for colors are entered in the appropriate fields.

You can see a list of currently supported colors for each field by entering an equal sign (=) in the mnemonic field. The AS_IS option tells QUICK not to change the terminal attributes.

The Custom Commands Binding Screen

The Custom Commands Binding screen is used to map QUICK commands to logical keys where the QUICK commands can take parameters.

This screen can be accessed by entering CUST in the Action field in either the Action Field Commands or the Action and Data Field Commands Context binding screen.

For example, you can map physical keyboard keys to commands such as D-1 (delete the first field) or ?-mark (display the help for the currently marked field).

01 Command

From the selectbox, select one of the following QUICK commands:

Delete	Designer	Dynamic Function Key
Extended Help	Field Page Down	Field Page Up
Field Scroll Down	Field Scroll Left	Field Scroll Right
Field Scroll Up	Help	Id

02 Logical Key Name

After you specify the command you wish to change, you must also specify the logical key name that will map to the physical key dependent on the terminal, console, or terminal emulator. An "=" sign will display the valid key names.

Custom Command Binding Options

Depending on the command that you choose in Field 01, you are prompted for one or more options.

03 Designer Procedure

Enter the name of the DESIGNER procedure to be mapped to the keyboard key. If you select this option from the Enter command field, note that you can also change the parameter type.

04 Dynamic Function Key

Enter the number of the Dynamic Function Key.

Minimum: 1

Maximum: 32

05 Parameter type

From the selectbox, choose one of the following parameter types:

ID	ID Range	Mark
None	Prompt	

06 Id Values

Enter the ID of the field to use as a parameter for the command. If you choose the parameter type ID in the previous option, QKGO prompts you to enter a single ID value. If you choose ID Range, QKGO prompts you to enter a range of ID values.

Minimum: 1

Maximum: 999

Mark or Prompt doesn't require an ID parameter. Mark tells QUICK to supply the currently marked field as a parameter to the command. Prompt tells QUICK to prompt the user for the field parameter.

Modifying TIC Files

Introduction

The TIC mappings are saved to separate TIC files, one per terminal type. We recommend that you use the QKGO TIC screen to generate and maintain the TIC files. As they are text files, however, they can also be edited.

A different TIC file is required for each terminal type because the key codes are different. A default TIC file is provided in the qkgo\resource installation directory. This TIC file contains the key codes for all the keys and key combinations that can be mapped as well as a set of default key mappings that correspond to the PCANSI terminal type as defined for PowerHouse on UNIX. It can be modified as required to satisfy application requirements.

To use a TIC file, use either the PHTICRS or AXTICRS environment variable to point to the TIC file. It is recommended that you make a copy of the default TIC file before modifying it.

The Format of a TIC File

A TIC file consists of two sections. The Key Section appears first and defines the key mnemonics and relates keyboard key codes to those mnemonics. The Key Section defines only those keys or key combinations that are available to be mapped to QUICK commands, so the letters, numbers, and special characters are not described. The Binding Section maps key mnemonics to QUICK commands. It also specifies the context and the QKGO function key setting in which the key or key combination can be used.

Within the TIC file, a semicolon can be used to add comments. Anything after the semicolon on that line is ignored. Blank lines are ignored.

The Key Section

The format of each line in the Key Section is as follows:

- The word "key", not in quotes, followed by one or more spaces.
- A period (.) followed by one or more spaces.
- The key mnemonic, for example "F1", not in quotes, followed by one or more spaces.
- A colon (:) followed by one or more spaces.
- The octal keyboard code in quotes, for example "\000\073". This is the code that QUICK receives from the key press.
- A colon (:) followed by one or more spaces.
- One of three words, not in quotes, that relates the key availability to the QKGO function key setting. The word "always" means that the key is always available regardless of the QKGO setting. The word "fixed" means that the key is available if the QKGO setting is for fixed standard. The word "dfk" means that the key is available if the QKGO setting is for dynamic function key support. It is recommended that the word be left as "always" and that the Bindings section be used to specify which keys are available in which context.

A sample line from the Key Section looks like this:

```
key . F1            :  "\000\073"            :  always
```

This line defines the key "F1" as being always available. The code returned by the key press is "\000\073".

The default TIC file includes all of the key and key combinations that can be mapped. This section should not be changed if you modify the TIC file.

Note that keys such as the Enter key (CR), Tab key (TAB), and control-letter key combinations should be used so as not to conflict with the normal use of those keys.

Windows: On the PC keyboard, the shift-tab key combination does not return a key code and so cannot be used to reverse tab as is typically done in Windows. Instead, use control-tab.

The Binding Section

The format of each line in the Binding Section is as follows:

1. One of ten words, not in quotes, that specifies the QUICK context in which the key is valid. The words are:
   - action - for action context,
   - data - for data context,
   - action_data - for action and data context,
   - popup - for popup messages,
   - menu - for drop-down menus and select boxes,
   - actionbar - for action bars
   - line_edit - for single line field editing
   - fieldmark - for field mark usage,
   - text_edit - for multi-line field editing, and
   - system - for commands applicable at any time.

   Although not a requirement, it is recommended that the lines in each group be kept together for ease of modification.
2. A colon followed by one or more spaces.
3. One of three words, not in quotes, that relates the key availability to the QKGO function key setting. The word "always" means that the key is always available regardless of the QKGO function key setting. The word "fixed=0" means that the key is available if the QKGO setting is for fixed standard. The word "dfk" means that the key is available if the QKGO setting is for dynamic function key support. If "dfk" is specified, it must be immediately followed by an equals sign and a number, with no intervening spaces, as in "dfk=1". The number must be between 1 and 32 and relates to the number of dynamic function keys allowed in QKGO and the number on the KEY statement in QDESIGN.
4. A period followed by one or more spaces.
5. The key mnemonic, not in quotes, as defined in the Key Section, followed by one or more spaces.
6. An equals sign followed by one or more spaces.
7. The QUICK command number. The available commands are shown in tables in the next section.

A sample line from the Binding Section looks like this

```
action      : fixed=0 . F7 = 0          ; Entry Mode
```

This line specifies that pressing the "F7" key in action context will tell QUICK to start Entry Mode if the function key setting is Fixed Standard.

Another example, that is useful in Panel processing is

```
data        : always  . F8 = 2          ; Backout
```

This line set the "F8" key to the backout command regardless of whether the function key setting is disabled, fixed standard, or dynamic. It overrides any setting, either in QKGO or by the KEY statement. However, any label specified would be used.

The default TIC file provided with the PowerHouse for Windows install includes action and data context key bindings for Fixed Standard and Dynamic QKGO function key settings that become active when the corresponding setting is made in QKGO.

The bindings for popup, menu, actionbar, line_edit, fieldmark, text_edit, and system are set to be available always, that is, regardless of the QKGO setting.

If you want bindings for action, data, and action_data available regardless of the QKGO setting, change the word fixed=0 to always. Note that you won't be able to use dynamic function keys. If you want to use function keys, you should change the QKGO setting to use Fixed Standard or Dynamic as opposed to changing the binding word.

To change a binding, locate the fixed=0 line with that command and change the key mnemonic. Not every QUICK command is represented in the default TIC file. To add a binding or command, copy a line in the section where you want to add the line and change the key mnemonic and the QUICK command number. Ensure that keys are not duplicated or results will be unpredictable.

The default TIC file includes bindings for 32 dynamic function keys.The actual number available will depend on the setting in QKGO and the KEY statements specified in QDESIGN.

QUICK Commands

The following tables show the QUICK command numbers that can be mapped in the Binding Section. Also shown is the command, keyword and default command mnemonic for action and data context commands.

Action Context Commands (action)

Command	Keyword	Default Mnemonic	Command number
Entry Mode	ENTRY [MODE]	E	0
Find Mode	FIND [MODE]	F	1
Select Mode	SELECT [MODE]	S	2
Next Data	NEXT DATA	<CR>	3
Previous Data	PREVIOUS [DATA]	\	4
Return - Prev Screen	RETURN	^	5
Return to Stop	RETURN TO STOP	^^	6
Update	UPDATE	U	9
Update Stay	UPDATE STAY	US	10
Update Return	UPDATE RETURN	UR	11
Delete	DELETE	D	12
Set Function Keys	RESTORE KEYS	K	16
Set Key Labels	RESTORE LABELS	KL	17
Update Next	UPDATE NEXT	UN	18
Append	APPEND	A	19
Next	NEXT	N	20
Return to Start	RETURN TO START	^^^	21
Action Bar	ACTIONBAR	BAR	22
Action Field	ACTIONFIELD	ACT	23
Field Mark	FIELDMARK	MARK	24
Field Page Up	PAGEUP	PU	25
Field Page Down	PAGEDOWN	PD	26

Command	Keyword	Default Mnemonic	Command number
Field Scroll Up	SCROLLUP	SU	27
Field Scroll Down	SCROLLDOWN	SD	28
Modify	MODIFY	M	29
Office Interrupt	INTERRUPT	IT	31
Designer	DESIGNER	name	34
Id	ID	n	35
Previous Data	PREVIOUS DATA	\	40
System	SYSTEM	:/$/!shell	41
Block Mode Toggle			43
Cache - Prev Records	PREVIOUS RECORD	PR	48
Cache - Next Records	NEXT RECORD	NR	49
Cache - First Record	FIRST RECORD	FR	50
Cache - Last Record	LAST RECORD	LR	51

Data Context Commands (data)

Command	Keyword	Default Mnemonic	Command number
Duplicate	DUPLICATE	_	0
Backup Field	BACKUP	\	1
Backout	BACKOUT	^	2
Return to Action	SKIP ALL	//	3
Skip Cluster	SKIP CLUSTER	/	4
Reverse Input Toggle		|	7
Popup Toggle	POPUP	+	8
Select Box	SELECTBOX	#	9
Enter Null Value			10

Action and Data Context Commands (action_data)

Command	Keyword	Default Mnemonic	Command number
Help	HELP	?	-4
Extended Help	EXTENDED HELP	??	-5

Command	Keyword	Default Mnemonic	Command number
Dynamic Function Key			-6
List	LIST	L	13
List All	LIST ALL	L@	14
Information	INFORMATION	I	15
Toggle Thread	TOGGLE THREAD	T	30
Refresh Screen	REFRESH	<Ctrl-G>	163
Refresh All	REFRESH ALL	<Ctrl-W>	164
Field Help	FIELD HELP	?	213
Extended Field Help	EXTENDED FIELD HELP	??	214

Popup Message Commands (popup)

Command	Command Number
Scroll Up	100
Scroll Down	101
Page Up	102
Page Down	103
Cancel	104
Accept	105

Menu and Selectbox Commands (menu)

Command	Command Number
Move Up	120
Move Down	121
Page Up	122
Page Down	123
Cancel	124
Accept	125

Action Bar Commands (actionbar)

Command	Command Number
Accept	130

Command	Command Number
Cancel	131
Previous Option	132
Next Option	133

Line Edit Commands (line_edit)

Command	Command Number
Input Completion	-8
Single Line Recall	42
Insert	110
Delete Word	111
Delete Line	112
Move - End of Line	113
Move - Start of Line	114
Move Left	115
Move Right	116
Delete Char	117
Delete Previous Char	118
Cancel Input	128
Prev Field in Panel	180
Next Field in Panel	181

Field Mark Commands (fieldmark)

Command	Command Number
Previous Option	145
Next Option	146
Accept	147
Cancel	148

Text Edit Commands (text_edit)

Command	Command Number
Input Completion	-8
Single Line Recall	42

Command	Command Number
Insert	110
Delete Word	111
Delete Line	112
Move - End of Line	113
Move - Start of Line	114
Move Left	115
Move Right	116
Delete Char	117
Delete Previous Char	118
Cancel Input	128
End of Paragraph	160
Move Up	161
Move Down	162
Page Up	166
Page Down	167
Delete - End of Line	168
Prev Field in Panel	180
Next Field in Panel	181

System Commands (system)

Command	Keyword	Default Mnemonic	Command number
Refresh Screen	REFRESH	<Ctrl-G>	163
Refresh All	REFRESH ALL	<Ctrl-W>	164
Refresh Line	REFRESH LINE		165
Exit			999

QUICK Initialization File

To specify most QKGO settings, you can alternatively use the QUICK initialization file (QKI). The QUICK initialization file is a text file which you can create with a text editor. The extension for the QUICK initialization file must be .qki (**OpenVMS, UNIX, Windows**). On MPE/iX, the QUICK initialization file doesn't have an extension and the file format determines whether the file is a QUICK initialization file or a QKGO file.

TIC settings are not contained in the QKI file as they are available in the TIC file.

Here is a complete default QKI file:

```
[General Settings]
FIRST_SCREEN=
DICTIONARY=

[Execution Time Parameter Values]
APPLICATION_LINES=48
BLOCK_MODE_RETRIES=5
COMMON_AREA_SIZE=1
DRIVER=QKDRIVER
ERROR_RECALL=N
EXPRESSION_SIZE=400
EXTERNAL_SUBROUTINES=1
FIELD_TERMINATORS=M
INPUT_MODE=C
LOCK_ATTEMPTS=16
LOCK_RETRY_INTERVAL=2
LOCK_UNCONDITIONAL=N
MAXIMUM_PAGED_MEMORY=2048
HORIZONTAL_LINE=*
HP_TERM_HAS_LDW=N
LOCK_REQUEST_WAIT=30
LOCK_MESSAGE_WAIT=5
ROLLBACK_CLEAR=Y
ROLLBACK_TIMEOUT=0
ROLLBACK_BUFFER=1024
SCREEN_LEVELS=5
SCREEN_SECTION=G
SECONDARY_BLOCKS=32
SELECTION_SIZE=20
SCREEN_TABLE=15
TERMINAL_BUFFER=512
TERMINAL_TIMEOUT=0
UPSHIFT_ACTIONS=Y
MAX_THREAD_COUNT=3
DO_EXTERNAL_SAVE=N
RUN_COMMAND_SAVE=Y
VERTICAL_LINE=*
LINE_INTERSECTION=*
FUNCTION_KEY_MODE=1
FUNCTION_KEYS=8
SHIFT_LEVELS=2
LABELS_ACTIVE=Y
BANK_LABELS=N
LOCK_FUNCTION_KEYS=N

[Action Field Commands]
APPEND=A
ACTION_BAR=BAR
ACTION_FIELD=ACT
BLOCK_MODE=B
CHANGE_SEPARATOR=SEP
CHARACTER_MODE=C
DELETE=D
ENTRY_MODE=E
FIELDMARK=MARK
```

```
FIELDMARK_CHAR==
FIND_MODE=F
ID_CHAR=-
ID_RANGE_CHAR=/
INFORMATION=I
LIST_ALL=L@
LIST=L
MODIFY=M
NEXT_PRIMARY=N
PREVIOUS_DATA=\
SELECT_MODE=S
TOGGLE_THREAD=T
FIRST_RECORD=FR
LAST_RECORD=LR
NEXT_RECORD=NR
PREVIOUS_RECORD=PR
RETURN=^
RETURN_TO_STOP=^^
RETURN_TO_START=^^^
PAGE_UP=PU
PAGE_DOWN=PD
SET_KEYS_AND_LABELS=KL
SET_SOFT_KEYS=K
SCROLL_UP=SU
SCROLL_DOWN=SD
UPDATE=U
UPDATE_NEXT=UN
UPDATE_RETURN=UR
UPDATE_STAY=US

[Action and Data Field Commands]
EXTENDED_HELP=??
HELP=?
SEPARATOR_CHAR=;

[Data Field Commands]
BACKOUT=^
BACKUP=\
DUPLICATE=_
GENERIC_SEARCH_CHAR=@
PATTERN_MATCH_CHAR=%
SKIP_CLUSTER=/
SKIP_ALL=//
VALUE_SELECTION_LIST=#
POPUP=+
ENTER_NULL_VALUE=~
REVERSE_TOGGLE_CHAR=|
```

For more information about the value settings, see the preceding sections of this chapter.

Chapter 7: QDESIGN Procedures

Overview

This chapter provides a detailed reference of QDESIGN procedures. For each procedure you'll find

- syntax summaries
- detailed syntax descriptions
- detailed discussion of the procedure
- examples

QDESIGN Procedure Summary

The following table lists QDESIGN procedures and briefly describes what they do.

Procedure	Default Generated?	Purpose
APPEND	Yes	Controls Append processing for PRIMARY and DETAIL files.
BACKOUT	No	Performs processing when the user backs out of an entry sequence or fails to update a change.
DELETE	Yes	Marks one or more of the current data records on the screen for deletion.
DESIGNER	No	Creates a designer-defined Action field command.
DETAIL DELETE	Yes	Marks for deletion specific data records in a DETAIL file, and any files that occur with it.
DETAIL FIND	Yes	Controls data record retrieval for a DETAIL file and any files that occur with it.
DETAIL POSTFIND	No	Performs processing after successful completion of a DETAIL FIND procedure.
EDIT	No	Performs editing on a value entered in a named field.
ENTRY	Yes	Performs the standard entry sequence for the screen.
EXIT	No	Performs processing just before returning to a higher-level screen.
FIND	Yes	Retrieves data records as indicated by the PATH procedure.
INITIALIZE	No	Performs processing when the screen is initiated from a higher-level screen.
INPUT	No	Performs data conversion for a value entered in the named field prior to any editing.

Procedure	Default Generated?	Purpose
INTERNAL	No	Creates an internal subroutine in the QDESIGN procedural language.
MODIFY	Yes	Controls QUICK processing in CHANGEMODE or CORRECTMODE. Panel mode only.
OUTPUT	No	Performs data conversion for a value in the named field prior to the display of the value.
PATH	Yes	Establishes the method of record retrieval for the FIND procedure.
POSTFIND	No	Performs processing after successful completion of the FIND procedure.
POSTPATH	No	Performs processing following successful completion of the PATH procedure and just before the FIND procedure.
POSTSCROLL	No	Performs processing after the occurrence window has scrolled but before it is re-displayed.
POSTUPDATE	No	Performs processing after successful completion of the UPDATE procedure.
PREENTRY	No	Performs processing at the beginning of the entry sequence.
PRESCROLL	No	Performs processing before scrolling occurs.
PREUPDATE	No	Performs processing prior to the UPDATE procedure.
PROCESS	No	Performs processing after a new or changed value is entered in the named field.
SELECT	Yes	Controls SELECTMODE processing.
UPDATE	Yes	Controls update processing.

APPEND

Controls Append processing for PRIMARY and DETAIL files.

Syntax

PROCEDURE APPEND

Discussion

QDESIGN generates the APPEND procedure

- for a PRIMARY file with multiple occurrences (provided that the NOAPPEND option isn't specified on the FILE statement)
- for a DETAIL file

QDESIGN doesn't create an APPEND procedure if you write an ENTRY or APPEND procedure in the screen design. A designer-written ENTRY procedure must include the PERFORM APPEND verb to provide Append processing. If you don't want Append capabilities for the PRIMARY file of a screen, add the NOAPPEND option to the FILE statement.

Note: For information about verb and procedure compatibility, see (p. 239).

Implication of Modifying the APPEND Procedure

If you modify the default procedures that QDESIGN generates, the APPEND procedure must be located before the ENTRY procedure that contains the PERFORM APPEND verb.

Use caution when modifying the APPEND procedure because it's used by QUICK for Append processing in both Entry mode and Find mode.

When the APPEND Procedure is Initiated

The APPEND procedure is initiated when

- the PERFORM APPEND verb is encountered in the ENTRY procedure
- the PERFORM APPEND verb is encountered in a FOR MISSING loop in the MODIFY procedure
- the screen is in Entry or Find mode and the QUICK screen user enters an Append command
- data retrieved from a PRIMARY or DETAIL file is changed and updated by an Update command

When data entered on the screen using Append processing is sent to the file, the APPEND procedure continues to execute so that the QUICK screen user can make further entries. The data is sent to the file

- when the QUICK screen user enters any of the update commands
- as soon as the screen is full, when the AUTOUPDATE option is specified for the screen

Error Handling in the APPEND Procedure

If an error occurs during the execution of an APPEND procedure, QUICK performs the following steps:

1. QUICK first tries to back up to the last ACCEPT or PROMPT verb or to the last BLOCK TRANSFER control structure within the APPEND procedure
2. QUICK backs up to the last ACCEPT or PROMPT verb, or to the last BLOCK TRANSFER control structure in the ENTRY procedure if
   - there is no ACCEPT or PROMPT verb, or BLOCK TRANSFER in the APPEND procedure, and
   - the APPEND procedure is executing in response to a PERFORM APPEND verb in the ENTRY procedure
3. QUICK prompts at the Action field (without having updated)

- if there is no ACCEPT or PROMPT verb, or BLOCK TRANSFER in the ENTRY procedure
- if the APPEND procedure is executing in response to an Append or Update command
- in response to a specified AUTOUPDATE option

During the execution of the APPEND procedure, the ENTRYMODE predefined condition is true.

Append Processing and the CLUSTER Statement

In the generated source code for a QUICK screen that has a DETAIL file or a PRIMARY file with multiple occurrences, there is always a CLUSTER statement preceding the FIELD statements. The CLUSTER statement is essential; without it, the APPEND and UPDATE procedures don't work correctly. If you save a generated screen specification, don't delete the CLUSTER statement, even if the PRIMARY or DETAIL file occurs only once.

Examples

Append processing is available by default on a screen where a PRIMARY record-structure with multiple occurrences is declared. QDESIGN generates an APPEND procedure and an ENTRY procedure, as in

```
> SCREEN BRANCH
> FILE BRANCHES OCCURS 10
> CLUSTER OCCURS WITH BRANCHES
>   FIELD BRANCH OF BRANCHES REQUIRED NOCHANGE &
>       LOOKUP NOTON BRANCHES
>   FIELD BRANCHNAME OF BRANCHES
> CLUSTER
> BUILD LIST

> PROCEDURE APPEND
>   BEGIN
>     ACCEPT BRANCH OF BRANCHES
>     ACCEPT BRANCHNAME OF BRANCHES
>     END
> PROCEDURE ENTRY
>   BEGIN
>     FOR BRANCHES
>       BEGIN
>         PERFORM APPEND
>         END
>     END
```

DETAIL Files and the APPEND Procedure

The QUICK screen user can enter one data record for EMPLOYEES and as many SKILLS data records as necessary on the screen described by the following QDESIGN statements.

This screen creates a one-to-many relationship between EMPLOYEES and SKILLS:

```
> SCREEN EMPLOYS
> FILE EMPLOYEES
> FILE SKILLS DETAIL OCCURS 16
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF
EMPLOYEES.
> FIELD EMPLOYEE OF EMPLOYEES REQUIRED NOCHANGE   &
>     LOOKUP NOTON EMPLOYEES
> ALIGN (1,4,21) (,,45)
> FIELD FIRSTNAME OF EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES
> SKIP 2
> ALIGN (1,4,12)
> CLUSTER OCCURS WITH SKILLS FOR 2,36
>   FIELD SKILL OF SKILLS
> CLUSTER
> BUILD LIST
SKILLS accessed via EMPLOYEE.
```

Entering Data with Append Processing

If the QUICK screen user enters a Skip All command when the screen is not full, PowerHouse assumes that the user has entered as much data as desired. In such cases, after updating, QUICK clears both the PRIMARY file records and all DETAIL file records. QUICK then prompts for the next PRIMARY file record. Entering an Update Stay command doesn't clear the screen of data.

If a full screen is updated, QUICK assumes that the user still has more DETAIL records to enter for the current PRIMARY record. As a result, the PRIMARY data record remains on the screen, the DETAIL records are updated and cleared, and the user is prompted in the first DETAIL occurrence for new entries. If an update is triggered because an AUTOUPDATE option was included on the SCREEN statement, the AUTOUPDATE option performs in the same manner as an update action.

If an Update Next command is entered, then an update is done, all fields are cleared, and the user is first prompted for a new PRIMARY record, and then for DETAIL entries. QDESIGN generates two procedures to control the entry sequence in the previous screen design as follows:

```
> PROCEDURE APPEND
>   BEGIN
>     ACCEPT SKILL OF SKILLS
>     END
>
> PROCEDURE ENTRY
>   BEGIN
>     ACCEPT EMPLOYEE OF EMPLOYEES
>     ACCEPT FIRSTNAME OF EMPLOYEES
>     ACCEPT LASTNAME OF EMPLOYEES
>     FOR SKILLS
>       BEGIN
>         PERFORM APPEND
>         END
>   END
```

The entry sequence for the DETAIL file is ended with a Skip All command. The entry sequence for the DETAIL file is restarted with an Append command.

BACKOUT

Performs processing when the user backs out of an entry sequence or fails to update a change.

Syntax

PROCEDURE BACKOUT

Discussion

When the BACKOUT Procedure is Invoked

The BACKOUT procedure is invoked if the user (while entering, correcting, or changing data) enters a Backout command when the cursor is positioned in a field, or if the user leaves a screen without updating. When the user enters a Backout command during Entry, Correct, or Change mode, QUICK warns that the data will be lost. The user is asked to confirm that this is acceptable. If the user confirms the action, the cursor returns to the Action field or, in Correct mode, begins the next Entry sequence. The **confirmer** program parameter changes QUICK's default action in this case. For more information about the **confirmer** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

When leaving a slave screen, the BACKOUT procedure is only invoked when data local to the screen is changed and not updated. If data in a received MASTER file is changed, the BACKOUT procedure is not invoked because the slave screen is considered an extension of the calling screen and the file originated there. However if a locally declared DESIGNER file is changed and not updated, the BACKOUT procedure will execute upon leaving the screen.

Within the BACKOUT procedure, you can undo the update that might have been performed by other designer-specified procedures part way through Entry, Correction, or Change phases.

Note: For information about verb and procedure compatibility, see (p. 239).

When to Use a BACKOUT Procedure

The BACKOUT procedure is used to undo PUT verbs (for non-relational records) that were not performed in the standard UPDATE procedure. If all file updates are centralized in the UPDATE procedure, QUICK automatically handles the rollback for these files and a BACKOUT procedure is not necessary.

To avoid the need for the BACKOUT procedure, limit the use of PUT verbs to the UPDATE procedure or use the RECOVERABLE option on the PREUPDATE, POSTUPDATE, or INITIALIZE procedures.

If you add the RECOVERABLE option on the PREUPDATE, POSTUPDATE, or INITIALIZE procedures, any BACKOUT procedures that you have coded to handle rollbacks must be reviewed to ensure that a "double rollback" is not performed.

During the execution of this procedure, one of the predefined conditions, ENTRYMODE, CHANGEMODE, or CORRECTMODE, is true.

Error Handling During the BACKOUT Procedure

If an error occurs during the execution of this procedure, unpredictable results may occur.

Example

This example demonstrates the use of the designer-written BACKOUT procedure. The BATCH screen includes a control file, BATCHNEXT, that is used to track the next available batch number which is to be assigned to a new BATCHES record. If the screen user fails to update new or changed data, the BACKOUT procedure is initiated and the current batch number is written to the BATCHBAD file, along with the time and date.

```
> SCREEN BATCH
>
> FILE BATCHES PRIMARY
```

```
> FILE BATCHNEXT DESIGNER
> FILE BATCHBAD DESIGNER
>
> FIELD BATCHNO OF BATCHES DISPLAY
> FIELD BATCHCOUNT OF BATCHES
> FIELD BATCHTOTAL OF BATCHES
> FIELD TRANSCOUNT OF BATCHES
> FIELD TRANSTOTAL OF BATCHES
> PROCEDURE ENTRY
>   BEGIN
>     ACCEPT BATCHCOUNT OF BATCHES
>     GET BATCHNEXT USING 1
>     LET BATCHNO OF BATCHES = NEXTBATCHNO OF BATCHNEXT
>     LET LASTBATCHNO OF BATCHNEXT = NEXTBATCHNO &
>          OF BATCHNEXT
>     LET NEXTBATCHNO OF BATCHNEXT = NEXTBATCHNO &
>          OF BATCHNEXT + 1
>     PUT BATCHNEXT AT 1
>
>     DISPLAY BATCHNO OF BATCHES
>     ACCEPT BATCHTOTAL OF BATCHES
>     ACCEPT TRANSCOUNT OF BATCHES
>     ACCEPT TRANSTOTAL OF BATCHES
>     END
>
> PROCEDURE BACKOUT
>   BEGIN
>     IF NEWRECORD OF BATCHES
>       THEN BEGIN
>         LET BATCHNO OF BATCHBAD = &
>           BATCHNO OF BATCHES
>         LET DATESTAMP OF BATCHBAD = SYSDATE
>         LET TIMESTAMP OF BATCHBAD = SYSTIME/100
>         PUT BATCHBAD
>         END
>      END
>
> BUILD
```

DELETE

Marks one or more of the current data records on the screen for deletion.

Syntax

PROCEDURE DELETE

Discussion

The DELETE procedure marks data records for deletion in PRIMARY and associated DETAIL and SECONDARY files. These data records are physically deleted when a subsequent update command is issued.

A screen cannot perform the DELETE activity unless a DELETE procedure exists.

Note: For information about verb and procedure compatibility, see (p. 239).

The Default DELETE Procedure

QDESIGN constructs a default DELETE procedure for each screen that has a PRIMARY file. QDESIGN also constructs a default DETAIL DELETE procedure for each screen design that contains a DETAIL file.

Disabling Deletion for Specific Record-structures

The NODELETE option of the FILE statement suppresses the inclusion of a DELETE verb for that record-structure in the DELETE and DETAIL DELETE procedures.

User-Defined DELETE Procedures

If you construct a DELETE procedure, you should include only DELETE verbs in it. The DELETE verb marks data records for deletion at a later update. As a result, fields on the screen that are related to such data records are blanked. Unless they specifically occur with another record-structure, fields for temporary items are treated as if they were in the PRIMARY record-structure and are also blanked.

The DELETE procedure works for data records in both the Correction phase of Entry mode or in the Change phase of Find mode. The procedure addresses one occurrence of the PRIMARY record-structure, even if the record-structure occurs more than once on the screen. This is because the DELETE procedure is used in response to the Delete command, which can address one, some, or all of the occurrences.

When the DELETE Procedure is Initiated

The procedure is initiated when a Delete command is entered.

Deleting Specific Occurrences of a File

If a QUICK screen user initiates the DELETE procedure on a screen with a repeating PRIMARY file, the DELETE procedure is invoked for each occurrence of the PRIMARY file. If the Delete command includes a number or a range of numbers (for example, D-2 or D-2/5), the DELETE procedure is invoked only for the specific occurrences.

The DELETE procedure usually addresses only one occurrence of the file in question. (A generated DELETE procedure uses a FOR control structure to address a DETAIL or SECONDARY file, if the DETAIL or SECONDARY file occurs more than once and the PRIMARY file occurs only once.) If you write your own DELETE procedure, exercise caution when you use a FOR control structure to address more than one occurrence.

During the execution of this procedure, one of the predefined conditions, CHANGEMODE or CORRECTMODE, is true.

Error Handling in the DELETE Procedure

If an error occurs during the execution of this procedure, the rest of the procedure is skipped and QUICK prompts at the Action field.

Examples

The following example illustrates the DELETE procedure for a repeating PRIMARY record-structure.

```
> SCREEN SKILL RECEIVING EMPLOYEES
> FILE EMPLOYEES MASTER
> FILE SKILLS OCCURS 10
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF EMPLOYEES.
> CLUSTER OCCURS WITH SKILLS
> FIELD SKILL OF SKILLS REQUIRED NOCHANGE
> BUILD LIST
.
.
.
> PROCEDURE DELETE
>  BEGIN
>      DELETE SKILLS
>      END
```

The following example illustrates the DELETE procedure for PRIMARY, SECONDARY, and DELETE record-structures.

```
> SCREEN EMPLOYEE
> FILE EMPLOYEES PRIMARY
> FILE SHARES SECONDARY OCCURS 5
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF EMPLOYEES.
> FILE SKILLS DELETE
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF SHARES.
> GENERATE NOLIST
> BUILD LIST
SKILLS accessed via EMPLOYEE.
.
.
.
> PROCEDURE DELETE
>  BEGIN
>      DELETE EMPLOYEES
>      FOR SHARES
>          BEGIN
>              DELETE SHARES
>          END
>      DELETE SKILLS
>  END
```

Avoiding "Widowed" Detail Records

If there are more DETAIL data records on the file than can be displayed on the current screen, only those data records currently displayed are deleted. Since the PRIMARY data record is deleted, any remaining DETAIL data records are now "widowed".

To design a screen that simultaneously deletes the PRIMARY file data record and associated DETAIL file data records, redeclare the DETAIL file as a DELETE file with an alias (for example, DELSKILL). QDESIGN then generates a modified DELETE procedure:

```
> FILE SKILLS DETAIL OCCURS 10
> FILE SKILLS DELETE ALIAS DELSKILL
.
.
.
> PROCEDURE DELETE
>   BEGIN
>     DELETE EMPLOYEES
>     FOR SKILLS
>       BEGIN
```

```
>         DELETE SKILLS
>         END
>     DELETE DELSKILL
>     END
> PROCEDURE DETAIL DELETE
>   BEGIN
>     DELETE SKILLS
>     END
```

DESIGNER

Creates a designer-defined Action field command.

Syntax

PROCEDURE DESIGNER n|name [HELP string]
[NODATA] [PRECOMMANDS conditional-command-list]
[POSTCOMMANDS conditional command list]

n|name

Specifies a number (n) or a name that identifies the DESIGNER procedure.

DESIGNER procedure names should be chosen to avoid conflict with Action field commands. In the case of a conflict, the Action field command takes precedence.

Limit: Only the first four characters of the name are recognized; the number must be a positive integer, up to 9999.

HELP string

Specifies a message that is displayed when the user enters the Extended help (??) command in the Action field.

If no message is specified, QUICK displays a default message.

NODATA

Allows the user to execute the named DESIGNER procedure even if there is no data on the screen. The NODATA option is unnecessary on menu screens.

Limit: The NODATA option is ignored for numbered DESIGNER procedures.

PRECOMMANDS conditional-command-list

Lists the command options to be executed before processing the procedural code within the DESIGNER procedure.

POSTCOMMANDS conditional-command-list

Lists the command options to be executed after processing the procedural code within the DESIGNER procedure.

conditional-command-list

Specifies what command(s) the PRECOMMANDS or POSTCOMMANDS options execute and, optionally, under what conditions. 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. The general form of a command list is:

command [, command]...

For a list of the available commands, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

condition

A condition is a logical test that has the general form:

[NOT] condition [AND|OR [NOT] condition]...

For more information about conditions or conditional command lists, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Discussion

You can use DESIGNER procedures to

- make changes to selected fields or groups of fields
- initiate COMMAND and SUBSCREEN statements in either the Correction phase of Entry mode or in the Change phase of Find mode processing

For the purposes listed above, these procedures are identical to manually performing the specified actions, except for the effects of the OMIT (ON ENTRY|FIND), DISPLAY (ON ENTRY|FIND), NOCHANGE, and NOCORRECT field options.

During execution of this procedure, one of CHANGEMODE, CORRECTMODE, ENTRYMODE, or FINDMODE is true, or they can all be false. If SELECTMODE is true, FINDMODE is also true.

Note: For information about verb and procedure compatibility, see (p. 239).

Numbered DESIGNER Procedures

QUICK generates a default DESIGNER procedure for each ID-number on your QUICK screen. The source is not available for modification. When you write your own numbered DESIGNER procedures, the number is treated as an ID-number and overrides any default procedure associated with the number. When the QUICK screen user enters an ID-number in the Action field, QUICK initiates the DESIGNER procedure with the corresponding number.

The construction of default numbered DESIGNER procedures follows the same general rules as those for constructing default ENTRY procedures, with the following exceptions:

- The IF option is ignored.
- The AUTO option on COMMAND and SUBSCREEN statements isn't needed.
- The FOR control structure is used only when a cluster and all of its contents have the same ID-number.

QUICK generates one numbered DESIGNER procedure per cluster, and associates the ID number of the first field in the cluster with that DESIGNER procedure. The procedure generated for the first field in the cluster is used for every occurrence of the cluster. DESIGNER procedures that have ID numbers of other occurrences in a cluster are ignored.

Named DESIGNER Procedures

A DESIGNER procedure with a designer-defined name can be written to create a new action.

A named DESIGNER procedure is used for a specific occurrence of a cluster (or a set of occurrences) similar to the Delete action. For example, for a DESIGNER procedure TRANSFER, you can use TRAN-2 or TRAN-3/5 in the Action field.

Invoking DESIGNER Procedures

Unless the NODATA option has been specified, DESIGNER procedures (whether numbered by ID-numbers or named) can only be invoked by the user when there is active data (not marked for deletion) on the screen.

Error Handling in DESIGNER Procedures

If an error occurs during the execution of this procedure, QUICK backs up to the last ACCEPT or PROMPT verb, or to the last BLOCK TRANSFER control structure. If no such verb or control structure exists, QUICK prompts at the Action field.

Warning

Don’t name DESIGNER procedures using letters or symbols to override an already assigned function in QUICK. For example, if you write a DESIGNER procedure named "D", this procedure is automatically overridden by the Delete (D) Action field command. Note, however, that if the "D" command is changed in QKGO, then the DESIGNER procedure named "D" works.

Examples

The following layout statements for a screen for the EMPLOYEES record-structure that uses the BRANCHES record-structure as a REFERENCE record-structure

```
> FIELD EMPLOYEE OF EMPLOYEES NOCHANGE ID 5
> FIELD BRANCH OF EMPLOYEES ID SAME LOOKUP ON BRANCHES
> SUBSCREEN SKILLS PASSING EMPLOYEES ID SAME
> FIELD BRANCHNAME OF BRANCHES ID SAME
> COMMAND "QUIZ" ID SAME
```

result in the following default numbered DESIGNER procedure for the Change phase of Find mode by QUICK. The source code for the default procedure is not available.

QDESIGN automatically places the RUN SCREEN verb after the DISPLAY and ACCEPT verbs in the DESIGNER procedure even though the design statements that generate the verbs are not in this order. In contrast, the ENTRY procedure that is generated from the same design statements doesn't rearrange the verbs.

```
> PROCEDURE DESIGNER 5
>   BEGIN
>     DISPLAY EMPLOYEE OF EMPLOYEES
>     ACCEPT BRANCH OF EMPLOYEES
>     DISPLAY BRANCHNAME OF BRANCHES
>     RUN SCREEN SKILLS PASSING EMPLOYEES
>     RUN COMMAND "QUIZ"
>   END
```

The following named DESIGNER procedure can be specified for the STAFF screen to prompt the user through all the fields normally involved in a staff transfer to a new branch. The procedure is performed when a QUICK screen user enters TRAN in the Action field after retrieving a valid EMPLOYEES file data record on the screen.

The command TRAN is valid in Find mode once a record has been retrieved. It is also valid in the Correction phase of Entry mode.

```
> PROCEDURE DESIGNER TRAN
>   BEGIN
>     ACCEPT ADDRESS OF EMPLOYEES
>     ACCEPT STREET OF EMPLOYEES
>     ACCEPT CITY OF EMPLOYEES
>     ACCEPT PROVSTATE OF EMPLOYEES
>     ACCEPT POSTALCODE OF EMPLOYEES
>     ACCEPT PHONE OF EMPLOYEES
>     ACCEPT BRANCH OF EMPLOYEES
>     DISPLAY BRANCHNAME OF BRANCHES
>   END
```

Using the PRECOMMANDS and POSTCOMMANDS Options

The following code includes a standard invoice MASTER file and a DETAIL file with scrolling clusters. The invoice lines are automatically totalled and added to the invoice total. When the user finishes entering DETAIL records, he returns to the Action field and enters the Designer command "PUT", instead of a standard update command.

The PUT Designer procedure equates the items in the AUDIT file to similar items in the MASTER file. The POSTCOMMAND option on the procedure updates the invoice records. This saves time because normally the user would have to enter a PUT to save the audit record and then enter update to save the main record (or have a PUT for each file in the DESIGNER procedure).

```
> SCR INVCAP
> FILE INVMAST PRIMARY
> FILE INVDET DETAIL OCCURS 8 CACHE ; Here is the Cache
> FILE BILLAUDIT SECONDARY
> FILE EMPLOYEES REFERENCE
> FILE POSITION REFERENCE
> FILE CUSTOMERS REFERENCE
> FILE PROJECTS REFERENCE
> ALIGN ( ID 1 , LABEL 4 , DATA 21 ) &
>   ( ID 41 , LABEL 44 , DATA 61 )
> FIELD INVOICENO OF INVMAST REQUIRED NOCHANGE  &
```

```
>  LOOKUP NOTON INVMAST
> FIELD PROJECTNO OF INVMAST REQUIRED NOCHANGE  &
>  LOOKUP ON PROJECTS
> FIELD CUSTOMER OF INVMAST REQUIRED NOCHANGE  &
>  LOOKUP ON CUSTOMERS
> FIELD DATEYEAR OF INVMAST LABEL "Inv. Date"
> FIELD DATEMONTH OF INVMAST NOID NOLABEL DATA AT ,67
> FIELD DATEDAY OF INVMAST NOID NOLABEL DATA AT ,70
> FIELD DUEDATE OF INVMAST
> FIELD INVTOTAL OF INVMAST DISPLAY
> FIELD INVPAID OF INVMAST
> SKIP 2
> TITLE "   Employee      Chargeout     No. of Days    Total"
> CLUSTER OCCURS WITH INVDET
> ALIGN (1,,6)(,,18)(,,36)(,,55)
> FIELD EMPLOYNO OF INVDET REQUIRED NOCHANGE LOOKUP  &
>  ON EMPLOYEES USING EMPLOYNO OF INVDET
> FIELD  CHARGEOUT OF INVDET NOID DISPLAY LOOKUP ON  &
>  POSITION USING POSITION OF EMPLOYEES
> FIELD NBROFDAYS OF INVDET NOID
> FIELD LINETOTAL OF INVDET NOID DISPLAY
> CLUSTER
> PROCEDURE EDIT EMPLOYNO
> BEGIN
>   LET CHARGEOUT OF INVDET = CHARGEOUT OF POSITION
> END
> PROCEDURE EDIT NBROFDAYS
> BEGIN
>   LET LINETOTAL = CHARGEOUT OF INVDET * NBROFDAYS
>   DISPLAY CHARGEOUT OF INVDET
>   DISPLAY LINETOTAL OF INVDET
>   LET INVTOTAL = INVTOTAL + LINETOTAL OF INVDET
>   DISPLAY INVTOTAL OF INVMAST
> END
> ;Next is the designer with postcommands
>
> PROCEDURE DESIGNER PUT POSTCOMMANDS UPDATE NEXT
>
> ; This procedure enters an audit record into BILLAUDIT
> ; equating all relevant values in the INVMAST record
> ; and adding the system time and date.
> BEGIN
>   LET INVOICENO OF BILLAUDIT = INVOICENO OF INVMAST
>   LET DATEYEAR OF BILLAUDIT  = DATEYEAR OF INVMAST
>   LET DATEMONTH OF BILLAUDIT = DATEMONTH OF INVMAST
>   LET DATEDAY OF BILLAUDIT   = DATEDAY OF INVMAST
>   LET CUSTOMER OF BILLAUDIT  = CUSTOMER OF INVMAST
>   LET INVTOTAL OF BILLAUDIT  = INVTOTAL OF INVMAST
>   LET DUEDATE OF BILLAUDIT   = DUEDATE OF INVMAST
>   LET INVPAID OF BILLAUDIT   = INVPAID OF INVMAST
>   LET DATESTAMP OF BILLAUDIT = SYSDATE
>   LET TIMESTAMP OF BILLAUDIT = SYSTIME
>   PUT BILLAUDIT
>  END
```

DETAIL DELETE

Marks for deletion specific data records in a DETAIL file and any files that occur with it.

Syntax

PROCEDURE DETAIL DELETE

Discussion

QDESIGN generates the DETAIL DELETE procedure to mark specific data records of a DETAIL file and files that occur with it, for deletion.

Note: For information about verb and procedure compatibility, see (p. 239).

When the DETAIL DELETE Procedure is Initiated

The DETAIL DELETE procedure is initiated when the QUICK screen user specifies a range to be deleted in the Action field, as in D-2 or D-2/5, where the numbers refer to data records of the DETAIL file.

When a range of numbers is used, the DETAIL DELETE procedure is executed for all cluster occurrences whose first ID-number falls within the range. This is true for both PANEL and NOPANEL screens. The data records are not deleted until the UPDATE procedure has been executed.

During the execution of this procedure, one of the predefined conditions, CHANGEMODE or CORRECTMODE, is true.

Error Handling in the DETAIL DELETE Procedure

If an error occurs during the execution of this procedure, the rest of the procedure is skipped and QUICK prompts at the Action field.

Example

The following screen design generates these procedures: DELETE and DETAIL DELETE.

```
> SCREEN EMPSKILL
> FILE EMPLOYEES
> FILE SKILLS DETAIL OCCURS 8
Item EMPNUM initialized (fixed) to EMPNUM OF EMPLOYEES.
>
> FIELD EMPNUM OF EMPLOYEES &
>   REQUIRED NOCHANGE &
>   LOOKUP NOTON EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES  &
>   REQUIRED NOCHANGE
> FIELD FIRSTNAME OF EMPLOYEES
>
> ALIGN (1,4,14)
> CLUSTER OCCURS WITH SKILLS FOR 2,36
>   FIELD SKILLCODE OF SKILLS
> CLUSTER
>
> BUILD LIST
SKILLS accessed via EMPNUM.
```

The generated DELETE and DETAIL DELETE procedures are as follows:

```
> PROCEDURE DELETE
>   BEGIN
>     DELETE EMPLOYEES
>     FOR SKILLS
>       BEGIN
>         DELETE SKILLS
>         END
```

```
>      END
>
> PROCEDURE DETAIL DELETE
>   BEGIN
>     DELETE SKILLS
>     END
```

DETAIL FIND

Controls data record retrieval for a DETAIL file and any files that occur with it.

Syntax

PROCEDURE DETAIL FIND

Discussion

QDESIGN generates the DETAIL FIND procedure to control the retrieval of DETAIL data records. This procedure also controls the retrieval of data records from any file that is declared to occur with a DETAIL file.

Note: For information about verb and procedure compatibility, see (p. 239).

When the DETAIL FIND Procedure is Initiated

The DETAIL FIND procedure is executed after the FIND procedure (or after the POSTFIND procedure if one exists), and before retrieved data is displayed on the screen.

How the DETAIL FIND Procedure Works

After the FIND procedure retrieves a data record from the PRIMARY file, the DETAIL FIND procedure retrieves as many DETAIL data records as can be displayed on the screen. When the screen user repeatedly presses [Return], the procedure retrieves successive groups of DETAIL data records associated with the PRIMARY data record. When all the DETAIL data records associated with PRIMARY data records are displayed, pressing [Return] retrieves the next PRIMARY data record. The DETAIL FIND procedure can be stopped by the CANCEL, NEXT, or UPDATE NEXT commands.

During the execution of this procedure, the FINDMODE predefined condition is true. SELECTMODE is true if the Select action is in effect.

Error Handling in the DETAIL FIND Procedure

If an error occurs during the execution of this procedure, QUICK backs up to the last executed GET verb for the DETAIL file. If no GET verb exists, QUICK prompts the user without displaying any retrieved data.

Record Status in the DETAIL FIND Procedure

Assigning values to items within the DETAIL FIND procedure doesn't change the data record status. Use DETAIL POSTFIND for this processing. Wherever possible, make changes in the POSTFIND and DETAIL POSTFIND procedures rather than in the default FIND and DETAIL FIND procedures.

Example

The following screen counts the skills of selected company personnel. In the following example:

- FOR SKILLS causes QUICK to repeat this DETAIL FIND procedure for each occurrence of SKILLS.
- Each successful GET verb for the SKILLS file causes QUICK to increment COUNTER by one. OPTIONAL causes QUICK to continue when a value in the SKILLS file isn't associated with the employee, and is therefore not retrieved.
- When all the values of SKILLS have been evaluated, QUICK displays the information message and the result of the count.

```
> SCREEN EMPSKILL
>
> TEMPORARY COUNTER NUMERIC*3 INITIAL 0
>
> FILE EMPLOYEES PRIMARY
> FILE SKILLS DETAIL OCCURS 12
```

```
Item EMPNUM initialized (fixed) to EMPNUM of EMPLOYEES.
>
> FIELD EMPNUM OF EMPLOYEES REQUIRED NOCHANGE &
>       LOOKUP NOTON EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES
> CLUSTER OCCURS WITH SKILLS
> FIELD SKILLCODE OF SKILLS
> CLUSTER

> PROCEDURE PATH
>   BEGIN
>    REQUEST EMP-NUM OF EMPLOYEES
>    IF PROMPTOK
>      THEN LET PATH = 1
>    IF PATH = 0
>     THEN BEGIN
>       LET PATH = 2
>       END
>    END
>
> PROCEDURE FIND
>   BEGIN
>     IF PATH = 1
>       THEN BEGIN
>        GET EMPLOYEES VIA EMP-NUM
>        IF NOT ACCESSOK
>          THEN ERROR &
>   "Sorry, that is not an employee number"
>        END
>      IF PATH = 2
>        THEN GET EMPLOYEES SEQUENTIAL
>      END
>
> PROCEDURE DETAIL FIND
>   BEGIN
>     FOR SKILLS
>       BEGIN
>         GET SKILLS OPTIONAL
>         IF ACCESSOK
>           THEN LET COUNTER = COUNTER + 1
>         ELSE INFORMATION = &
>              "This employee has " +&
>              ASCII(COUNTER) + " skills(s)."
>         END
>     END
> BUILD
```

DETAIL POSTFIND

Performs processing after successful completion of a DETAIL FIND procedure.

Syntax

PROCEDURE DETAIL POSTFIND

Discussion

The DETAIL POSTFIND procedure is an optional procedure that you can write to supplement the generated DETAIL FIND procedure. It's executed immediately after the successful completion of the DETAIL FIND procedure, and prior to the display of retrieved data.

During the execution of this procedure, the CHANGEMODE predefined condition is true.

Note: For information about verb and procedure compatibility, see (p. 239).

Error Handling in the DETAIL POSTFIND Procedure

If an error occurs during the execution of the DETAIL POSTFIND procedure, QUICK

- skips the rest of the procedure
- displays the data retrieved by the FIND and DETAIL FIND procedures
- prompts the QUICK screen user at the Action field

Example

In the following screen design, two temporary items are defined. The first is LASTSEQNO, an item that contains the last sequence number found on the screen. The second is OCCCOUNT, which counts the number of occurrences of the SKILLS file found on the screen.

```
> SCREEN EMPLOY
>
> TEMPORARY OCCCOUNT NUMERIC INITIAL 0
> TEMPORARY LASTSEQNO NUMERIC INITIAL 0
>
> FILE EMPLOYEES PRIMARY
> FILE SKILLS DETAIL OCCURS 7
Item EMPNUM initialized (fixed) to EMPNUM OF EMPLOYEES.
> FIELD EMPNUM OF EMPLOYEES REQUIRED NOCHANGE &
>       LOOKUP NOTON EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> FIELD FIRSTNAME OF EMPLOYEES
> CLUSTER OCCURS WITH SKILLS FOR 2,36
>   FIELD SKILLCODE OF SKILLS
> CLUSTER
```

The following procedure sets LASTSEQNO to the last sequence number found on the screen:

```
> PROCEDURE DETAIL POSTFIND
>   BEGIN
>     FOR SKILLS
>       BEGIN
>         LET LASTSEQNO = SEQNO OF SKILLS
>         LET OCCCOUNT = OCCURRENCE
>         END
>     IF OCCCOUNT = 0
>       THEN LET LASTSEQNO = 0
>     END
>
> BUILD LIST
SKILLS accessed via EMPNUM.
```

EDIT

Performs editing on a value entered in a named field.

Syntax

PROCEDURE EDIT field

field

Names the field with which the EDIT procedure is associated.

Discussion

The EDIT procedure is written by the designer to perform editing of values in the named field. This procedure is initiated in response to an EDIT verb or immediately after all dictionary and FIELD statement editing has been performed by an ACCEPT verb. The EDIT procedure refers only to the named field and is invoked when a new value or changed value is entered in the field.

During execution of this procedure, one of CHANGEMODE, CORRECTMODE, ENTRYMODE, or FINDMODE is true, or they can all be false. If SELECTMODE is true, FINDMODE is also true.

Note: For information about verb and procedure compatibility, see (p. 239).

The EDIT Procedure and Null Entries

The EDIT procedure allows for special validation of the value entered by a QUICK screen user. This procedure is performed only if the FIELDTEXT predefined item is not null at the end of the INPUT procedure. Null entries and default entries are not edited. Entries made using the DUPLICATE field command are edited.

Changing Data Values in the EDIT Procedure

The EDIT procedure should not directly change the value of the associated item. The value can be changed by using FIELDTEXT or FIELDVALUE (depending on the data type) as the target of a LET verb. This is because the value of the expression is immediately placed in the data record or temporary item buffer and is overwritten by the contents of FIELDTEXT or FIELDVALUE in a later processing step. To avoid changes based on data that has not yet been validated, use the PROCESS procedure to move data values into other items.

Comparing Entered Values to Existing Values

Within an EDIT procedure, there are two values associated with the named field: the old, or existing, value of the data record item or temporary item, and the new value entered by the user. The new value can be referenced in an expression using the predefined items FIELDTEXT (for character data) or FIELDVALUE (for numeric or date data), or the name of the item. The old value, which is in a buffer, can be referenced in an expression by using the OLDVALUE function.

Error Handling in the EDIT Procedure

If the EDIT procedure is called by the ACCEPT verb, the rest of the procedure is skipped and QUICK prompts at the Action field.

If the procedure is called by the EDIT verb, the rest of the procedure is skipped and QUICK continues error processing for the procedure containing the EDIT verb.

Example

The EDIT procedure is useful when editing requirements are too complex to be specified on a FIELD statement.

For example, if you want to validate combinations of entries for new records that have multi-segment indexes before QUICK actually retrieves data, specify an EDIT on the last segment as in:

```
> PROCEDURE EDIT PARTVARIANT
>  BEGIN
>  IF FIELDTEXT = "BLUE" AND PARTNUMBER = 20
>   THEN ERROR "Invalid Part Number/Variant combination"
>  END
```

Similarly, the following edit is too complex to be specified on a FIELD statement:

```
> PROCEDURE EDIT CUSTOMERCODE
>   BEGIN
> ;     If type Y customer
>     IF "Y" = FIELDTEXT[1:1]
>       THEN GET YFILE USING CUSTOMERCODE OPTIONAL
>       ELSE GET ZFILE USING CUSTOMERCODE OPTIONAL
>     IF NOT ACCESSOK
>       THEN ERROR "Customer not on file"
>     END
```

The CUSTOMERCODE entered is looked up on one of two different REFERENCE files depending on the first letter of the code. If the CUSTOMERCODE is found in a data record of the appropriate file, no further validation is performed.

ENTRY

Performs the standard entry sequence.

Syntax

PROCEDURE ENTRY

Discussion

The ENTRY procedure, together with the APPEND procedure, establishes the entry sequence for the screen when a user enters new data records.

Notes: An ENTRY procedure isn't generated for screen designs that include a designer-written APPEND or ENTRY procedure.

For information about verb and procedure compatibility, see (p. 239).

The Default ENTRY Procedure

The default ENTRY and APPEND procedures are based on the FIELD and other layout section statements in the screen design. Usually, QDESIGN automatically generates an ENTRY procedure. A screen can't perform standard data entry unless an ENTRY procedure exists.

The default ENTRY procedure is different for PANEL and NOPANEL screens. NOPANEL screens don't contain any BLOCK TRANSFER control structures. PANEL screens contain BLOCK TRANSFER control structures based on FIELD and CLUSTER statements. In addition, PANEL screens ignore the IF option on FIELD statements.

The default ENTRY procedure consists of the following:

- an ACCEPT verb for each FIELD statement that doesn't have a DISPLAY, OMIT ON ENTRY, NOENTRY, or SILENT option. The ACCEPT verb controls all aspects of prompting, validating, and storing a value entered by the QUICK screen user
- BLOCK TRANSFER statements based on FIELD and CLUSTER statements on PANEL screens
- a DISPLAY verb for each FIELD statement that has a DISPLAY ON ENTRY option or is associated with a defined item
- an EDIT verb for each FIELD statement that has a SILENT option
- a RUN SCREEN verb for every SUBSCREEN statement that has an AUTO or IF option
- a RUN COMMAND verb for every COMMAND statement that has an AUTO or IF option
- a PERFORM APPEND verb if the screen has a repeating PRIMARY or DETAIL file
- an IF control structure corresponding to the IF option on each FIELD, SUBSCREEN or COMMAND statement. For ENTRY procedures that contain BLOCK TRANSFER control structures, the IF option is ignored for FIELD statements
- a FOR control structure, if a cluster occurs a different number of times than the previous cluster, if the screen design includes a DETAIL file, or if a cluster occurs a different number of times than the PRIMARY file

Clusters in the Default ENTRY Procedure

Any CLUSTER statement in the layout section that references multiple occurrences of either a primary or detail record-structure produces a FOR (loop) control structure in the default ENTRY procedure.

The ENTRY Procedure and the APPEND Procedure

For PRIMARY and DETAIL files with more than one occurrence on a screen, the ENTRY procedure is modified and paired with an APPEND procedure. For more information about Append processing, see (p. 295).

Using FIELD Statements to Control the ENTRY Procedure

The FIELD statement and its options control the construction of the default ENTRY procedure:

- Normally, a FIELD statement produces an ACCEPT verb.
- The DISPLAY option of the FIELD statement and the FIXED option of the ITEM statement produce a DISPLAY verb.
- The OMIT (or OMIT ON ENTRY), FIXED, and NOENTRY FIELD options exclude the field from the ENTRY procedure.
- The SILENT FIELD option produces an EDIT verb. The field doesn't appear on the screen.
- The IF option of a FIELD statement produces an equivalent IF control structure in the ENTRY procedure, unless the ACCEPT verb for the field is within a BLOCK TRANSFER control structure.

Implications of Modifying the Default ENTRY Procedure

It is recommended that you don't explicitly write the ENTRY procedure; it is more effective to use design statements in the layout section that influence the way in which QDESIGN constructs the ENTRY procedure. The influence of these design statements depends upon their order and content in the screen layout section.

When modifying or replacing the ENTRY procedure, you should confine the activities specified to a dialogue with the user (in the form of REQUEST verbs) to create new data prior to placing it in a file. Remember that at any point prior to updating, the user can decide to back up to a previous field prompt, or back out of the process entirely.

Many of the procedural activities associated with data entry, such as editing, are field related and also apply to changing data (both in Change mode and in the Correction phase of Entry mode). These activities are best left to field processing procedures. As a general rule, processing that is related to a single field and can be done in the field processing procedures should not be done in the ENTRY procedure. For more information, see (p. 312), (p. 324), (p. 331), and (p. 353).

During the execution of the ENTRY procedure, the ENTRYMODE predefined condition is true.

Error Handling in the ENTRY Procedure

If an error occurs during the execution of the ENTRY procedure, QUICK backs up to the last ACCEPT or PROMPT verb, or to the last BLOCK TRANSFER control structure. If no such verb or control structure exists, QUICK prompts the screen user at the Action field.

Examples

The following example shows a simple screen design and the default ENTRY procedure that is produced:

```
> SCREEN BRANCH
> FILE BRANCHES PRIMARY
> FIELD BRANCH REQUIRED NOCHANGE &
>   LOOKUP NOTON BRANCHES
> FIELD BRANCHNAME
> FIELD BRANCHMANAGER
> BUILD LIST
.
.
.
> PROCEDURE ENTRY
>   BEGIN
>     ACCEPT BRANCH OF BRANCHES
>     ACCEPT BRANCHNAME OF BRANCHES
>     ACCEPT BRANCHMANAGER OF BRANCHES
>     END
.
.
.
```

The ENTRY procedure in the previous example accepts a value in each of the three fields of the screen, in the order that the fields are specified. In each case, the value is stored in the record buffer for the BRANCHES file.

The following portion of the STAFF screen

```
> FIELD DIVISION OF EMPLOYEES LOOKUP ON DIVISIONS &
>   IF BRANCH = " " AND LANGUAGE = "E"
> FIELD DIVISIONNAME OF DIVISIONS DISPLAY ID SAME
```

produces these statements in the default ENTRY procedure:

```
>   IF BRANCH OF EMPLOYEES EQ " "  &
>     AND LANGUAGE OF EMPLOYEES EQ "E"
>     THEN ACCEPT DIVISION OF EMPLOYEES
>     ELSE DISPLAY DIVISION OF EMPLOYEES
>     DISPLAY DIVISIONNAME OF DIVISIONS
```

The following QDESIGN statements list the resulting ENTRY procedure for a screen for the BRANCHES file with a cluster:

```
> SCREEN BRANCH
> FILE BRANCHES OCCURS 10
> TITLE "Branches File" AT 3,8
> TITLE "Branch" AT 5,4
> TITLE "Branchname" AT 5,12
> TITLE "Branchmanager" AT 5,34
.
.
.
> ALIGN (1,,4) (,,12) (,,34)
> CLUSTER OCCURS WITH BRANCHES
>   FIELD BRANCH  OF BRANCHES &
>     REQUIRED NOCHANGE &
>     LOOKUP NOTON BRANCHES
>   FIELD BRANCHNAME OF BRANCHES
>   FIELD BRANCHMANAGER OF BRANCHES
> CLUSTER
> BUILD LIST
>
> PROCEDURE APPEND
>   BEGIN
>     ACCEPT BRANCH OF BRANCHES
>     ACCEPT BRANCHNAME OF BRANCHES
>     ACCEPT BRANCHMANAGER OF BRANCHES
>   END
>
> PROCEDURE ENTRY
>   BEGIN
>     FOR BRANCHES
>       BEGIN
>         PERFORM APPEND
>         END
>   END
.
.
.
```

These layout statements

```
> SUBSCREEN ABC PASSING RUNOK AUTO
> COMMAND "QUIZ" IF RUNOK > 0
```

produce these statements in the ENTRY procedure:

```
> RUN SCREEN ABC PASSING RUNOK MODE E
> IF RUNOK > 0
>   THEN RUN COMMAND "QUIZ"
```

EXIT

Performs processing just before returning to a higher-level screen.

Syntax

PROCEDURE EXIT

Discussion

The EXIT procedure is initiated by a RETURN verb, or by any one of

- the Update Return (UR) Action field command
- the Return (^) Action field command
- the Return to Stop (^^) Action field command
- the Return to Start (^^^) Action field command

It is also initiated by the Update (U) Action field command on screens with the AUTORETURN option.

The EXIT procedure executes immediately before the backout buffers are updated. In a backout situation, any BACKOUT procedure is performed before the EXIT procedure.

During the execution of the EXIT procedure, the predefined conditions, CHANGEMODE, CORRECTMODE, ENTRYMODE, SELECTMODE and FINDMODE, are all false.

Notes: Changing the data record status of a file that is local to the screen does not create a backout situation; the change is lost.

For information about verb and procedure compatibility, see (p. 239).

Error Handling in the EXIT Procedure

If an error occurs during the execution of the EXIT procedure, QUICK backs up to the last ACCEPT or PROMPT verb, or to a previous BLOCK TRANSFER control structure. If no such verb or control structure exists, control returns to the invoking screen.

Example

This screen was designed with an EXIT procedure. When a user exits the screen, QUIZ is initiated to generate an up-to-date EMPLOYEE/SKILL report:

```
> SCREEN EMPLOYEE
>
> FILE EMPLOYEES
> FILE SKILLS DETAIL OCCURS 8
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF EMPLOYEES.
> FILE SKILLS DELETE ALIAS DELSKILL
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF EMPLOYEES
>
> FIELD EMPLOYEE OF EMPLOYEES REQUIRED NOCHANGE &
>   LOOKUP NOTON EMPLOYEES
> FIELD FIRSTNAME OF EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> CLUSTER OCCURS WITH SKILLS FOR 2,36
>   FIELD SKILL OF SKILLS
> CLUSTER
> PROCEDURE EXIT
>   BEGIN
>       ;generate an up-to-date
>       ;employee / skill report
>      RUN COMMAND 'QUIZ AUTO=EMPREP'
>   END
>
> BUILD
```

FIND

Retrieves data records as indicated by the PATH procedure.

Syntax

PROCEDURE FIND

Discussion

QDESIGN automatically generates a FIND procedure. A screen can't function in Find mode unless a FIND procedure exists. Menu and slave screens are excepted.

Assigning values to record items within the scope of the FIND procedure doesn't change the data record status. Use POSTFIND for this processing.

QUICK ignores the FIND procedure on slave screens. There is no FIND procedure on menu screens.

Note: For information about verb and procedure compatibility, see (p. 239).

When the FIND Procedure is Initiated

The FIND procedure is initiated

- after completion of the PATH and POSTPATH procedures
- when the QUICK screen user enters the Next (N) Action field command in Find mode
- when the QUICK screen user presses [Return] and there are no more detail records related to the primary record to be displayed
- when the screen is started in Find or Select mode

How the FIND Procedure Works

The FIND procedure is governed by the value of the predefined item PATH (which is established in the PATH procedure), and by any selection conditions that might restrict the data records actually retrieved.

The standard FIND procedure first performs one of several GET verbs for the PRIMARY file, based on the value of the predefined item PATH (set by the PATH procedure). The FIND procedure then performs a GET verb with the OPTIONAL keyword for each SECONDARY file.

When finding data, QUICK displays a PRIMARY data record and as many DETAIL data records as the screen can display. If the QUICK screen user presses [Return], QUICK displays any remaining DETAIL data records. The PRIMARY data record only changes when there are no more DETAIL data records.

Entering the Next or Update Next command causes QUICK to display the next PRIMARY data record and its associated DETAIL data records.

The FIND procedure should retrieve data records of the PRIMARY and SECONDARY files of the screen. If no primary records are retrieved, there is no data to be manipulated. Retrieval of secondary records is assumed to be optional.

Changing the Default FIND Procedure

Whenever possible, make changes to PATH and FIND processing with ACCESS statements. Changes in the generated PATH and FIND procedures are then forced by the ACCESS statement, rather than by direct manipulation of QDESIGN's default procedures. If you make changes to the FIND procedure, you must make corresponding changes in the PATH procedure. Any changes that can't be handled by ACCESS statements are best handled in the POSTPATH and POSTFIND procedures.

During the execution of this procedure, the FINDMODE predefined condition is true. SELECTMODE is true if the Select action is in effect.

SQL in the FIND Procedure

For a PRIMARY CURSOR, PowerHouse generates a number of SQL OPEN statements and a single SQL FETCH statement. For SECONDARY and DETAIL CURSORs, one OPEN and a single SQL FETCH statement are generated.

PowerHouse generates multiple SQL OPEN statements that contain substitutions for the WHERE and ORDERBY substitution-variables. For a VIA list with n items that can be generic, n+1 SQL OPEN statements are generated. Each SQL OPEN has a different WHERE substitution that reflects the search criteria that the user could enter.

At runtime, QUICK uses the correct SQL OPEN by analyzing the entered search criteria. The PATH procedure determines the value of the PATH and SUBPATH variables. The FIND procedure examines the PATH and SUBPATH variables and uses the appropriate SQL OPEN statement.

Error Handling in the FIND Procedure

If an error occurs during the execution of this procedure, QUICK backs up to the last GET verb for the PRIMARY file. If no such verb exists, QUICK prompts the screen user in the Action field, without displaying any retrieved data.

Example

The following example illustrates how the ACCESS statement affects the default FIND procedure.

The PATH procedure is based on the ACCESS statements specified for the PRIMARY record-structure. QDESIGN builds two paths: one based on the index EMPLOYEE (using segment EMPLOYEE) and a second one based on the index LASTNAME (using segment LASTNAME). If no values are entered for either of these fields in the QUICK screen, QUICK issues an error message, since no alternative access paths are specified (such as ACCESS SEQUENTIAL).

The FIND procedure uses the value for PATH to retrieve data records via one of the retrieval alternatives.

The LIST option of the BUILD statement causes QDESIGN to display the generated procedures.

```
> SCREEN MODEMP ACTIVITIES FIND, CHANGE, DELETE
>
> FILE EMPLOY1 PRIMARY
>  ACCESS VIAINDEX EMPLOYEE &
>      USING EMPLOYEE OF EMPLOY1 &
>      REQUEST EMPLOYEE
>  ACCESS VIAINDEX LASTNAME &
>      USING LASTNAME OF EMPLOY1 &
>      REQUEST LASTNAME
>
> TITLE "Modify Employee Attributes" &
>      CENTERED AT 4,1
>
> SKIP 2
> ALIGN (1,4,21)
> FIELD EMPLOYEE OF EMPLOY1 &
>  REQUIRED &
>  NOCHANGE &
>  LOOKUP ON EMPLOY1
> SKIP 1
> FIELD LASTNAME OF EMPLOY1 &
>  REQUIRED &
>  NOCHANGE
> FIELD CITY OF EMPLOY1
> FIELD STREET OF EMPLOY1
> FIELD PROVSTATE OF EMPLOY1 &
>   ID SAME &
>   LABEL "Prov/State"
.
.
.
> FIELD POSTALZIP OF EMPLOY1 &
```

```
>  ID SAME &
>  LABEL "PCode or Zip"
>
> FIELD PHONENUMBER OF EMPLOY1 &
>  ID SAME &
>  LABEL "Phone"
>
>
> BUILD LIST DETAIL
.
.
.
> PROCEDURE PATH
>   BEGIN
>     REQUEST EMPLOYEE OF EMPLOY1
>     IF PROMPTOK
>       THEN LET PATH = 1
>     IF PATH = 0
>       THEN BEGIN
>         REQUEST LASTNAME OF EMPLOY1
>         IF PROMPTOK
>           THEN LET PATH = 2
>         END
>     IF PATH = 0
>       THEN ERROR "Key/Index required."
>   END
> PROCEDURE FIND
>   BEGIN
>     IF PATH = 1
>       THEN GET EMPLOY1 VIAINDEX EMPLOYEE &
>            USING EMPLOYEE OF EMPLOY1
>     IF PATH = 2
>       THEN GET EMPLOY1 VIAINDEX LASTNAME &
>            USING LASTNAME OF EMPLOY1
>   END
.
.
.
```

The following example shows how the generated PATH and FIND procedures are affected by the use of SQL:

```
>  SQL DECLARE EMPLIST CURSOR FOR &
>  SELECT EMPLOYEE, FIRST_NAME, LAST_NAME, &
>      BRANCHES.BRANCH, BRANCH_NAME &
>      FROM EMPLOYEES, BRANCHES &
>      WHERE EMPLOYEES.BRANCH = BRANCHES.BRANCH
>  SCREEN EMPBRANCHC
>  CURSOR EMPLIST PRIMARY KEY EMPLOYEE
>  ACCESS VIA EMPLOYEE REQUEST EMPLOYEE
>  ACCESS VIA LAST_NAME REQUEST LAST_NAME
>  ACCESS SEQUENTIAL
.
.
.
> PROCEDURE PATH
>   BEGIN
>     REQUEST EMPLOYEE OF EMPLIST
>     IF PROMPTOK
>       THEN LET PATH = 1
>     IF PATH = 0
>       THEN BEGIN
>         REQUEST LAST_NAME OF EMPLIST
>         IF PROMPTOK
>           THEN LET PATH = 2
>         END
>     IF PATH = 0
>       THEN BEGIN
>         LET PATH = 3
```

```
>         END
>     IF PATH = 1
>       THEN BEGIN
>         LET SUBPATH = 0
>         END
>     IF PATH = 2
>       THEN BEGIN
>         IF RANGED( EMPLOYEES.LAST_NAME )
>           THEN LET SUBPATH = 1
>           ELSE LET SUBPATH = 0
>         END
>     END
> PROCEDURE FIND
>   BEGIN
>     IF NOT CURSOROPEN( EMPLIST )
>       THEN BEGIN
>       IF PATH = 1
>        THEN BEGIN
>         IF SUBPATH = 0
>          THEN SQL OPEN EMPLIST &
>           where( EMPLOYEES.EMPLOYEE &
>             =:linkvalue( EMPLOYEES.EMPLOYEE ,equal) )
>             END
>         IF PATH = 2
>          THEN BEGIN
>           IF SUBPATH = 1
>            THEN SQL OPEN EMPLIST &
>             where( EMPLOYEES.LAST_NAME &
>             between :linkvalue( EMPLOYEES.LAST_NAME &
>             ,lowest) and &
>             :linkvalue( EMPLOYEES.LAST_NAME &
>             ,highest) )
>           IF SUBPATH = 0
>            THEN SQL OPEN EMPLIST &
>             where( EMPLOYEES.LAST_NAME &
>             =:linkvalue( EMPLOYEES.LAST_NAME ,equal) )
>             END
>         IF PATH = 3
>           THEN BEGIN
>             SQL OPEN EMPLIST
>             END
>         END
>     SQL FETCH EMPLIST
>     END
```

INITIALIZE

Performs processing when the screen is initiated.

Syntax

PROCEDURE INITIALIZE [RECOVERABLE]

RECOVERABLE

Instructs QUICK to keep rollback information for PUTs to non-relational records.

Limit: Rollback information can only be kept if the INITIALIZE procedure is in a subscreen that is called from a recoverable procedure.

This option has no effect on PUTs to relational records since rollback information is always retained for relational records.

Discussion

You can write the optional INITIALIZE procedure to perform processing when the screen is first entered. The INITIALIZE procedure is executed every time the screen is initiated from a higher-level screen, after default item initialization is done, but before the screen background is displayed.

During the execution of the INITIALIZE procedure, the predefined conditions, CHANGEMODE, CORRECTMODE, ENTRYMODE, SELECTMODE, and FINDMODE, are all false.

Note: For information about verb and procedure compatibility, see (p. 239).

Error Handling in the INITIALIZE Procedure

If an error occurs during the execution of the INITIALIZE procedure, QUICK backs up to the last ACCEPT or PROMPT verb, or to the last BLOCK TRANSFER control structure. If no such verb or control structure exists, the EXIT procedure (if one exists) is executed and control returns to the invoking screen.

Rollbacks

A recoverable procedure is a procedure in a QUICK screen in which PUTs to non-relational records can be rolled back automatically by PowerHouse in the event of an error. By default, the UPDATE procedure is recoverable. The PREUPDATE and POSTUPDATE procedures are made recoverable by using the RECOVERABLE option.

INITIALIZE procedures can only be recoverable if they are in a subscreen that is called from a recoverable procedure. If an error occurs in the calling screen after the execution of the subscreen, the PUTs done in the INITIALIZE procedure of the subscreen can be rolled back provided the RECOVERABLE option of the INITIALIZE procedure is used.

If the RECOVERABLE option is used, PUT verbs to non-relational records are automatically rolled back by PowerHouse on encountering an error.

If the RECOVERABLE option is not used, non-relational rollback behavior is unchanged from previous versions.

Conversion

Prior to 7.33.C (**UNIX**), 7.10E1 (**OpenVMS**), and 8.09 (**MPE/iX**), no automatic rollback was available to cancel the effect of PUT verbs to non-relational records executed outside of the UPDATE procedure. If PUT verbs to non-relational records were used outside of the UPDATE procedure, you would have to write a BACKOUT procedure to reverse their effect.

If you use the RECOVERABLE option, any BACKOUT procedures that you have coded to handle rollbacks must be reviewed to ensure that a "double rollback" is not performed.

Example

This example shows a library system menu screen and one of the menu subscreens.

The KEY statement sets up [F1] as a Help function key for this screen.

```
> SCREEN LIBRARY MENU
> DRAW 3,20 TO 7,59
> TITLE "Library System Menu" AT 5,24
> SKIP 3
> ALIGN (26,32)
>
> KEY 1 LABEL "Help" ACTION AND DATA HELP
>
> SUBSCREEN BOOKINFO LABEL "Book Information"
> SUBSCREEN SUBINFO LABEL "Subject Information"
> SUBSCREEN PUBINFO LABEL "Publisher Information"
>
> BUILD
>
> SCREEN BOOKINFO
> FILE BOOKS
> FILE SUBJECTS REFERENCE
> FILE PUBLISHERS REFERENCE
>
> FIELD BOOKID OF BOOKS REQUIRED NOCHANGE &
>     LOOKUP NOTON BOOKS &
>     HELP "Enter a valid book number."
> FIELD AUTHORLASTNAME OF BOOKS &
>       REQUIRED NOCHANGE
> FIELD AUTHORFIRSTNAME OF BOOKS
> FIELD TITLE OF BOOKS
> FIELD SUBJECTCODE OF BOOKS LOOKUP ON SUBJECTS &
>     HELP "Enter the subject code."
> FIELD PUBLISHERCODE OF BOOKS LOOKUP ON PUBLISHERS &
>     HELP "Enter the publisher's name."
>
> PROCEDURE INITIALIZE
> BEGIN
>   INFORMATION =  &
>       "Press [F1] for help in most fields."
> END
>
> BUILD
```

The INITIALIZE procedure displays a message each time the BOOKINFO screen is initiated from the LIBRARY menu. The screen user is informed that pressing [F1] displays help information.

INPUT

Performs data conversion for a value entered in the named field prior to any editing.

Syntax

PROCEDURE INPUT field

field

Names the field with which the INPUT procedure is associated.

Discussion

You can write the optional INPUT procedure to modify data before any editing is done on a field. The INPUT procedure applies to a single field only and is initiated for new and changed values by an ACCEPT, PROMPT, REQUEST, or SELECT verb after size checking, but before type checking.

The INPUT procedure is performed when a user, in response to an ACCEPT, PROMPT, REQUEST, or SELECT verb

- enters data
- presses [Return]
- enters Skip All command(//)
- bypasses a field in a changed record as a result of entering the Skip All command (//).

The INPUT procedure should be used only to manipulate the screen user's entry in the FIELDTEXT predefined item.

An entry can be altered by changing the value of the predefined item FIELDTEXT with the LET verb. The INPUT procedure is always executed even when dealing with a null response. It can force the execution of the EDIT procedure (normally skipped on null responses) by placing any value other than a null response in FIELDTEXT.

During execution of this procedure, one of CHANGEMODE, CORRECTMODE, ENTRYMODE, or FINDMODE is true, or they can all be false. If SELECTMODE is true, FINDMODE is also true.

Notes: The item associated with the named field should not be used as the target of a LET verb. Only FIELDTEXT should be used.

For information about verb and procedure compatibility, see (p. 239).

Error Handling in the INPUT Procedure

If an error occurs during the execution of this procedure, the rest of the procedure is skipped and QUICK prompts at the current field.

Example

The INPUT procedure can save your QUICK screen users time and effort. For example, a screen for adding new suppliers is such that most entries in the CITY field are one of New York, Chicago, or Toronto. To save keystrokes, an entry of N, C, or T in the CITY field is set to correspond to those cities.

The PROCESS procedure for CITY sets the value of PROVSTATE to the correct province or state for New York, Chicago, or Toronto.

```
> SCREEN ADDSUPP PANEL  &
>   NOMODE  &
>   ACTION LABEL "==>" AT 20,1 &
>   ACTIVITIES ENTRY
>
> FILE SUPPLIERS PRIMARY
> FILE NEXTCODE DESIGNER
>   ACCESS USING 2
>
```

```
> TEMPORARY TEMPSTATE CHARACTER *2
> USE HEADER NOLIST NODETAIL
> TITLE "Add New Supplier" CENTERED AT 4,1
>
> DRAW 7,15 TO 19,65
>
> SKIP TO LINE 9
.
.
.
> PROCEDURE INPUT CITY
>   BEGIN
>     LET TEMPSTATE = " "
>     IF FIELDTEXT = "N"
>     THEN BEGIN
>       LET FIELDTEXT = "New York City"
>       LET TEMPSTATE = "NY"
>       END
>     ELSE IF FIELDTEXT = "C"
>     THEN BEGIN
>       LET FIELDTEXT = "Chicago"
>       LET TEMPSTATE = "IL"
>       END
>     ELSE IF FIELDTEXT = "T"
>     THEN BEGIN
>       LET FIELDTEXT = "Toronto"
>       LET TEMPSTATE = "ON"
>       END
> END
>
> PROCEDURE PROCESS CITY
>   BEGIN
>   IF TEMPSTATE <> " "
>   THEN BEGIN
>     LET PROVSTATE = TEMPSTATE
>     END
> END
>
> BUILD
```

INTERNAL

Creates an internal subroutine in the QDESIGN procedural language.

Syntax

PROCEDURE INTERNAL name

name

Names the procedure.

Limit: Must begin with a letter and can be up to 64 characters in length.

Discussion

INTERNAL procedures are subroutines written in the QDESIGN procedural language. They are invoked by other procedures using the DO INTERNAL verb and must be declared before they are referenced by a DO INTERNAL verb. The activities performed within an INTERNAL procedure must be consistent with the activities allowed for any procedure that invokes it.

There are no default INTERNAL procedures.

During execution of this procedure, the predefined condition remains the same as the calling procedure. The mode (CHANGEMODE, CORRECTMODE, ENTRYMODE, FINDMODE, or SELECTMODE) does not change.

Notes: Because an INTERNAL procedure can invoke itself, be careful not to create an infinite loop.

The INTERNAL procedure must be declared before the DO INTERNAL verb that invokes it, since QDESIGN has no forward referencing capability at compile time.

Verbs used should comply with the restrictions of the calling procedure. For more information about verb and procedure compatibility, see (p. 239).

Nesting INTERNAL Procedures

You can nest INTERNAL procedures to any level, as long as you specify them before they are referenced. In addition, you can create recursive procedures.

Error Handling in INTERNAL Procedures

If an error occurs during the execution of this procedure, QUICK follows the error processing for the procedure containing the DO INTERNAL verb.

Examples

Although QUICK doesn't allow array subscripting, you can manipulate data in arrays. You can use the INTERNAL procedure and the system function, OCCURRENCE, to address a specific occurrence of an item in an array by using verbs, as in:

```
> PROCEDURE INTERNAL SETARRAY
>   BEGIN
>     FOR EACH ARRAY1
>         LET ARRAY1 = OCCURRENCE
.
.
.
>   END
```

Standardizing Field Processing with INTERNAL Procedures

Assume that a company has a CHECKDIGIT validation routine that's applied to several items in a given application. Rather than specifying the contents of this routine within an EDIT procedure for each entry field, you can construct the following INTERNAL procedure:

```
> PROCEDURE INTERNAL STANDARDCHECK
```

```
>   BEGIN
>     IF FIELDVALUE  <> 0
>       THEN ERROR "Checkdigit edit failed"
>     END
> PROCEDURE EDIT AA
>   DO INTERNAL STANDARDCHECK
> PROCEDURE EDIT AX
>   DO INTERNAL STANDARDCHECK
> PROCEDURE EDIT BQ
>   DO INTERNAL STANDARDCHECK
.
.
```

MODIFY

Controls QUICK processing in CHANGEMODE or CORRECTMODE.

Syntax

PROCEDURE MODIFY

Discussion

The MODIFY procedure is activated when the QUICK screen user enters a Modify command. If the AUTOMODIFY option is specified on the SCREEN statement, the procedure is activated automatically after finding or entering data. The procedure is available whether the screen is using Panel mode or field mode.

During execution of this procedure, one of CHANGEMODE or CORRECTMODE is true.

Note: For more information about verb and procedure compatibility, see (p. 239).

The Default MODIFY Procedure

The default MODIFY procedure is similar to the default ENTRY procedure. The default MODIFY procedure contains an ACCEPT verb for each field that can be modified once a value has been entered into the field.

No default MODIFY procedure is generated if the NOPANEL option of either the SET statement or the SCREEN statement is in effect.

ACCEPT verbs are not generated for fields with the following options:

- SILENT
- DISPLAY
- fields with associated ITEM statements with the FIXED option
- both NOCHANGE and NOCORRECT

ACCEPT verbs in the MODIFY procedure appear in the order in which the fields are specified in the screen design. For PANEL screens, ACCEPT verbs for a given screen are grouped into a block with the BLOCK TRANSFER control structure.

The Default MODIFY Procedure and Append Processing

For screens with repeating PRIMARY or DETAIL record-structures, the default MODIFY procedure contains both a FOR and a FOR MISSING control structure. These control structures control the modification of values in repeating record-structures. In addition, the FOR MISSING control structure allows Append processing during the execution of the MODIFY procedure.

The FOR control structure in the default MODIFY procedure contains an ACCEPT verb for each field in the repeating record-structure. This causes QUICK to prompt for modifications in existing data records in the repeating record-structure.

The FOR MISSING control structure in the default MODIFY procedure contains the PERFORM APPEND verb. This causes QUICK to prompt for entries in occurrences for which no values currently exist.

Error Handling in the MODIFY Procedure

When QUICK detects an error in a field entry during the execution of the MODIFY procedure, it follows the same steps as those for the ENTRY procedure. QUICK backs up to the last ACCEPT or PROMPT verb, or the previous INTERNAL procedure, or to the last BLOCK TRANSFER control structure. If no such verb or control structure exists, QUICK prompts the user at the Action field.

For information about error handling in BLOCK TRANSFER control structures, see (p. 372).

Examples

The following screen design statements illustrate how QDESIGN generates a default MODIFY procedure. In this example:

- The CLUSTER statement affects how QDESIGN blocks groups of fields for entry in the ENTRY, MODIFY, and SELECT procedures.
- The fields referenced in the default MODIFY procedure are grouped in BLOCK TRANSFER control structures. Field blocking is based on the CLUSTER statements specified in the layout section.

```
> SCREEN MODCUST PANEL NOMODE
>
> FILE CUSTOMERS PRIMARY
>   ACCESS VIAINDEX CUSTOMERS &
>    USING CUSTOMERKEY OF CUSTOMERS &
>       REQUEST CUSTOMERKEY
>   ACCESS VIAINDEX CUSTOMERNAME &
>    USING CUSTOMERNAME OF CUSTOMERS &
>       REQUEST CUSTOMERNAME
>
> TITLE "Modify Customer Attributes" CENTERED AT 4,1
> SKIP TO LINE 8
> ALIGN (,25,40)
>
> CLUSTER BLOCK EACH
> FIELD CUSTOMERKEY OF CUSTOMERS &
>   REQUIRED &
>   NOCHANGE &
>   LOOKUP ON CUSTOMERS
> CLUSTER
>
> SKIP 1
>
> CLUSTER BLOCK ALL
> FIELD CUSTOMERNAME OF CUSTOMERS &
>   REQUIRED &
>   NOCHANGE
> FIELD CITY OF CUSTOMERS
> FIELD STREET OF CUSTOMERS
> FIELD PROVSTATE OF CUSTOMERS &
>   ID SAME &
>   LABEL "Prov/State"
> FIELD POSTALZIP OF CUSTOMERS &
>   ID SAME &
>   LABEL "PCode or Zip"
>
> FIELD PHONENUMBER OF CUSTOMERS &
>   ID SAME &
>   LABEL "Phone"
> CLUSTER
> BUILD LIST
.
.
.
> PROCEDURE MODIFY
>   BEGIN
>     BLOCK TRANSFER
>       BEGIN
>         ACCEPT CUSTOMERKEY OF CUSTOMERS
>       END
>     BLOCK TRANSFER
>       BEGIN
>         ACCEPT CUSTOMERNAME OF CUSTOMERS
>         ACCEPT CITY OF CUSTOMERS
>         ACCEPT STREET OF CUSTOMERS
>         ACCEPT PROVSTATE OF CUSTOMERS
>         ACCEPT POSTALZIP OF CUSTOMERS
```

```
>          ACCEPT PHONENUMBER OF CUSTOMERS
>        END
>   END
```

The Default MODIFY Procedure with a Repeating PRIMARY Record-structure

The following example illustrates how QDESIGN constructs the default MODIFY procedure when the screen design contains a repeating PRIMARY record-structure. In this example:

- The default MODIFY procedure is generated when the PANEL option of either the SET statement or the SCREEN statement is in effect.
- Because a BLOCK option is not specified on the CLUSTER statement, the generated MODIFY procedure defaults to BLOCK ALL.

```
> SCREEN CUSTLIST PANEL
>
> FILE CUSTOMERS PRIMARY OCCURS 4
> CLUSTER OCCURS WITH CUSTOMERS
> SKIP 1
> FIELD CUSTOMERKEY OF CUSTOMERS
> FIELD CUSTOMERNAME OF CUSTOMERS
> SKIP 1
> FIELD PHONENUMBER OF CUSTOMERS
> CLUSTER
>
> BUILD LIST
.
.
.
> PROCEDURE MODIFY
> BEGIN
> BLOCK TRANSFER
>   BEGIN
>     FOR FILE CUSTOMERS
>       BEGIN
>         ACCEPT CUSTOMERKEY OF CUSTOMERS
>         ACCEPT CUSTOMERNAME OF CUSTOMERS
>         ACCEPT PHONENUMBER OF CUSTOMERS
>       END
>       FOR MISSING CUSTOMERS
>         PERFORM APPEND
>   END
> END
```

OUTPUT

Performs data conversion for a value in the named field prior to the display of the value.

Syntax

PROCEDURE OUTPUT field

field

Names the field with which the OUTPUT procedure is associated.

For more information about verbs and control structures, see (p. 361).

Discussion

You can write the optional OUTPUT procedure to modify data between storage and output. It applies only to a single field and is initiated immediately, before formatting options are applied, by an ACCEPT, DISPLAY, PROMPT, REQUEST, or SELECT verb.

The OUTPUT procedure applies both to newly entered values which are immediately redisplayed and to existing values, which are displayed upon retrieval.

The OUTPUT procedure allows special formatting of the contents of the FIELDTEXT predefined item just prior to the automatic formatting by PICTURE or DATE formats. This procedure is used with all field-related verbs except the EDIT verb, or when QUICK performs automatic displays.

During execution of this procedure, one of CHANGEMODE, CORRECTMODE, ENTRYMODE, or FINDMODE is true, or they can all be false. If SELECTMODE is true, FINDMODE is also true.

Notes: Using the item associated with the named field as the target of a LET verb doesn't affect the displayed values and isn't recommended.

For more information about verb and procedure compatibility, see (p. 239).

Manipulating the FIELDTEXT Predefined Item in the OUTPUT Procedure

The OUTPUT procedure is based on the contents of the FIELDTEXT predefined item. When the FIELDTEXT predefined item is altered, the altered value is displayed. For example, if the FIELDTEXT predefined item contains leading zeros, formatting is not considered when significance is determined by QUICK. Therefore, a field with a picture of `"^^^.^^"` and a significance of five would display "00.01" if the string "0001" is assigned to the FIELDTEXT predefined item in the OUTPUT procedure.

Error Handling in the OUTPUT Procedure

If an error occurs during the execution of the OUTPUT procedure, the rest of the procedure is skipped, crosshatches (#) are displayed in the field, and processing continues.

Example

The following example demonstrates how the OUTPUT procedure can manipulate field values between the time they're entered and the time they're displayed. In this example:

- Employees' names are prefixed with a title that depends on their sex and marital status.
- The size of the field LASTNAME must be large enough to accommodate both the employee's last name and title.

```
> SCREEN EMPLOYEE
>
> FILE EMPLOYEES
>
> ALIGN (1,4,25)
>
> FIELD EMPNUM OF EMPLOYEES REQUIRED NOCHANGE &
>   LOOKUP NOTON EMPLOYEES
```

```
> FIELD DEPARTMENT OF EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> FIELD SEX OF EMPLOYEES
> FIELD MARITALSTATUS OF EMPLOYEES
>
> PROCEDURE OUTPUT LASTNAME
>   BEGIN
>     IF SEX = "M"
>       THEN LET FIELDTEXT = "Mr. " + LASTNAME
>     IF SEX = "F" AND MARITALSTATUS = "M"
>       THEN LET FIELDTEXT = "Mrs. " + LASTNAME
>     IF SEX = "F" AND MARITALSTATUS = "S"
>       THEN LET FIELDTEXT = "Ms. " + LASTNAME
>     END
>
> BUILD
```

PATH

Establishes the method of record retrieval for the FIND procedure.

Syntax

PROCEDURE PATH

Discussion

In Find or Select mode, the PATH and FIND procedures control the retrieval of data records of PRIMARY and SECONDARY files. The PATH procedure analyzes responses made by the QUICK screen user and sets the retrieval path based on these responses.

Notes: Assigning values to record items within the scope of the PATH procedure does not change the data record status.

Restrict the PATH procedure to determining the retrieval paths for the files to be found. It's not advisable to directly modify the generated PATH and FIND procedures, or to write completely new procedures that replace them. Any changes that cannot be made in the ACCESS statement should be made in the POSTPATH and POSTFIND procedures.

For more information about verb and procedure compatibility, see (p. 239).

How the PATH Procedure Works

The PATH procedure executes when the user initiates either Find mode or Select mode. Find mode is initiated when a QUICK screen user enters the F (Find) command in the Action field. Similarly, Select mode is initiated when a QUICK screen user enters the S (Select) command in the Action field.

The PATH procedure sets the value of the PATH predefined item, which is used by all Find and Select operations. The PATH predefined item is always assigned an integer value that's used by the FIND procedure to determine how to retrieve data records.

The value of PATH is based on whether or not the screen user enters non-null values in response to REQUEST verbs in the PATH procedure. A non-null response to a REQUEST verb for a given field causes QUICK to set the PROMPTOK predefined condition to TRUE for that field. The PATH procedure then sets the value of the PATH predefined condition based on the PROMPTOK value.

Once the PATH procedure has executed and the predefined item PATH has been set, the FIND procedure tests the value of PATH and initiates access corresponding to that value.

Once a QUICK screen user makes a non-null entry in response to a REQUEST verb, QUICK moves the entered value into the associated item in the selection buffer. This value is kept for any subsequent initializations of the record buffer, and survives the initialization phase at the beginning of each retrieval sequence. In other words, a value entered in response to a REQUEST verb stays in the selection buffer until a new value is entered for that field in a subsequent Find or Select action.

The Default PATH Procedure

QDESIGN automatically generates a PATH procedure for a PRIMARY file. The generated PATH procedure is determined by the way in which the PRIMARY file is indexed, and by any ACCESS statements that are specified for the PRIMARY file.

To change PATH and FIND processing, use ACCESS statements to force changes in the generated PATH and FIND procedures. Using ACCESS statements, you can specify both the retrieval alternatives and the sequence of their use. If ACCESS statements are specified for the PRIMARY file, they completely override the default set of retrieval alternatives.

The default PATH procedure contains one REQUEST verb for each segment of each index for the PRIMARY record-structure. QDESIGN generates REQUEST verbs for all segments of the record-structure's indexes in the order that the indexes appear in the data dictionary.

The default PATH procedure is different for PANEL and NOPANEL screens.

When the PANEL option of either the SET or the SCREEN statement is in effect, the REQUEST verbs for segments in each index are grouped into separate blocks by the BLOCK TRANSFER control structure. No BLOCK TRANSFER control structure are generated unless the PANEL option of either the SET or SCREEN statement is specified.

In addition to REQUEST verbs, the PATH procedure contains one or more LET verbs for the predefined item PATH. The number of LET verbs depends on how the PRIMARY file is indexed, and on any ACCESS statements that exist for the PRIMARY file. QDESIGN's generated PATH procedure contains one LET verb for each ordered subset of segments in each index for the PRIMARY record-structure, plus one for sequential access. An ordered subset of segments in a PowerHouse index consists of the first segment, or the first and second segments, or the first, second and third segments, and so on up to the number of segments in the index.

In general, there are "n + 1" retrieval alternatives generated automatically for each screen, where n is the total number of segments of all indexes in the primary record-structure. For QDESIGN to construct indexed access properly, the index segments of the file must also be fields on the screen.

If the PRIMARY record-structure has no indexes, only one LET verb is generated and only sequential access is available.

During the execution of this procedure, the FINDMODE predefined condition is true. SELECTMODE is true if the Select action is in effect.

If data records can be retrieved without the user supplying information (if the information is passed from a higher-level screen, for example), the PATH procedure sets the value of the predefined item PATH to 1; there is only one retrieval method.

The PATH Procedure and MENU and SLAVE Screens

For MENU and SLAVE screens, QDESIGN generates PATH and FIND procedures that contain only the NULL verb.

Disabling Sequential Access

To suppress sequential retrieval, you can either

- include the NOSEQUENTIAL option on the SCREEN statement

  or

- not include an ACCESS statement with the SEQUENTIAL option when other ACCESS statements are specified

Both actions cause QDESIGN to suppress the generation of a sequential path option in the generated PATH procedure.

Use of the SUBPATH Predefined Item

The PATH and SUBPATH predefined items are used together to determine the method of data retrieval. The generated PATH procedure sets the value of SUBPATH depending on search criteria entered by the user. SUBPATH is then used in the FIND procedure to select the most appropriate cursor OPEN statement.

The value of SUBPATH depends on the number of linkitems in the VIA list that are character. These items cause a BETWEEN option to be generated for the WHERE option, which is used when the user enters a generic search value.

SUBPATH is only valid for relational data accessed through a CURSOR statement. It is also only valid for generic retrieval on character fields. If NOGENERIC is used on the ACCESS statement, only one variation is possible and SUBPATH is not used.

Error Handling in the PATH Procedure

If an error occurs during the execution of the PATH procedure, the rest of the procedure is skipped and QUICK prompts at the Action field without completing the retrieval cycle.

Examples

The relationship of a MASTER file to a PRIMARY file is similar to the relationship of a PRIMARY file to a SECONDARY file.

The following example illustrates a SKILLS screen in which the EMPLOYEES file is passed to the screen as a MASTER file. QDESIGN doesn't need to construct a PATH procedure to prompt the user for an index of the SKILLS record-structure; a relationship already exists between the passed EMPLOYEES record-structure and the SKILLS record-structure.

QDESIGN recognizes this relationship because the item named EMPLOYEE is in both the SKILLS and EMPLOYEES record-structures. Consequently, QDESIGN constructs the following PATH and FIND procedures:

```
> SCREEN SKILL RECEIVING EMPLOYEES
> FILE EMPLOYEES MASTER
> FILE SKILLS PRIMARY OCCURS 12
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF EMPLOYEES.
>
> CLUSTER OCCURS WITH SKILLS
> FIELD SKILL OF SKILLS REQUIRED NOCHANGE
> BUILD LIST
.
.
.
> PROCEDURE PATH
> BEGIN
>    LET PATH = 1
>    END
>
> PROCEDURE FIND
>   BEGIN
>     FOR SKILLS
>       BEGIN
>         GET SKILLS VIA EMPLOYEE
>         END
>     END
```

Using the ACCESS Statement to Control the PATH Procedure

QDESIGN constructs the PATH procedure based on the indexes that are declared for the PRIMARY file. If you specify access explicitly for the PRIMARY file by including an ACCESS statement, QDESIGN modifies the PATH to match the indexes and segments that are specified in the ACCESS statement.

In the following example, the PATH procedure is based on the ACCESS statements specified for the PRIMARY record-structure. QDESIGN builds two paths: one based on the index EMPLOYEE (using segment EMPLOYEE) and a second one based on the index LASTNAME (using segment LASTNAME). If no values are entered for either of these fields in the QUICK screen, QUICK issues an error message, since no alternative access paths are specified (such as ACCESS SEQUENTIAL).

The FIND procedure uses the value for PATH to retrieve data records via one of the retrieval alternatives.

The LIST option of the BUILD statement causes QDESIGN to display the generated procedures.

```
> SCREEN MODEMP  &
>      ACTIVITIES FIND, CHANGE, DELETE
>
> FILE EMPLOY1 PRIMARY
>      ACCESS VIAINDEX EMPLOYEE &
>         USING EMPLOYEE OF EMPLOY1 &
>         REQUEST EMPLOYEE
>      ACCESS VIAINDEX LASTNAME &
>         USING LASTNAME OF EMPLOY1 &
>         REQUEST LASTNAME
>
> TITLE "Modify Employee Attributes" CENTERED AT 4,1
>
```

```
> SKIP 2
> ALIGN (1,4,21)
> FIELD EMPLOYEE OF EMPLOY1 &
>      REQUIRED &
>      NOCHANGE &
>      LOOKUP ON EMPLOY1
> SKIP 1
>
> FIELD LASTNAME OF EMPLOY1 &
>      REQUIRED &
>      NOCHANGE
> FIELD CITY OF EMPLOY1
> FIELD STREET OF EMPLOY1
> FIELD PROVSTATE OF EMPLOY1 &
>   ID SAME &
>   LABEL "Prov/State"
.
.
.
> FIELD POSTALZIP OF EMPLOY1 &
>      ID SAME &
>      LABEL "PCode or Zip"
>
> FIELD PHONENUMBER OF EMPLOY1 &
>      ID SAME &
>      LABEL "Phone"
>
> BUILD LIST DETAIL
.
.
.
> PROCEDURE PATH
>   BEGIN
>     REQUEST EMPLOYEE OF EMPLOY1
>     IF PROMPTOK
>       THEN LET PATH = 1
>     IF PATH = 0
>       THEN BEGIN
>         REQUEST LASTNAME OF EMPLOY1
>         IF PROMPTOK
>           THEN LET PATH = 2
>         END
>     IF PATH = 0
>       THEN ERROR "Key/Index required."
>   END
>
> PROCEDURE FIND
>   BEGIN
>     IF PATH = 1
>        THEN GET EMPLOY1 VIAINDEX EMPLOYEE &
>        USING EMPLOYEE OF EMPLOY1
>     IF PATH = 2
>        THEN GET EMPLOY1 VIAINDEX LASTNAME &
>        USING LASTNAME OF EMPLOY1
>   END
.
.
.
```

The following example shows how the generated PATH and FIND procedures are affected by the use of SQL:

```
>   SQL DECLARE EMPLIST CURSOR FOR &
>   SELECT EMPLOYEE, FIRST_NAME, LAST_NAME, &
>      BRANCHES.BRANCH, BRANCH_NAME &
>      FROM EMPLOYEES, BRANCHES &
>      WHERE EMPLOYEES.BRANCH = BRANCHES.BRANCH
>   SCREEN EMPBRANCHC
>   CURSOR EMPLIST PRIMARY KEY EMPLOYEE
```

```
>  ACCESS VIA EMPLOYEE REQUEST EMPLOYEE
>  ACCESS VIA LAST_NAME REQUEST LAST_NAME
>  ACCESS SEQUENTIAL
.
.
.
> PROCEDURE PATH
>   BEGIN
>     REQUEST EMPLOYEE OF EMPLIST
>     IF PROMPTOK
>       THEN LET PATH = 1
>     IF PATH = 0
>       THEN BEGIN
>         REQUEST LAST_NAME OF EMPLIST
>         IF PROMPTOK
>           THEN LET PATH = 2
>         END
>     IF PATH = 0
>       THEN BEGIN
>         LET PATH = 3
>         END
>     IF PATH = 1
>       THEN BEGIN
>         LET SUBPATH = 0
>         END
>     IF PATH = 2
>       THEN BEGIN
>         IF RANGED( EMPLOYEES.LAST_NAME )
>           THEN LET SUBPATH = 1
>           ELSE LET SUBPATH = 0
>         END
>     END
> PROCEDURE FIND
>   BEGIN
>     IF NOT CURSOROPEN( EMPLIST )
>       THEN BEGIN
>       IF PATH = 1
>        THEN BEGIN
>         IF SUBPATH = 0
>          THEN SQL OPEN EMPLIST &
>           where( EMPLOYEES.EMPLOYEE &
>             =:linkvalue( EMPLOYEES.EMPLOYEE ,equal) )
>             END
>         IF PATH = 2
>          THEN BEGIN
>           IF SUBPATH = 1
>            THEN SQL OPEN EMPLIST &
>             where( EMPLOYEES.LAST_NAME &
>             between :linkvalue( EMPLOYEES.LAST_NAME &
>             ,lowest) and &
>             :linkvalue( EMPLOYEES.LAST_NAME &
>             ,highest) )
>           IF SUBPATH = 0
>            THEN SQL OPEN EMPLIST &
>             where( EMPLOYEES.LAST_NAME &
>             =:linkvalue( EMPLOYEES.LAST_NAME ,equal) )
>             END
>         IF PATH = 3
>           THEN BEGIN
>             SQL OPEN EMPLIST
>             END
>         END
>     SQL FETCH EMPLIST
>     END
```

POSTFIND

Performs processing after successful completion of the FIND procedure.

Syntax

PROCEDURE POSTFIND

Discussion

The POSTFIND procedure allows QUICK to perform processing immediately following the completion of the FIND procedure. This procedure is intended to supplement the FIND procedure rather than to replace it. The POSTFIND procedure can be used to retrieve data records based on data that is already retrieved.

Note: For more information about verb and procedure compatibility, see (p. 239).

When the POSTFIND Procedure is Initiated

The POSTFIND procedure executes after the FIND procedure finishes, but before the retrieved data is displayed on the QUICK screen.

Error Handling in the POSTFIND Procedure

If an error occurs during the execution of this procedure, the rest of the procedure is skipped. QUICK then prompts at the Action field without displaying retrieved data.

During the execution of the POSTFIND procedure, the CHANGEMODE predefined condition is true.

Example

The POSTFIND procedure in the following example sums values for individual project billings into a temporary item.

```
> SCREEN PROJBILL
>
> TEMPORARY TOTBILLINGS NUMERIC *8
> FILE PROJECTS PRIMARY
> FILE BILLINGS DETAIL OCCURS 6
Item PROJNO initialized (fixed) to PROJNO of PROJECTS
> FILE BILLINGS ALIAS BILLDES DESIGNER OPEN 1
>
> FIELD TOTBILLINGS &
>   NOID &
>   LABEL "Billings to Date" AT 3,50 &
>   DATA AT ,67 &
>   PICTURE "^^^,^^^.^^" DISPLAY
> SKIP 1
> FIELD PROJNO OF PROJECTS &
>   REQUIRED NOCHANGE &
>   LOOKUP NOTON PROJECTS
> FIELD PROJNAME OF PROJECTS
> FIELD PROJMGR OF PROJECTS
> FIELD PROJBUDG OF PROJECTS
>
> SKIP 1
> TITLE "Employee" AT ,4
> TITLE "Month" AT ,15
> TITLE "Billing" AT ,25
> TITLE "Employee" AT ,44
> TITLE "Month" AT ,55
> TITLE "Billing" AT ,65
> ALIGN (1,,4) (,,15) (,,25)
> SKIP 1
>
```

```
> CLUSTER OCCURS WITH BILLINGS FOR 1,39
>   FIELD EMPNUM OF BILLINGS REQUIRED NOCHANGE
>   FIELD MONTH OF BILLINGS
>   FIELD BILLING OF BILLINGS
> CLUSTER
.
.
.
> PROCEDURE POSTFIND
>   BEGIN
>     WHILE RETRIEVING BILLDES VIA PROJNO &
>       USING PROJNO OF PROJECTS
>     LET TOTBILLINGS = &
>     TOTBILLINGS + BILLINGS OF BILLDES
>   END
>
> BUILD
```

POSTPATH

Performs processing following successful completion of the PATH procedure and just before the FIND procedure.

Syntax

PROCEDURE POSTPATH

Discussion

The POSTPATH procedure allows QUICK to perform processing immediately before the record-retrieval cycle begins. The POSTPATH procedure is intended to supplement the PATH procedure rather than to replace it.

Note: For more information about verb and procedure compatibility, see (p. 239).

When the POSTPATH Procedure is Initiated

The POSTPATH procedure is executed after the completion of the PATH procedure, after selection values are entered in Select mode, and before the execution of the FIND procedure.

Error Handling in the POSTPATH Procedure

If an error occurs during the execution of this procedure, the rest of the procedure is skipped and QUICK prompts the user in the Action field without executing the retrieval cycle

During the execution of this procedure, the FINDMODE predefined condition is true. SELECTMODE is true if the Select action is in effect.

Record Status in the POSTPATH Procedure

Assigning values to items within the scope of the POSTPATH procedure doesn't change the data record status.

Example

This product information screen design informs the screen user that sequential retrieval of data may take a few minutes.

If the screen user responds to the FIND mode prompt by pressing [Return] without specifying a retrieval value, data is retrieved sequentially.

```
> SCREEN PRODUCT
>
> FILE PRODUCTS PRIMARY
>   SELECT IF PRODUCTNUMBER > 10000  &
>     AND PRODUCTNUMBER < 50000
>
> FIELD PRODUCTNUMBER OF PRODUCTS &
>   REQUIRED NOCHANGE LOOKUP NOTON PRODUCTS
> FIELD PRODDESCRIPTION OF PRODUCTS
> FIELD COST OF PRODUCTS
> FIELD QUANTITY OF PRODUCTS
>
> PROCEDURE PATH
>   BEGIN
>     REQUEST PRODUCTNUMBER OF PRODUCTS
>     IF PROMPTOK
>       THEN LET PATH = 1
>     IF PATH = 0
>       THEN LET PATH = 2
>     END
>
> PROCEDURE POSTPATH
>   BEGIN
```

```
>    IF PATH = 2
>     THEN INFORMATION &
>     "Your request may take a few minutes. Please wait."
>     END
>
> BUILD
```

POSTSCROLL

Performs processing after the occurrence window has scrolled but before it is re-displayed.

Syntax

PROCEDURE POSTSCROLL

Discussion

QDESIGN does not generate a default POSTSCROLL procedure.

When QUICK determines that a command will cause the occurrence window to move, it takes the following steps:

- If there is a PRESCROLL procedure, QUICK invokes it.
- QUICK moves the occurrence window accordingly.
- If there is a POSTSCROLL procedure, QUICK invokes it.
- QUICK re-displays the occurrence window to show the changes.

The mode (CHANGEMODE, CORRECTMODE, ENTRYMODE, FINDMODE, SELECTMODE) does not change for this procedure; it remains the same as the current context.

Note: For more information about verb and procedure compatibility, see (p. 239).

Error Handling in the POSTSCROLL Procedure

When an error occurs in the POSTSCROLL procedure, QUICK skips the rest of the procedure, issues an error message and prompts for the next action.

Example

The following screen is used to browse through employees. The POSITION field indicates which records are currently being viewed. In this example:

- NUM_EMPLOYEES is the number of employee records in the cache.
- The FOR DISPLAYED loop is used to determine the first and last records displayed.
- The first FOR loop is used to determine the number of records entered or retrieved.
- The second FOR loop is used to initialize the record counter.
- The internal procedure, UPDATE_POSITION, is called to set the initial value of POSITION. The POSTSCROLL procedure is not called when the records are initially displayed, only.
- The POSTSCROLL procedure is invoked every time the records are scrolled. This procedure uses the internal procedure, UPDATE_POSITION, to update the POSITION field.

```
> SCREEN SCROLL_EMPLOYEES
>
> FILE EMPLOYEES OCCURS 15 CACHE 40
>
> TEMPORARY EMPLOYEE_NUM  INTEGER*3 UNSIGNED   &
>   OCCURS WITH EMPLOYEES
> TEMPORARY FIRST_EMPLOYEE RESET AT STARTUP
> TEMPORARY LAST_EMPLOYEE  RESET AT STARTUP
> TEMPORARY NUM_EMPLOYEES  RESET AT STARTUP
>
> DEFINE POSITION CHARACTER*60 = SUBSTITUTE   &
>   ( "You are viewing records ^ to ^ out of ^.", &
>             ASCII( FIRST_EMPLOYEE ), &
>             ASCII( LAST_EMPLOYEE  ), &
>             ASCII( NUM_EMPLOYEES  ) )
> ALIGN (11,,15) (,,20) (,,28) (,,43)
> TITLE "Record  Employee    First            Last"  &
>   AT 4,11
> TITLE "Number   Number      Name             Name"  &
>   AT 5,11
```

```
> CLUSTER OCCURS WITH EMPLOYEES
> FIELD EMPLOYEE_NUM PICTURE "^^^" SIGNIFICANCE 3 DISPLAY
> FIELD EMPLOYNO OF EMPLOYEES REQUIRED NOCHANGE  &
>   LOOKUP NOTON EMPLOYEES
> FIELD FIRSTNAME OF EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> CLUSTER
> SKIP
> ALIGN (,,11)
> FIELD POSITION ID 99
>
> PROCEDURE INTERNAL UPDATE_POSITION
>   BEGIN
>     LET FIRST_EMPLOYEE = 0
>     FOR DISPLAYED EMPLOYEES
>       BEGIN
>         IF FIRST_EMPLOYEE = 0
>         THEN
>           LET FIRST_EMPLOYEE = OCCURRENCE
>         ELSE
>           LET LAST_EMPLOYEE  = OCCURRENCE
>       END
>     FOR EMPLOYEES
>       BEGIN
>         LET NUM_EMPLOYEES = OCCURRENCE
>       END
>     DISPLAY POSITION
>   END
> PROCEDURE POSTFIND
>   BEGIN
>     FOR EACH EMPLOYEES
>       BEGIN
>         LET EMPLOYEE_NUM = OCCURRENCE
>       END
>     DO INTERNAL UPDATE_POSITION
>   END
>
>
>
> PROCEDURE POSTSCROLL
>   BEGIN
>     DO INTERNAL UPDATE_POSITION
>   END
```

```
MODE:F ACTION:

        Record  Employee    First            Last
        Number   Number      Name            Name
        01  016   1015   LINDA          RYAN
        02  017   1016   KEILA          NIKKANEN
        03  018   1017   DONNA          MCPHERSON
        04  019   1018   HEATHER        SAMPSON
        05  020   1019   GLENN          WALKER
        06  021   1020   JUDY           CLARK
        07  022   1021   PAUL           LARSON
        08  023   1022   LINDA          MANNING
        09  024   1023   THOMAS         WILLIAMS
        10  025   1024   SAMANTHA       KELLY
        11  026   1025   MAURICE        PHILLIPS
        12  027   1026   EDWARD         MITCHELL
        13  028   1027   MALCOLM        TIERNEY
        14  029   1028   CLAYTON        HARWOOD
        15  030   1029   SUZANNE        KNIGHT

        You are viewing records 16 to 30 out of 37.
```

POSTUPDATE

Performs processing after successful completion of the UPDATE procedure.

Syntax

PROCEDURE POSTUPDATE [RECOVERABLE]

RECOVERABLE

Instructs QUICK to keep rollback information for PUTs done to non-relational records during the POSTUPDATE procedure. This option has no effect on PUTs to relational records since rollback information is always retained for relational records.

Discussion

You can write the optional POSTUPDATE procedure to perform processing after the successful completion of the UPDATE procedure. The POSTUPDATE procedure is intended to supplement the UPDATE procedure rather than to replace it.

Note: For more information about verb and procedure compatibility, see (p. 239).

When the POSTUPDATE Procedure is Initiated

The POSTUPDATE procedure is initiated after files are updated, but before QUICK proceeds to the next processing cycle. For example, you can use the POSTUPDATE procedure to control the submission of batch jobs that update other files. If the AUTORETURN option is specified on the SCREEN statement, the POSTUPDATE procedure is performed before QUICK returns to the higher-level screen.

During the execution of this procedure, one of the predefined conditions, CHANGEMODE or CORRECTMODE, is true.

Error Handling in the POSTUPDATE Procedure

If an error occurs during the execution of this procedure, the rest of the procedure is skipped, and QUICK prompts the user at the Action field.

If an error occurs in the POSTUPDATE procedure, any updates performed by the POSTUPDATE procedure before the occurrence of the error, and any updates performed by the UPDATE procedure are not rolled back.

Rollbacks

If any error conditions are detected in the middle of a multiple record or multiple file update, a special rollback facility restores the files to the state that they were in before the update began. This applies to PUT verbs in the update procedure. Outside the update procedure, PUT verbs are only rolled back in procedures using the RECOVERABLE option.

If the RECOVERABLE option is used, PUT verbs to non-relational records are rolled back automatically by PowerHouse on encountering an error.

The rollback provisions remove most causes of inconsistent data. However, there are three exceptions:

- If a failed update encounters another failed condition while attempting to rollback, the files may remain inconsistent. In this situation, QUICK tells the user what has happened.
- Append-type files cannot be restored to their original state. If append-type audit files are involved in the rollback, an audit record is written to reflect the update done by rollback processing. For example, when a record is deleted, a record reflecting the delete is added to the end of the audit file. If that record is subsequently rolled back, a record reflecting the previous state of the record (with the record added) is then added to the end of the audit file.
- Relational systems provide their own rollback mechanism. Therefore, for tables and views, QUICK invokes a database rollback when errors are encountered, rather than invoking its own rollback mechanism.

Part of PUT verb processing is the update of rollback information with a copy of the record as it existed before and after the update.

To accommodate the potentially large volume of information that must be saved to accomplish rollback (especially for the records of delete files), a three-level storage scheme is used. A relatively small buffer is allocated in the primary memory that accommodates any small-volume updates. Extra virtual memory is used for spillovers from the primary memory buffer. Temporary files are used when the volume of data specified is too great or an execution-time parameter exceeds a limit.

Sufficient storage must be available for rollback recovery of any given update. Otherwise, QUICK stops the update, rolls back the buffers to the starting values while it still has all the data, and issues an error message.

When QUICK updates an existing record, it can tell whether the record has been updated by some other process during the time span between the first retrieval and update. This is accomplished by comparing a checksum calculated when the record was first read to a checksum calculated when the record is read again prior to updating. If there's a difference between the two checksums, QUICK halts the update process, issues a message, and restores the files to the state they were in before the update began.

The checksum calculation omits:

- calculated columns. If they were included, the values could have been changed by the database, resulting in a checksum mismatch. This can easily occur if the user does an Update Stay. Removing calculated columns from the checksum calculation eliminates these false errors.
- blob columns. These are excluded from the checksum calculation for performance reasons, as they can be very large.
- relational columns not referenced by the screen. These are excluded because the checksum is based on the underlying SQL generated for the QUICK screen.

Users who write their own QDESIGN procedures must handle rollbacks themselves in certain instances. Only PUT verbs that are used in the UPDATE procedure, or procedures with the RECOVERABLE option, can take advantage of PowerHouse's full rollback. PUT verbs that appear outside these procedures don't automatically roll back if an error condition occurs.

If a subscreen is called from the UPDATE procedure without the KEEP ROLLBACK option on the RUN SCREEN verb, all rollback information is lost and the rollback isn't possible. Either use the KEEP ROLLBACK option or call the subscreen from the POSTUPDATE procedure (which is only performed after successful completion of the UPDATE procedure).

Conversion

Prior to 7.33.C (**UNIX**), 7.10E1 (**OpenVMS**) and 8.09 (**MPE/iX**), no automatic rollback was available to cancel the effects of PUT verbs to non-relational records executed outside of the UPDATE procedure. If PUT verbs to non-relational records were used outside of the UPDATE procedure, you would have to write a BACKOUT procedure to reverse the effects of the PUT verbs.

If you use the RECOVERABLE option, any BACKOUT procedures that you have coded to handle rollbacks must be reviewed to ensure that a "double rollback" is not performed.

Example

This screen allows users to enter new part numbers in an order entry system. The part numbers are assigned automatically from a DESIGNER file named NEXTCODE.

All parts are uniquely defined by a two segment index consisting of a PARTNUMBER and a PARTVARIANT. As soon as the user has finished entering a new part and has performed an update, the POSTUPDATE procedure prompts to determine if there are more PARTVARIANT values to be added for the current PARTNUMBER.

QUICK invokes a subscreen based on the user's response to the prompt issued in the MORE field.

```
> SCREEN ADDPART  &
>   NOMODE  &
>   ACTION LABEL "==>" AT 20,1 &
```

```
>   ACTIVITIES ENTRY
>
> FILE PARTS PRIMARY
> FILE NEXTCODE DESIGNER
>   ACCESS USING 1
> TEMPORARY PARTNUM NUMERIC *4
>
> TEMPORARY PARTNAME CHARACTER *12
>
> TEMPORARY MORE CHARACTER *1 &
>   INITIAL "N"
> TITLE "Add New Part" CENTERED AT 4,1
.
.
.
> FIELD MORE &
>   REQUIRED PREDISPLAY NOID &
>   DATA AT 4,40 OMIT ON ENTRY &
>   VALUES "Y", "N", "y", "n" &
>   LABEL "More Variants for this Part?"
.
.
.
> PROCEDURE POSTUPDATE
> BEGIN
>   LET PARTNUM = PARTNUMBER OF PARTS
>   LET PARTNAME = PARTNAME OF PARTS
>   PROMPT MORE
>   IF MORE = "Y"
>     THEN RUN SCREEN ADDVAR1 &
>          PASSING PARTNUM, PARTNAME
>   RETURN
> END
>
> BUILD
```

PREENTRY

Performs processing at the beginning of the entry sequence.

Syntax

PROCEDURE PREENTRY

Discussion

The PREENTRY procedure is an optional procedure that you can write to perform tests, calculations, or initializations at the beginning of the entry sequence.

Note: For more information about verb and procedure compatibility, see (p. 239).

When the PREENTRY Procedure is Initiated

The PREENTRY procedure is initiated by the entry sequence. QUICK performs the processing specified by this procedure in the entry phase of Entry mode processing. For information about Entry mode, see (p. 248).

The PREENTRY procedure is not executed in a slave screen because the ENTRY procedure is considered as an extension to the ENTRY procedure in the calling screen

Error Handling in the PREENTRY Procedure

If QUICK encounters an error, it backs up to the last ACCEPT verb or BLOCK TRANSFER control structure. If there is no ACCEPT verb or control structure, then QUICK prompts at the Action field without executing the ENTRY procedure.

The ENTRYMODE predefined condition is true during the PREENTRY procedure.

Example

The following screen uses a PREENTRY procedure to obtain a value from a control file when assigning an account number automatically to customers.

In this example, the PREENTRY procedure retrieves the next available customer number from the NEXTCODE record-structure. This number is converted to a string, concatenated with the letter "C", and is assigned to the item CUSTOMERKEY of CUSTOMERS.

```
> SCREEN ADDCUST  &
>   NOMODE  &
>   ACTION LABEL "===>" AT 20,1 &
>   ACTIVITIES ENTRY
>
> FILE CUSTOMERS PRIMARY
> FILE NEXTCODE DESIGNER
>   ACCESS USING 3
>
>
> TITLE "Add New Customer" CENTERED AT 4,1
>
> DRAW 7,15 TO 19,65
>
> SKIP TO LINE 9
>
> ALIGN (20,25,40)
> FIELD CUSTOMERKEY OF CUSTOMERS &
>   DISPLAY &
>   LOOKUP NOTON CUSTOMERS
> FIELD CUSTOMERNAME OF CUSTOMERS
> FIELD STREET OF CUSTOMERS
> FIELD CITY OF CUSTOMERS &
>   ID SAME
> FIELD PROVSTATE OF CUSTOMERS &
>   LABEL "Prov/State" &
```

```
>   ID SAME
> FIELD POSTALZIP OF CUSTOMERS &
>   LABEL "PCode/Zip" &
>   ID SAME
> FIELD PHONENUMBER OF CUSTOMERS &
>   ID SAME &
>   BWZ
.
.
.
> PROCEDURE PREENTRY
> BEGIN
>   GET NEXTCODE USING 3
>   LET CUSTOMERKEY OF CUSTOMERS = &
>     "C" + PACK(ASCII(CODE OF NEXTCODE))
>   LET CODE OF NEXTCODE = CODE OF NEXTCODE + 1
>   PUT NEXTCODE AT 3
>   CLOSE NEXTCODE
>   DISPLAY CUSTOMERKEY OF CUSTOMERS
> END
>
> BUILD
```

PRESCROLL

Performs processing before scrolling occurs.

Syntax

PROCEDURE PRESCROLL

Discussion

QDESIGN does not generate a default PRESCROLL procedure.

When QUICK determines that a command will cause the occurrence window to move, it takes the following steps:

- If there is a PRESCROLL procedure, QUICK invokes it.
- QUICK moves the occurrence window accordingly.
- If there is a POSTSCROLL procedure, QUICK invokes it.
- QUICK re-displays the occurrence window to show the changes.

The mode (CHANGEMODE, CORRECTMODE, ENTRYMODE, FINDMODE, SELECTMODE) does not change for this procedure; it remains the same as the current context.

QUICK action commands that change the occurrence window include:

FIRST RECORD	LAST RECORD	
NEXT	NEXT DATA	PREVIOUS DATA
NEXT RECORD	PREVIOUS RECORD	
UPDATE	UPDATE NEXT	

Any user action (such as Skip All (///) or Backup (\)) when entering, modifying, or appending data records, may require QUICK to move the occurrence window. In addition, QUICK automatically moves the occurrence window when it is full and there are empty record buffers in the cache not currently displayed.

Notes: Any verb that prompts for input may cause QUICK to reposition the occurrence window, and as QUICK is already in the process of moving the window, the results are unpredictable.

For more information about verb and procedure compatibility, see (p. 239).

Error Handling in the PRESCROLL Procedure

When an error occurs in the PRESCROLL procedure, QUICK skips the rest of the procedure and stops the command or verb that caused this procedure to be invoked. This stops QUICK from scrolling. QUICK issues an error message and prompts for the next action. QUICK uses the backup stack to determine where to prompt if the scrolling was initiated automatically during field processing.

Example

The PRESCROLL procedure can be used to prevent scrolling before an Update action occurs. The following example checks each on-screen record to see if changes have been made. If any changes have been made, the scrolling action is not allowed. The user must update before leaving the screen.

```
> PROCEDURE PRESCROLL
>   BEGIN
>      FOR EACH DISPLAYED EMPLOYEE
>         IF ALTEREDRECORD
>      THEN ERROR &
>         "An Update must be performed before scrolling"
```

```
> END
```

PREUPDATE

Performs processing prior to the UPDATE procedure when the user enters one of the update commands.

Syntax

PROCEDURE PREUPDATE [RECOVERABLE]

RECOVERABLE

Instructs QUICK to keep rollback information for PUTs done to non-relational records during the PREUPDATE procedure. This option has no effect on PUTs to relational records.

Discussion

You can write the optional PREUPDATE procedure to perform field processing and editing prior to the UPDATE procedure. The PREUPDATE procedure is started when one of the update commands is entered, or automatically at the end of the standard entry sequence for screens with the AUTOUPDATE option.

The PREUPDATE procedure usually consists of one or more edits that can lead to error messages. QUICK processes any infield validation statements in the PREUPDATE procedure before it performs an update in response to any of the update commands. If any error messages are issued, updating isn't performed.

Note: For more information about verb and procedure compatibility, see (p. 239).

When the PREUPDATE Procedure is Initiated

The PREUPDATE procedure is initiated when the QUICK screen user enters one of the update commands when there are changes pending.

In addition, the PREUPDATE procedure is executed automatically at the end of the standard entry sequence for screens with the AUTOUPDATE option.

During the execution of this procedure, one of the predefined conditions, CHANGEMODE or CORRECTMODE, is true.

Error Handling in the PREUPDATE Procedure

If an error occurs during the execution of this procedure, the rest of the procedure is skipped. Any updates performed by this procedure are rolled back if the procedure has the RECOVERABLE option, otherwise no updates are rolled back. QUICK then prompts the user at the Action field without performing the UPDATE procedure.

Rollbacks

If any error conditions are detected in the middle of a multiple record or multiple file update, a special rollback facility restores the files to the state that they were in before the update began. For more information, see (p. 322).

Conversion

Prior to 7.33.C (**UNIX**), 7.10E1 (**OpenVMS**), and 8.09 (**MPE/iX**), no automatic rollback was available to cancel the effect of PUT verbs to non-relational records executed outside of the UPDATE procedure. If PUT verbs to non-relational records were used outside of the UPDATE procedure, you would have to write a BACKOUT procedure to reverse their effect.

If you use the RECOVERABLE option, any BACKOUT procedures that you have coded to handle rollbacks must be reviewed to ensure that a "double rollback" is not performed.

Examples

The following PREUPDATE procedure does an edit check involving four separate fields entered by the user.

```
> PROCEDURE PREUPDATE
>   BEGIN
>     IF ORDERTOTAL  <> &
>       QUANTITY * (UNITPRICE + TAX)
>     THEN ERROR
>      "Order total does not compute correctly"
>   END
```

In the next example, if an error occurs on the PUT to RECORD_A in the PREUPDATE procedure, the QUICK screen stops processing and the PUT is rolled back.

If an error occurs on the PUT to RECORD_B in the UPDATE procedure, processing stops and both the PUT to RECORD_A and RECORD_B are automatically rolled back since both the UPDATE and PREUPDATE procedures are recoverable. The PUTs that take place in the two procedures are treated as a unit of work, and are, therefore, rolled back together.

Similarly, if a POSTUPDATE procedure exists with the RECOVERABLE option, the PUTs in all three procedures are treated as a unit and are all rolled back in the event of an error.

```
>   SCREEN RECOVER03
>   FILE RECORD_A DESIGNER
>   FILE RECORD_B PRIMARY
.
.
.
>   PROCEDURE PREUPDATE RECOVERABLE
>   BEGIN
>      PUT RECORD_A
>   END
>
>   PROCEDURE UPDATE
>   BEGIN
>      PUT RECORD_B
>   END
```

PROCESS

Performs processing after a new or changed value is entered in the named field.

Syntax

PROCEDURE PROCESS field

field

Names the field with which the PROCESS procedure is associated.

The PROCESS procedure is initiated by the ACCEPT verb for a field. For more information, see (p. 364).

Discussion

You can write the optional PROCESS procedure to allow processing based on entering or changing the associated item.

You can calculate and display field values directly in the PROCESS procedure. The values for FIELDTEXT and FIELDVALUE have already been moved to the record buffer by the time the PROCESS procedure is executed. This means that they might not be current, making references to them (or assignments from them to the record buffer) unpredictable. Because there's no danger of the calculated values being overwritten by either FIELDTEXT or FIELDVALUE, reference the field by name.

Note: For more information about verb and procedure compatibility, see (p. 239).

When the PROCESS Procedure is Initiated

The PROCESS procedure is initiated by an ACCEPT verb when a new or changed value is entered in the named field, or by an EDIT verb. The PROCESS procedure immediately follows the successful completion of an EDIT procedure (if there is one for the named field) and balance summing. At the time the procedure is invoked, the new value is stored in the record buffer or the temporary item buffer, and has already been accepted by QUICK.

The PROCESS procedure is performed only if the value entered for the field named in the PROCESS procedure passes editing. It isn't performed for null entries unless the default or duplicate value is moved to the item.

During execution of this procedure, one of CHANGEMODE, CORRECTMODE, ENTRYMODE, or FINDMODE is true, or they can all be false. If SELECTMODE is true, FINDMODE is also true.

Error Handling in the PROCESS Procedure

If the PROCESS procedure is called by the ACCEPT verb, the rest of the procedure is skipped and QUICK prompts at the Action field.

If the procedure is called by the EDIT verb, the rest of the procedure is skipped and QUICK continues error processing for the procedure containing the EDIT verb.

Examples

The following example demonstrates how a PROCESS procedure can be used to assign values to items depending on other item values. The PROCESS procedure is initiated when a value is entered in the CITY field. Whenever the value of CITY changes, the procedure attempts to change the value of PROVINCE automatically.

```
> SCREEN PROVINCE
>
> FILE INVOICES
> FILE GEOGRAPHY DESIGNER
> FIELD INVOICENO OF INVOICES REQUIRED NOCHANGE &
>     LOOKUP NOTON INVOICES
```

```
> FIELD INVOICEDATE OF INVOICES
> FIELD CITY OF INVOICES
> FIELD PROVINCE OF INVOICES IF PROVINCE &
>   OF INVOICES = " "
>
> PROCEDURE PROCESS CITY
>   BEGIN
>     GET GEOGRAPHY VIA CITY USING CITY &
>         OF INVOICES OPTIONAL
>     IF ACCESSOK
>       THEN BEGIN
>         LET PROVINCE = PROVINCE OF GEOGRAPHY
>         END
>     END
```

In the next example, there are four inter-related fields. The first must be entered, the last three are optional, and if any one is entered, the others may be calculated. The INPUT procedure is used to determine if the user has entered a value in the INCREMENT field.

```
.
.
.
> TEMPORARY ONEENTERED CHARACTER*1
.
.
.
> FIELD INITIALVALUE REQUIRED
> FIELD INCREMENT OF TESTVALUES
> FIELD PERCENTCHANGE OF TESTVALUES &
>   IF ONEENTERED <>  "Y"
> FIELD FINALVALUE OF TESTVALUES &
>   IF ONEENTERED <>  "Y"
>
> PROCEDURE INPUT INCREMENT
> BEGIN
>   IF 0 = SIZE(FIELDTEXT)
>     THEN LET ONEENTERED = "N"
> END
> PROCEDURE PROCESS INCREMENT
>   BEGIN
>     LET FINALVALUE = INITIALVALUE + INCREMENT
>     LET PERCENTCHANGE =
>         100 * INCREMENT/INITIALVALUE
>     DISPLAY FINALVALUE
>     DISPLAY PERCENTCHANGE
>     LET ONEENTERED = "Y"
>     END
>
> PROCEDURE PROCESS PERCENTCHANGE
>   BEGIN
>     LET FINALVALUE = INITIALVALUE * &
>         (1+PERCENTCHANGE/100)
>     LET INCREMENT = FINALVALUE - INITIALVALUE
>     DISPLAY FINALVALUE
>     DISPLAY INCREMENT
>     LET ONEENTERED = "Y"
>     END
>
> PROCEDURE PROCESS FINALVALUE
>   BEGIN
>     LET INCREMENT = FINALVALUE - INITIALVALUE
>     LET PERCENTCHANGE =
>       100 * INCREMENT/INITIALVALUE
>     DISPLAY INCREMENT
>     DISPLAY PERCENTCHANGE
>     END
```

The preceding example isn't complete. It works correctly in the standard Entry sequence and when a correction or change is made to one of the three optional fields. However, it does not work when the value of the INITIALVALUE field is changed or corrected. To rectify the problem, you can

- put the NOCORRECT NOCHANGE option on the FIELD statement for the INITIALVALUE item. This works but is somewhat restrictive since the item INITIALVALUE can't be changed.
- allow the change but include a PREUPDATE procedure that checks for the inconsistency and warns the user to adjust one of the other four fields
- include a PROCESS procedure for the INITIALVALUE field, as in

```
> PROCEDURE PROCESS INITIALVALUE
>  IF CORRECTMODE OR CHANGEMODE
>    THEN BEGIN
>     INFO &
>       "PERCENTCHANGE and FINALVALUE altered"
>     LET PERCENTCHANGE = 100 * INCREMENT/INITIALVALUE
>     LET FINALVALUE = INITIALVALUE + INCREMENT
>     DISPLAY FINALVALUE
>     DISPLAY PERCENTCHANGE
>     END
```

Remember that the field procedures are performed at several phases of processing, and users have wide flexibility in changing data after it has been entered. When performing calculations involving several fields, you must allow for this.

SELECT

Controls Select mode processing.

Syntax

PROCEDURE SELECT

Discussion

QUICK generates a default SELECT procedure only for PANEL screens. When no SELECT procedure exists for a QUICK screen, Select mode processing is handled by a predefined set of processing steps as described later in this section.

The Default SELECT Procedure

The default SELECT procedure contains a SELECT verb for the first occurrence of each field in the current screen. By default, the SELECT verbs within the SELECT procedure are blocked into units (based on CLUSTER statements) for processing with the BLOCK TRANSFER, BEGIN, and END control structures.

For more information about the BLOCK TRANSFER control structure, see (p. 372).

No SELECT verb is generated for fields that

- are marked as NOSELECT
- have a DISPLAY, FIXED, or SILENT option on the ITEM or FIELD statement
- contain temporary items

If no SELECT procedure is specified or generated by default for a QUICK screen, QUICK uses a standard process for selecting data records:

1. Run the PATH procedure, prompting the screen user for segment values.
2. Prompt the screen user for field ID-numbers. The numbers entered identify the fields in which QUICK will prompt for additional selection values.
3. Prompt the screen user for a value in each field identified in Step 2.

If a SELECT procedure exists, it replaces Step 2 of the standard select process. If QUICK is run with the **charmode=field** program parameter, the SELECT procedure is ignored.

During the execution of this procedure, the FINDMODE and SELECTMODE predefined conditions are true.

Example

The following example illustrates a simple SELECT procedure for a basic screen design. In this example, users of the screen that includes this SELECT procedure are limited to the fields ORDERDATE and QUANTITYORDERED in Select mode.

```
.
.
.
> PROCEDURE SELECT
>   BEGIN
>     SELECT ORDERDATE OF ORDERMASTER
>     SELECT QUANTITYORDERED OF ORDERMASTER
>   END
.
.
.
```

UPDATE

Controls update processing.

Syntax

PROCEDURE UPDATE

Discussion

Files are updated when new data records are created, or when old data records are changed or deleted. The updating activity is normally governed exclusively by the UPDATE procedure, and doesn't begin until the QUICK screen user enters one of the Update Action field commands.

Note: For more information about verb and procedure compatibility, see (p. 239).

The Default UPDATE Procedure

QDESIGN generates an UPDATE procedure automatically, assuming that MASTER, PRIMARY, SECONDARY, DETAIL, and DELETE files are to be updated

- when the user enters an Update command
- automatically at the end of the standard Entry sequence (for screens with the AUTOUPDATE option)

The standard UPDATE procedure controls all file updates. If there is an OCCURS option on a FILE statement, the corresponding PUT verb is included in a FOR control structure.

DELETE files are updated only if the PRIMARY records are marked for deletion. All AUDIT files are updated automatically with their related files. REFERENCE files are not updated. You must control all DESIGNER file updating.

Backing Out of an Update

If all updating is performed in the UPDATE procedure, QUICK screens allow the user to back out of the screen at any time without updating and without affecting the data in the file. Data in the file is always maintained at the state of the last successful update.

The PREUPDATE Procedure

If the PREUPDATE procedure is included in the screen design, it is performed first. If the PREUPDATE procedure completes successfully, balance checking is performed, and then the UPDATE procedure is initiated.

Error Handling in the UPDATE Procedure

If an error occurs during the execution of this procedure, the rest of the procedure is skipped. Any updates performed by the UPDATE procedure before the occurrence of the error are rolled back, and QUICK prompts at the Action field. This rollback facility removes most causes of inconsistent data records, with two exceptions:

- If a failed update reaches another error while attempting to rollback, the files may remain inconsistent. (QUICK issues an error message to this effect.) Note that data records added to SEQUENTIAL and DIRECT files can't be deleted. If rollback occurs and these types of files are present (and are not AUDIT files), the rollback fails.
- Append-type AUDIT files can't be restored to their original state. If these types of files are involved in the rollback, QUICK writes audit data records that reflect the updates performed by rollback processing.

Relational systems provide their own rollback mechanism. Therefore, for tables and views, QUICK invokes a database rollback when errors are encountered rather than invoking its own rollback mechanism.

Using PUT Verbs Outside of the UPDATE Procedure

QUICK normally updates backout buffers at the end of the UPDATE procedure. However, if a PUT verb is written outside the UPDATE procedure, QUICK updates its associated backout buffer immediately. QUICK updates backout copies of passed temporary items only when the screen user performs the UPDATE procedure. The use of PUT verbs outside of the UPDATE procedure is not recommended.

During the execution of this procedure, one of the predefined conditions, CHANGEMODE or CORRECTMODE, is true.

PUT Verb Order in the Default UPDATE Procedure

QDESIGN generates PUT verbs in the UPDATE procedure to do the actual updating. Generated UPDATE procedures have up to three sections depending on the screen requirements and the relationships between files and tables. There are three types of PUT verbs corresponding to the three sections:

- The add constraints section of the generated UPDATE procedure consists of PUT verbs with the NEW option. The NEW option states that the record status must be new before the PUT verb is executed. The PUT verb with the NEW option is generated for IMAGE and Eloquence master files so that the master dataset data record is added before any associated IMAGE or Eloquence detail dataset data records.
- The delete constraints section of the generated UPDATE procedure consists of PUT verbs with the DELETED option. The DELETED option states that the record must be marked for deletion before the PUT verb is executed. The PUT verb with the DELETED option is generated for IMAGE and Eloquence detail datasets so that the records in the detail dataset are deleted before the associated master dataset data record. This section also contains PUT verbs for DELETE files which are IMAGE or Eloquence detail datasets.
- The consistency section of the generated UPDATE procedure ensures that all files and tables are updated. This section contains all PUT verbs generated without the NEW or DELETED options. The order that they are generated in depends on the **update=bottomup|topdown|fkc_put_order** program parameter setting. The default is **bottomup**, as described below. For more information on the **update** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

Bottomup: Default Order of PUT Verb Generation

1. Generate SECONDARY files that occur with the DETAIL file
2. Generate DELETE files that occur with the DETAIL file
3. Generate the DETAIL file
4. Generate SECONDARY files that occur with or are related to the PRIMARY file
5. Generate DELETE files that do not occur with the DETAIL file
6. Generate the PRIMARY file
7. Generate MASTER files that occur with the PRIMARY file
8. Generate MASTER files that do not occur with the PRIMARY file
9. Generate standalone AUDIT files

Topdown: 7.10 Order of PUT Verb Generation (OpenVMS)

This is the order of PUT verb generation on OpenVMS in version 7.10 and before.

1. Generate Master files
2. Generate the Primary file
3. Generate the Detail file and Secondary files
4. Generate Delete files
5. Generate stand-alone Audit Files

fkc_put_order: Order of PUT Verb Generation for Foreign Key Constraints

The ordering of PUT verbs based on foreign key constraints is available for relational databases, except those defined with the RDB/VMS file type. For RDB/VMS databases, see "fkc_put_order for RDB/VMS Databases" (p. 359).

In order to generate the PUT verbs in the correct sequence, the primary keys referenced by foreign keys must have unique indexes created on them. Most databases will automatically create a unique index when the primary key is created, however some will not.

A parent is a relational table that is referenced by a foreign key constraint from another table. A child is a relational table with a foreign key constraint that references another table.

The rules for the order of PUT verb generation for parent and child tables on a screen are:

1. The add constraints section contains PUT verbs with the NEW option for all parents that are not DELETE-type files.
2. The delete constraints section contains PUT verbs with the DELETED option for all children that are not DELETE-type files. It also contains PUT verbs with no options for all children that are DELETE-type files.
3. The consistency section contains PUT verbs for all tables on the screen, except for the DELETE-type files handled in step 2, for which PUT verbs have already been generated.

The PUT verb generation in steps 1 and 2 is done respecting multi-level constraints that might be in place between the parent and child tables.

fkc_put_order for RDB/VMS Databases (OpenVMS)

The foreign key constraint functionality will not work for databases defined in the dictionary with type RDB/VMS since PowerHouse is unable to determine the foreign key constraint rules. In order to bypass this limitation and generate the PUT verb sequence in the UPDATE procedure based on foreign key constraint rules, use the following technique:

1. Add an entry to the dictionary for the same database defined as type RDB.
2. Make a copy of the screen source code and change the references to the database in the screen source to the new database definition.
3. Compile the new screen against this new dictionary database definition using the **update=fkc_put_order** program parameter. Use BUILD LIST DETAIL. This will cause the PUT verb sequence in the UPDATE procedure to be generated according to the foreign key constraint rules.
4. Save the generated UPDATE procedure, copy it to the original screen and recompile. The **update=fkc_put_order** is not necessary.

Example

The following partial example illustrates the UPDATE procedure for a subscreen:

```
> SCREEN MIDDLE RECEIVING MASTER1, MASTER2
> FILE MASTER1 MASTER
> FILE MASTER2 MASTER
> FILE PRIME PRIMARY OCCURS 6
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF MASTER2.
> FILE SECONDA SECONDARY OCCURS WITH PRIME
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF PRIME.
> FILE SECONDB SECONDARY OCCURS WITH PRIME
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF PRIME.
> FILE SUBPRIME1 DELETE OCCURS WITH PRIME
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF SECONDB.
> FILE SUBPRIME2 DELETE OCCURS WITH PRIME
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF SECONDB.
> FILE PRIMEAUDIT AUDIT WITH PRIME OCCURS WITH PRIME
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF PRIME.
.
.
.
> BUILD LIST
```

All files involved in a single logical relationship should be updated together. This minimizes the time during which the files are logically inconsistent.

Using the previous screen design statements, the default UPDATE procedure constructed by QDESIGN for an indexed file is:

```
> PROCEDURE UPDATE
```

```
>   BEGIN
>     FOR PRIME
>       BEGIN
>         PUT SECONDA
>         PUT SECONDB
>         PUT SUBPRIME1
>         PUT SUBPRIME2
>         PUT PRIME
>         END
>     PUT MASTER1
>     PUT MASTER2
>     END
```

Chapter 8: QDESIGN Verbs and Control Structures

Overview

This chapter provides a detailed reference of QDESIGN verbs and control structures. QDESIGN verbs perform specific actions within procedures. QDESIGN control structures determine the processing flow within procedures. For each verb and control structure you'll find

- syntax summaries
- detailed syntax descriptions
- detailed discussions
- examples

Summary of QDESIGN Verbs and Control Structures

The following table lists QDESIGN verbs and control structures with brief descriptions of what they do.

Verb or Control Structure	Purpose
ACCEPT	Prompts for, edits, and displays a value in a field.
BEGIN ... END	Marks the beginning and end of a compound statement.
BLOCK TRANSFER	Controls user prompting for groups of fields.
BREAK	Stops data record retrieval.
[SQL] CALL	Calls a stored procedure or function from the specified database.
CLEAR	Clears terminal lines.
[SQL] CLOSE	Closes a file or cursor.
COMMIT	Commits one or more transactions.
DELETE	Marks a data record for deletion.
[SQL] DELETE	Deletes rows from a table.
DISABLE	Prevents the use of a procedure.
DISPLAY	Displays a value in a field.
DO BLOB	Executes an external program to handle the contents of the BLOB.
DO EXTERNAL	Executes an external program.
DO INTERNAL	Executes an internal procedure.

Verb or Control Structure	Purpose
EDIT	Edits a value in a field.
ERROR	Stops processing and issues an error message.
[SQL] FETCH	Retrieves the next row of data for the specified cursor.
FOR	Establishes a control structure that repeats a procedural statement.
GET	Retrieves a data record.
IF	Establishes a conditional statement.
INFORMATION	Issues an informational message.
[SQL] INSERT	Adds new rows to a table.
LET	Sets the value of an item equal to an expression.
LOCK	Locks a file, an IMAGE database or dataset, an ALLBASE/SQL table or an Oracle table.
MEMOLOG (**MPE/iX**)	Writes a message to an IMAGE log file.
NULL	Performs a null action.
[SQL] OPEN	Opens a cursor and gets the result rows ready to be accessed with subsequent SQL FETCH statements.
PERFORM APPEND	Executes the APPEND procedure.
PROMPT	Prompts for and displays a value in a field.
PUSH	Places a command or list of command options at the top of the command stack.
PUT	Updates the data record.
REFRESH	Clears and rewrites (refreshes) an area of terminal memory.
REQUEST	Prompts for a value required for record retrieval.
RETURN	Exits from a screen.
ROLLBACK	Restores the data affected by an update to the state that it was in before the transaction was started.
RUN COMMAND	Executes an operating system command.
RUN REPORT	Executes a QUIZ report.
RUN RUN	Executes a QTP run.
RUN SCREEN	Invokes a lower-level screen.
RUN THREAD	Specifies a screen thread.
SELECT	Prompts for a selection value in Select mode.
SEVERE	Aborts processing and issues a severe message.
START	Starts a transaction.

Verb or Control Structure	Purpose
STARTLOG (**MPE/iX**)	Marks the start of a transaction set in an IMAGE log file.
STOPLOG (**MPE/iX**)	Marks the end of a transaction set in an IMAGE log file.
UNLOCK	Unlocks a file.
[SQL] UPDATE	Updates rows in a table.
WARNING	Issues a warning message.
WHILE	Executes the next procedural statement as long as the condition is true.
WHILE RETRIEVING	Retrieves and processes data records in a loop.

ACCEPT

Prompts for, edits, and displays a value in a field.

Syntax

ACCEPT field

field

Names the field where the user is prompted for input.

Discussion

The ACCEPT verb is the most powerful of the field processing verbs. When the ACCEPT verb is encountered, QUICK

1. Prompts the user to enter a value at the specified field.
2. Checks and edits the entered value.
3. Stores the value from Step 2 in a record buffer or a temporary item buffer, and performs balancing and other secondary processing as required.
4. Displays the value back in the appropriate field in the correct format.

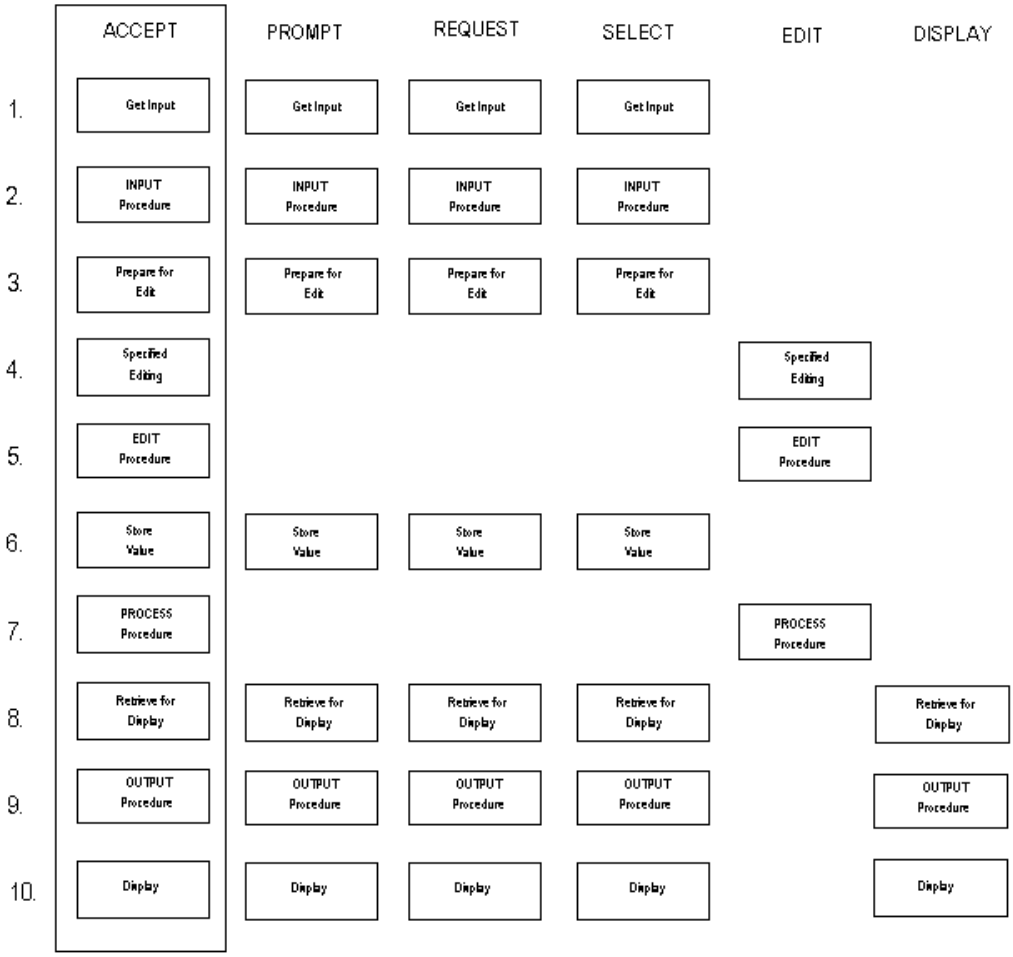
The ACCEPT verb is used in the ENTRY and APPEND procedures to accept new data. The ACCEPT verb is also used in both the MODIFY procedure and numbered DESIGNER procedures for altering existing values.

Since there are so many activities associated with accepting and editing a data item, the process is broken into a series of steps. Each step provides user exits in the form of procedures for you to intervene at preset points in the process.

Note: For information about verb and procedure compatibility, see (p. 239).

Processes Initiated by the ACCEPT Verb

The following figure illustrates the processes that are performed when the ACCEPT verb is executed, and contrasts these steps with the steps that are performed by other field processing verbs:



The steps the ACCEPT verb follows depend on what QUICK screen users enter. Exceptions to the steps are caused by

- null entries
- user-entered Backup commands
- user-entered Duplicate commands
- user-entered Help or Refresh commands
- the Skip commands ("/", "/n", "//")

The Skip commands simulate null entries.

For information about how QUICK handles these exceptions, see (p. 367).

The steps performed in processing the ACCEPT verb are discussed below. In addition, these steps are contrasted with similar steps that are performed by other field processing procedures. Not all steps are performed in every case. Some steps may be bypassed, depending on the specific verb and the nature of the entry.

1. Get input from the user.

If the ACCEPT verb doesn't occur within a BLOCK TRANSFER control structure, QUICK prompts for a value in the field associated with the ACCEPT verb. If the field already contains an entry, QUICK suppresses formatting on the entry, highlights the field, and performs a terminal read once the user enters a value. Processing is performed based on the value read.

If the field occurs within a BLOCK TRANSFER control structure, QUICK processes the ACCEPT verb in two separate passes.

During the first pass, QUICK determines which of the fields associated with verbs in the BLOCK TRANSFER (including ACCEPT verbs) should be enabled for input. If these fields already contain entries, QUICK suppresses formatting on them and enables them for input. The user can then enter or modify values in the enabled fields, and presses [Return] only when all values for the enabled fields in the BLOCK TRANSFER are entered.

During the second pass, QUICK performs a terminal read. This occurs only after the QUICK screen user presses [Return]. QUICK processes the ACCEPT verbs associated with each field for which a value was entered.

If the field type is CHARACTER, the UPSHIFT and DOWNSHIFT options are applied if they are specified. If the user presses a data field function key, that key is trapped in this step. Further processing depends on the actions triggered by the trapped function key.

2. Perform the INPUT procedure.

   The optional INPUT procedure related to the field is performed. In this procedure, you can alter the contents and size of the FIELDTEXT predefined item before editing takes place.

3. Prepare for edit.

   If the FIELDTEXT predefined item has a size of zero, the PROMPTOK predefined condition is set to false for the field; otherwise, it is set to true.

   If the PROMPTOK predefined condition is false and none of the REQUIRED, DUPLICATE, or DEFAULT options are specified, the rest of Step 3 is skipped and processing continues with Step 8.

   The value of the FIELDTEXT predefined item is stored in the duplicate buffer for this field.

   If the field type is NUMERIC, the field is checked for numbers. The result is scaled by the input scale.

   If the field type is DATE, the field is checked for a proper date as follows:

   - The month portion must be in either alphabetical or numeric form.
   - The year portion must be in either two or four-digit form, and with the year, month, and day components in the proper order as determined by the format.
   - The entered value is converted to a six-digit number in the format YYMMDD, or an eight-digit number in the format YYYYMMDD.

   If the field type or item type is either numeric or date, the numeric value is placed in the FIELDVALUE predefined item.

4. Perform specified editing.

   Data dictionary or FIELD statement editing is performed using

   - the FIELDTEXT predefined item for character items
   - the FIELDVALUE predefined item for numeric or date items

   This editing includes VALUE options, pattern matching, and lookups, in that order.

   Any reference to the associated item by name in the specified edits is trapped, and the FIELDTEXT predefined item (for character items) or the FIELDVALUE predefined item (for numeric or date items) is used instead. For pattern matching of numeric or date items, the field value is converted to a character string. The existing value of the item in the record buffer may be referenced by using the OLDVALUE function.

   For more information about FIELD statement editing, see (p. 105). See also the ELEMENT statement in Chapter 2, "PDL Statements", in the *PDL and Utilities Reference book*.

5. Perform the EDIT procedure.

   The optional EDIT procedure related to the field is performed. In this procedure, you can specify additional procedural editing that may not be possible using only FIELD statement options. For example, you can validate combinations of segments in multi-segment indexes. Within this procedure, any reference to the associated item within an expression is trapped and the FIELDTEXT or FIELDVALUE predefined item is used instead. The existing value of the item can be referenced with the OLDVALUE function. The existing value should not be changed; this is done automatically in Step 6. Restrict activities in the EDIT procedure to validating the entered value.

6. Store the value.

The accepted value in the FIELDVALUE or FIELDTEXT predefined item is stored in the item or segment and in the associated duplicate buffer; any balancing sums are also performed. If the value in a record item changes, or if the value is different from the INITIAL value for the field, QUICK sets the record status to CHANGED (the ALTEREDRECORD predefined condition is set to true). QUICK doesn't set the record status to CHANGED if the value for the field is the same as the INITIAL value.

7. Perform the PROCESS procedure.

   The optional PROCESS procedure related to the field is performed. At this point, the data has been accepted and is now stored in the associated item. You can use this procedure to specify spin-off processing and data manipulation that is performed when a new or changed value is placed in the item.

8. Retrieve the value for display.

   The value of the item is retrieved and placed in the FIELDTEXT predefined item.

   A numeric item is scaled by the output scale, placed in the FIELDTEXT predefined item (right-justified in an area matching the field size) with the minus sign (where applicable) just to the left of the left-most digit, and blank filled.

   A date item is converted to a six-digit number with the format YYMMDD or an eight-digit number with the format YYYYMMDD and placed in the FIELDTEXT predefined item.

   A character item is moved unchanged to the FIELDTEXT predefined item.

9. Perform the OUTPUT procedure.

   The optional OUTPUT procedure related to the field is performed. At this point, you may specify reformatting of the FIELDTEXT predefined item prior to the final formatting options.

10. Display the value.

    If the field type is DATE, the content of the FIELDTEXT predefined item is converted to a display string under control of the date format, separator, and null separator options.

    If the field type is NUMERIC, the content of the FIELDTEXT predefined item is converted to a display string under control of the numeric picture and format options.

    If the field type is CHARACTER, the FIELDTEXT predefined item is converted to a display string under control of the character picture.

    The display string is displayed back to the QUICK screen user.

How the ACCEPT Verb Responds to Null Entries

When the FIELDTEXT predefined item has a zero length at the end of Step 3, the PROMPTOK predefined condition is set to false for that field, and further processing is dependent on the field options DEFAULT, REQUIRED, and DUPLICATE. The DUPLICATE field option can be specified either alone or in combination with a REQUIRED or a DEFAULT expression.

If the DUPLICATE option is specified for the field, QUICK moves the previously entered value to the FIELDTEXT predefined item. QUICK ignores the option if there is no previous entry to duplicate. Processing continues as if the user had entered the value. The PROMPTOK predefined condition is set to true for the field if a duplicate value is used.

If a DEFAULT expression is specified (and no value has yet been entered for the field), the expression is evaluated and moved to the associated item. Any balancing sums are performed at this stage, and QUICK skips Steps 3 through 5 (editing). If the DEFAULT and REQUIRED options are used on the same statement, the DEFAULT option is ignored. The PROMPTOK predefined condition remains false for the field and the value isn't put in the duplicate buffer.

If the REQUIRED option is specified and no value has yet been entered for the field, an error is signaled.

How the ACCEPT Verb Responds to a User-Entered Backup Command

If a user enters a Backup command while the cursor is positioned in a field, the cursor moves backwards one field at a time. The Backup command takes the user back to the previous ACCEPT or PROMPT verb, or the previous BLOCK TRANSFER control structure.

As the cursor moves back, the value in the field from which the backup was signaled is redisplayed using Steps 8 through 10. QUICK then backs up to the previous ACCEPT or PROMPT verb, or the previous BLOCK TRANSFER control structure.

Limit: The Backup command is not available for BLOCK TRANSFERs.

How the ACCEPT Verb Responds to a User-Entered Duplicate Command

Entering a Duplicate command places the contents of the duplicate buffer for the field to be placed in the field on the terminal. The INPUT procedure is performed only if the duplicated value is modified; otherwise processing continues with Step 3 as if the value had been entered by the user. If no duplicate value exists, QUICK issues a warning message. The PROMPTOK predefined condition is set to true for the field if a duplicate value is used.

How the ACCEPT Verb Responds to a User-Entered Skip All Command

Entering the Skip All command isn't the same as entering a Backout command. A Skip All command simulates null data entries into the remaining fields. A Backout command returns the QUICK screen user to the Action field and backs out all changes the user made.

If a field in a NOPANEL screen contains a partial entry, a Skip All command causes QUICK to process the partial entry before returning to the Action field.

On screens with repeating PRIMARY or repeating DETAIL record-structures, the Skip All command simulates null entries only to complete the entry sequence for the current occurrence.

In the following example, the Skip All command causes QUICK to simulate null entries only if a value is entered for the BRANCH field for the current occurrence. Otherwise, since the occurrence doesn't actually exist, QUICK terminates the entry sequence with no field processing. In contrast, a Skip All command entered for a BRANCHNAME field simulates null entries for the BRANCHNAME and BRANCHMANAGER fields and then terminates the entry sequence.

```
> SCREEN BRANCHES
> FILE BRANCHES OCCURS 5
> CLUSTER OCCURS WITH BRANCHES
>   FIELD BRANCH REQUIRED NOCHANGE &
>         LOOKUP NOTON BRANCHES
>   FIELD BRANCHNAME
>   FIELD BRANCHMANAGER
> CLUSTER
> BUILD
```

With repeating DETAIL record-structures, the Skip All command finishes only the current occurrence. If // is entered in any field of the PRIMARY record, null entries are simulated for the remaining PRIMARY fields. All occurrences of the DETAIL fields are skipped. If // is entered in the first field of any occurrence of a DETAIL record, the current occurrence (and remaining occurrences) are skipped. If // is entered in any DETAIL field except the first field of an occurrence, the current occurrence is completed. Null entries are simulated for remaining fields in the current occurrence. Remaining occurrences are skipped.

If a screen has a PRIMARY record-structure and a SECONDARY record-structure, and a field of the SECONDARY record-structure has a REQUIRED option, entering // in the Entry sequence stops the cursor at the required field. This happens even if the record status of the SECONDARY data record is New and Unchanged; the SECONDARY record-structure is considered an extension of the PRIMARY record-structure. Null entries in required PRIMARY record-structure fields are not acceptable.

To separate the SECONDARY file field from the PRIMARY record-structure, place a CLUSTER statement before the first SECONDARY file field (as in CLUSTER OCCURS WITH file), even though the SECONDARY record-structure doesn't repeat. If a Skip Cluster command is entered before data is entered into any of the SECONDARY file fields, and if the record status of the SECONDARY file has not been changed procedurally, all of the SECONDARY file fields are bypassed.

While QUICK simulates null entries for verbs requiring user input (ACCEPT, PROMPT, and REQUEST), most other verbs are executed normally. However, the DO EXTERNAL and RUN SCREEN verbs are ignored.

Example

The following screen design demonstrates how to use ACCEPT verbs.

```
> SCREEN SALES
>
> FILE CUSTOMERS PRIMARY
> FILE INVIOCES SECONDARY
Item CUSTNO initialized (fixed) to CUSTNO OF CUSTOMERS.
Item DISCOUNT initialized (fixed) to DISCOUNT OF CUSTOMERS.
> FIELD CUSTNO OF CUSTOMERS REQUIRED NOCHANGE &
>   LOOKUP NOTON CUSTOMERS
> FIELD CUSTNAME OF CUSTOMERS
> FIELD INVOICENO OF INVOICES REQUIRED NOCHANGE &
>   LOOKUP NOTON INVOICES
> FIELD INVOICEDATE OF INVOICES
> FIELD INVOICETOTAL OF INVOICES
>
> PROCEDURE ENTRY
>   BEGIN
>
>     ACCEPT CUSTNO OF CUSTOMERS
>     ACCEPT CUSTNAME OF CUSTOMERS
>     ACCEPT INVOICENO OF INVOICES
>     ACCEPT INVOICEDATE OF INVOICES
>     ACCEPT INVOICETOTAL OF INVOICES
>     END
>
> BUILD LIST
INVOICES accessed via CUSTNO
```

QDESIGN automatically generates ACCEPT verbs in the default ENTRY, APPEND, and MODIFY procedures.

BEGIN...END

Marks the beginning and end of a compound statement.

Syntax

BEGIN
    compound-statement
END

BEGIN

Marks the beginning of a compound statement. The BEGIN keyword must appear alone on a line except when it is part of an IF control structure. In this case, it may be preceded by the keyword THEN.

compound-statement

One or more verbs and/or control structures.

You can nest compound statements to any level. Each compound statement starts with a BEGIN keyword and concludes with an END keyword.

END

Marks the end of a compound statement. The END keyword must appear alone on a line.

Discussion

There must be one END keyword for each BEGIN keyword.

Where to Use BEGIN and END

If a procedure contains more than one verb and/or control structure, the BEGIN and END structures are required. The BEGIN and END control structures are not necessary in procedures that contain only one statement. The BEGIN and END control structures are not included in any procedure where the DISABLE verb is used to prevent execution of an activity, such as deletion.

Combining Compound Statements with Other Control Structures

Compound statements can be combined to work within other control structures. For example, you can include several verbs or control structures in THEN and ELSE clauses of the IF control structure. Similarly, you can include several verbs or control structures in a FOR loop by using the BEGIN and END control structure.

Example

The following example shows how to use the BEGIN and END keywords to group individual verbs and control structures into compound statements. In this example:

- The first BEGIN works together with the final END to block the entire ENTRY procedure.
- The second BEGIN works together with the first END to block the PERFORM APPEND verb.

```
> PROCEDURE ENTRY
>   BEGIN
>     ACCEPT INVOICENUMBER OF INVOICES
>     ACCEPT ACCOUNTNUMBER OF INVOICES
>     ACCEPT INVOICEDATE OF INVOICES
>     FOR INVOICESTATS
>       BEGIN
>         PERFORM APPEND
>       END
>   END
```

Using Compound Statements with Other Control Structures

The following example demonstrates how to incorporate a compound statement into an IF control structure. In this example, the two procedural statements RUN and LET are executed only if the specified condition is true.

```
> IF FIELDTEXT = "???"
>   THEN BEGIN
>     RUN SCREEN BRANCHK PASSING TEMPBRANCH MODE F
>     LET FIELDTEXT = TEMPBRANCH
>     END
```

BLOCK TRANSFER

Controls user prompting for groups of fields.

Syntax

BLOCK TRANSFER [SEQUENCED] control structure|verb

SEQUENCED

Prevents the QUICK screen user from skipping entries in response to prompts within the block. When the SEQUENCED option is specified, the QUICK screen user can't enter a data value into a field within the BLOCK TRANSFER until values have been entered for all preceding fields in the BLOCK TRANSFER.

In general, the BLOCK TRANSFER control structure can be used anywhere that the ACCEPT verb is valid. Blocks that contain more than one verb are created by entering the BEGIN and END control structures following the BLOCK TRANSFER control structure.

control structure|verb

The BLOCK TRANSFER control structure must always be followed by one of the following procedural statements:

Control structures	Verbs
BEGIN ... END	ACCEPT
FOR	DISPLAY
FOR MISSING	EDIT, PERFORM APPEND, PROMPT, REQUEST, SELECT

Discussion

The BLOCK TRANSFER control structure, together with the FOR control structure, allows PowerHouse applications to accept and process one or more data fields as a block, rather than individually. This behavior is called Panel input.

If you include the PANEL option on the SET statement or the SCREEN statement, QDESIGN generates the APPEND, ENTRY, MODIFY, PATH, and SELECT procedures, and includes BLOCK TRANSFER control structures in each, as in

```
> PROCEDURE MODIFY
>   BEGIN
>     BLOCK TRANSFER
>       BEGIN
>         ACCEPT FIELD1
>         ACCEPT FIELD2
>         ACCEPT FIELD3
>       END
>   END
.
.
.
```

where FIELD1, FIELD2, and FIELD3 are grouped into a panel.

BLOCK EACH and BLOCK ALL options on CLUSTER statements affect the way the BLOCK TRANSFER control structures are generated in the APPEND and ENTRY procedures.

How the BLOCK TRANSFER Control Structure Works

The processing associated with a BLOCK TRANSFER control structure is divided into three phases of execution. In these phases, QUICK

1. Determines which fields in the block are to be enabled for user input.

Each ACCEPT, REQUEST, PROMPT, or SELECT verb is scanned to determine whether or not the associated field is to be enabled for input. All fields that aren't marked as both NOCORRECT and NOCHANGE are enabled for entry, including fields in repeating primary, secondary, or detail record-structures.

In all enabled fields, formatting is suppressed on existing data values to facilitate modifications.

2. Accepts input from the user in all fields that were enabled for input in Step 1. Prompting occurs in enabled fields, according to field id sequence. The user can move from field to field, entering and changing data entries as desired in enabled fields. Step 2 ends when the user presses [Enter] and transmits the block to QUICK for processing.
3. Executes the verbs and control structures in the block in the order in which they appear in the BLOCK TRANSFER control structure. The normal field processing procedures (INPUT, EDIT, PROCESS, OUTPUT) are executed for each field in the block in the order in which they occur.

How QUICK Determines Which Fields to Enable for Input in a Block

QUICK examines the fields that are within the BLOCK TRANSFER control structure. Usually, QUICK enables all fields that are referenced by the ACCEPT, PROMPT, REQUEST, or SELECT verbs.

Fields are not enabled for input even when they are referenced by one of the ACCEPT, PROMPT, REQUEST, or SELECT verbs if

- the referenced field has the NOCHANGE option specified and the record status is Old
- the referenced field has the NOCORRECT option specified, the record status is New, and the field has previously been processed by QUICK
- the item associated with the referenced field belongs to a record marked for deletion
- the referenced field has the NOSELECT option specified. This applies only to the SELECT verb.

If a FOR control structure appears within a BLOCK TRANSFER, then all fields in all occurrences implied by the FOR control structure are enabled for entry, subject to the constraints already discussed. In addition, the FOR control structure must contain only the verbs and/or control structures that are allowed in the BLOCK TRANSFER.

Similarly, when a PERFORM APPEND verb appears within a BLOCK TRANSFER, all fields in the APPEND procedure are examined to determine whether or not they should be enabled for input. The APPEND procedure can contain a BLOCK TRANSFER control structure. However, all BLOCK TRANSFER control structures in the APPEND procedure are ignored if the PERFORM APPEND verb is called from within a BLOCK TRANSFER. Only the verbs and control structures that are valid in a BLOCK TRANSFER can be used in the APPEND procedure (see (p. 372)).

Once a field is enabled for entry, any existing data in the field is displayed unformatted to facilitate data entry for the QUICK screen user; no extraneous characters (such as date separators and currency symbols) are required. Data is always assumed to exist in a field when

- the record buffer is marked as either New and Changed or Old and Changed
- the field is associated with a temporary item

Error Handling in a BLOCK TRANSFER Control Structure

Once the user transmits a block of field values to QUICK, the control structures and verbs in the BLOCK TRANSFER are processed in the order in which they occur.

As verbs are processed, QUICK handles any data entry errors. As soon as QUICK detects an invalid entry in a field (due to the failure of a field EDIT procedure, for example), the erroneous field is marked internally and processing continues for the rest of the field processing verbs in the BLOCK TRANSFER control structure. However, for all of the remaining fields in the block, only the Input and Edit phases are executed; the Process and Output phases are ignored. In addition, QUICK clears any fields for which the DISPLAY verb was specified.

Upon completion of the verbs in the BLOCK TRANSFER, QUICK positions the cursor on the first field in error, highlights all fields in error, and displays the error message associated with the first highlighted field. QUICK then runs the entire BLOCK TRANSFER again. In cases where QUICK processes some fields before the error occurs, not all of the fields that were enabled for the first execution of the BLOCK TRANSFER are enabled when the BLOCK TRANSFER is run again. For example, a field for which NOCORRECT is specified isn't enabled on subsequent executions of a BLOCK TRANSFER if QUICK accepted a valid value for that field before any errors occurred.

For information about the Input, Edit, Process, and Output phases in QUICK, see (p. 364), (p. 452), and (p. 481).

It's important to note that when an error occurs within a BLOCK TRANSFER, QUICK always reruns the entire block transfer.

For screens that are not designated as PANEL screens, no BLOCK TRANSFER control structures are generated. The result is that QUICK treats each field as a single block of information.

Where the BLOCK TRANSFER is Generated Automatically

The BLOCK TRANSFER control structure affects procedures that use the ACCEPT, PROMPT, REQUEST, and SELECT verbs. BLOCK TRANSFERs are generated automatically for the following procedures when PANEL is specified on either the SCREEN statement or the SET statement:

APPEND	ENTRY	PATH
MODIFY	numbered DESIGNER procedures	SELECT

Nesting BLOCK TRANSFER Control Structures

Explicitly nested BLOCK TRANSFER control structures are not valid in QDESIGN. However, you can nest a BLOCK TRANSFER control structure implicitly in the ENTRY procedure by including a PERFORM APPEND verb that references an APPEND procedure containing a BLOCK TRANSFER control structure.

For examples of BLOCK TRANSFER control structures, see (p. 329) and (p. 330).

BREAK

Stops data record retrieval.

Syntax

BREAK

Discussion

The BREAK verb stops data record retrieval. The BREAK verb provides the QUICK screen user with the option of stopping the control structure loop when a certain condition is met, instead of allowing the loop to complete.

Limit: The BREAK verb is valid within the FOR, the WHILE, and the WHILE RETRIEVING control structures.

Example

In this example, an edit procedure compares the quantity on hand to the quantity required. If the quantity required is available, then the break verb stops the while retrieving control structure from processing.

```
> SCREEN STOCK
> FILE STOCK
> FILE STOCKDETAIL DESIGNER
> TEMPORARY QUANTITYREQUIRED NUMERIC*8
> TEMPORARY QUANTITYFOUND NUMERIC*8
.
.
.
> PROCEDURE EDIT QUANTITYONHAND
>    BEGIN
>    LET QUANTITYFOUND = 0
>    WHILE RETRIEVING STOCKDETAIL VIA STOCKNO &
>      USING STOCKNO OF STOCK
>      BEGIN
>       LET QUANTITYFOUND =  &
>           QUANTITYFOUND + QUANTITYONHAND
>       IF QUANTITYFOUND GE QUANTITYREQUIRED
>          THEN BREAK
>      END
> IF QUANTITYFOUND < QUANTITYREQUIRED
>   THEN ERROR = "Not enough" + STOCKNAME + "On hand."
> END
```

[SQL] CALL

Calls a stored procedure or stored function from the specified database.

```
[SQL [IN database]
[{TRANSACTION transaction_name
    [FOR {CONSISTENCY|{[CONCURRENCY]
        phase-option [,phase-option]...}}]}]]
CALL stored-procedure|stored-function
    [([ITEM] item [IN [OUT]]|[OUT]
        [,[ITEM] item [IN [OUT]]|[OUT]]...)]
    [ON ERROR CONTINUE|TERMINATE]
    [RETURNING return-parameter
```

Limit: This statement is valid only for DB2, Oracle, ODBC, or Sybase databases.

IN database

Specifies against which database the stored procedure or function is executed.

Limit: Stored procedure calls are valid for DB2, ODBC, Oracle, Oracle Rdb (declared as TYPE RDB in the dictionary), and Sybase databases.Stored function calls are valid only for Oracle databases.

TRANSACTION transaction_name [FOR {CONSISTENCY| {[CONCURRENCY] phase-option[,phase-option]...}}]

If the TRANSACTION option is not used, one of PowerHouse's default transactions is used. Default transactions are associated with every file, table or stored procedure statement. The default transactions are Query, Update and Consistency.

TRANSACTION

Specifies that the transaction is associated with the stored procedure call.

transaction_name

Any valid PowerHouse name.

FOR CONSISTENCY

Determines that a stored procedure call is associated with a particular transaction in Consistency model.

Limit: Only one transaction association can be specified.

FOR [CONCURRENCY] phase-option [,phase-option]...

Determines that a stored procedure call is associated with a particular transaction or transactions in Concurrency model.

Limit: Up to three transaction associations can be specified.

phase-option

Specifies the screen phase with which the transaction is associated.

Phase option	Description
PROCESS	The phase in which you are entering, correcting, or changing data records on the screen.
QUERY	The phase in which data is retrieved from the database.
UPDATE	The phase in which data is updated.

For more information about transactions, see the *PowerHouse and Relational Databases* book.

CALL stored-procedure|stored-function

The name of a stored procedure or stored function in the database.

The syntax for a procedure name varies with the RDBMS. For information on a specific database system, see "Stored Procedures" in the *PowerHouse and Relational Databases* book.

([ITEM] item [IN [OUT]]|[OUT] [,[ITEM] item [IN [OUT]]|[OUT]]...)

Items which are passed to the stored procedure or Oracle stored function, or received from the stored procedure. Input parameters can be temporary, defined or record items. Output parameters can be temporary or record items.

BLOB items may also be used for both input and output parameters when calling an Oracle stored procedure or stored function.

IN

Specifies that the item is an input parameter.

IN OUT

Specifies that the item is both an input and output parameter. The changed values of the input/output parameters are available to PowerHouse when stored procedure execution is complete.

OUT

Specifies that the item is an output parameter. The changed values of the output parameters are available to PowerHouse when stored procedure execution is complete.

Default: IN

RETURNING return-parameter

The return-parameter must be defined as a temporary or record item.

For Sybase, identifies the item that contains the return status from a stored procedure upon completion of the Sybase stored procedure.

For Oracle, identifies the item that contains the value returned by a stored function upon completion of the Oracle stored function.

Limit: Valid for Oracle stored functions but not valid for Oracle stored procedures. For Sybase, the return-parameter must be defined as a 32-bit (4-byte) integer.

ON ERROR CONTINUE|TERMINATE

Specifies the action to be taken if the SQL statement fails. If TERMINATE is in effect, the SQL error causes QUICK to process the error as it would for an ERROR verb. If CONTINUE is specified, the SQL error is ignored and the processing continues as if the error had not occurred.

Default: TERMINATE

Discussion

Using the CALL statement, the developer can call local or remote stored procedures and pass input and output parameters and receive execution status. The stored procedures that fall into this category are ones that process a row or set and return output parameters, status or values to the calling application. Stored procedures that return result sets must be called as part of the DECLARE CURSOR statement.

CLEAR

Clears terminal lines.

Syntax

CLEAR ALL|SCREEN|[LINES] n [TO m]

ALL

Clears all terminal lines.

SCREEN

Clears the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears the area between and including lines n to m, numbering from the first terminal line (n). LINE n by itself clears line n only.

Discussion

The CLEAR verb only works if the **restore=lines** program parameter is used.

The CLEAR verb instructs QUICK to blank out part or all of the terminal lines. The options allow the designer to restrict the portion that is to be blanked out. One option must be given. The CLEAR verb does not automatically cause the area cleared to be refreshed when QUICK performs the next input operation. The effects of the CLEAR verb are not apparent until QUICK is ready to prompt the user, and may be generated by other CLEAR or REFRESH verbs or options.

Note: For information about verb and procedure compatibility, see (p. 239).

[SQL] CLOSE

Closes a file or cursor.

Syntax

[SQL] CLOSE record-structure|cursor-name

SQL

The SQL keyword appears in all generated procedures when a CLOSE verb references a cursor. It is for documentation only and does not affect the operation of the CLOSE.

record-structure

Names the record-structure in the file to be closed. If the file was opened on a higher-level screen, the CLOSE verb on the lower-level screen is ignored.

cursor-name

The name of a cursor declared on a CURSOR statement.

Discussion

The CLOSE verb immediately closes the file that contains the named record-structure. If the file was opened on a higher-level screen, the CLOSE verb on the lower-level screen is ignored.

Closing Relational Files

We recommend that you do not use the CLOSE verb (distinct from the SQL CLOSE) or the CLOSE option of the FILE statement on relational tables. The results are unpredictable if there are any uncommitted transactions related to the table or database when the CLOSE is performed.

When a CLOSE verb or option is used, QUICK will immediately commit all transactions against that database (not just the ones associated with the table), and logically detach from the database. If errors are encountered, a rollback is attempted, and rollback pending does not apply.

This behavior can affect performance, since the attach must be re-established before work can continue against that database. It may also affect data integrity, since unrelated transactions may be committed as a result of the CLOSE. In addition, if there are other physical databases involved in the same PowerHouse transaction, committing the transactions against only one database may result in inconsistent data.

If your intent is to commit one or more transactions, we suggest you use the COMMIT verb. For more information, see the COMMIT and ROLLBACK verbs on (p. 380) and (p. 465), respectively. See also information about the transaction control options available on the SCREEN, FILE, FIELD, and TRANSACTION statements.

Closing Cursors

If there are still rows to be retrieved for the cursor, then an implicit cancel is executed for the cursor.

A cursor is closed automatically when you exit the screen on which the cursor is declared or when another OPEN is executed for the same cursor.

COMMIT

Commits one or more transactions.

Syntax

**COMMIT [TRANSACTION] [transaction_name
[,transaction_name]...]**

TRANSACTION

An optional keyword for documentation purposes only.

transaction_name

Names the transaction with which the COMMIT verb is associated.

Discussion

If the COMMIT verb with no options is executed, as in:

```
> COMMIT
```

all locally active transactions are committed. If several transaction names are specified on a single COMMIT verb, as follows:

```
> COMMIT A,B,C
```

PowerHouse attempts to commit those transactions together as a unit using a two-phase commit protocol. Support for this depends on the support provided by the underlying database system.

In contrast to QUICK's automatic commit processing, the COMMIT verb with a list of transactions issues a commit to all transactions listed regardless of whether or not they are locally active. Also, no transactions other than those listed by the COMMIT verb are affected.

A COMMIT on an inactive transaction is ignored.

DELETE

Marks a data record for deletion.

Syntax

DELETE record-structure

record-structure

Names the record-structure containing the data record to be marked for deletion. The record-structure must be declared in a FILE statement on the screen.

Discussion

The DELETE verb marks the current data record of the named record-structure for deletion. The data record is actually deleted by PUT verbs, which are normally found in the UPDATE procedure. Once a data record is marked for deletion, the data from that data record is no longer accessible.

The DELETE verb reverses the sums and counts that involve the data record marked for deletion (unless it is in a DELETE file). For example, if an item of a file is summed into another item, and the data record is marked for deletion by the DELETE verb, the sum is reversed by the DELETE verb.

Note: For information about verb and procedure compatibility, see (p. 239).

Example

The delete verb is most often used in delete and detail delete procedures. For example,

```
> PROCEDURE DELETE
>   BEGIN
>     DELETE EMPLOYEES
>     END
```

For additional examples of how to use the DELETE verb, see (p. 300).

[SQL] DELETE

Deletes rows from a table.

Syntax

```
[SQL [IN database]
[TRANSACTION transaction_name
    [FOR CONSISTENCY|{[CONCURRENCY]
        phase-option[,phase-option]...}]]]
DELETE FROM tablespec
[WHERE sql-condition|DBKEY=:expression]
```

IN database

Specifies the name PowerHouse uses to attach to the database. This is the name used to declare the database in PDL. For more information, see the *PowerHouse and Relational Databases* book.

TRANSACTION transaction_name [FOR {CONSISTENCY| {[CONCURRENCY] phase-option[,phase-option]...}]...

Defines transactions used for relational data structures and SQL DML verbs.

transaction_name

Any valid PowerHouse name.

FOR CONSISTENCY

Determines that a relational data structure is associated with a particular transaction in Consistency model.

Limit: Only one transaction association can be specified.

FOR [CONCURRENCY] phase-option [,phase-option]...

Determines that the relational data structure is associated with a particular transaction or transactions in Concurrency model.

Limit: Up to three transaction associations can be specified.

phase-option

Specifies the screen phase with which the transaction is associated.

Phase option	Description
PROCESS	The phase in which you are entering, correcting, or changing data records on the screen.
QUERY	The phase in which data is retrieved from the database.
UPDATE	The phase in which data is updated.

DELETE FROM tablespec

The name of a table in a relational database from which rows are to be removed. The syntax for tablespec is:

[[server-name.]database-name.][owner-name.]table-name

If server-name is included in a Sybase tablespec, double quotes are required for the server-name and database-name. For example,

```
"dbsvr01.accnt".manager.billings_tbl
```

For Oracle, the syntax is:

[owner-name.]table-name[@database-linkname]

If the database-linkname is included, it is treated as part of the table-name, and double quotes are required. For example,

```
manager."billings_tbl@dblnk01"
```

Oracle synonyms may be used for table-names. For more information about how PowerHouse uses Oracle synonyms, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

WHERE sql-condition|DBKEY = :expression

The sql condition is a condition which is limited to use within Cognos SQL syntax. For more information about SQL conditions, Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book, or refer to an SQL reference manual. This option provides a way to determine which rows will be deleted. Without it, all rows in the table are deleted. DBKEY is available only if the underlying database supports it.

Limit: DBKEY cannot be used with Sybase.

Discussion

The DELETE verb acts directly on a table or view in the database and is never generated by PowerHouse.

DISABLE

Prevents the use of a procedure.

Syntax

DISABLE

Discussion

The DISABLE control structure prevents the execution of the activity controlled by the procedure in which the DISABLE control structure appears. If the QUICK screen user enters an activity corresponding to a disabled procedure, QUICK issues an error message.

The DISABLE control structure isn't required for disabling optional procedures; this can be done by simply excluding the procedure from the screen. Similarly, you don't have to specify a DISABLE control structure to disable procedures that can be controlled by the ACTIVITIES option of the SCREEN statement.

The DISABLE control structure must be the only statement in the procedure in which it appears.

Procedures that are Automatically Disabled

By default, the UPDATE and DELETE procedures generated by QDESIGN for menu screens contain only the DISABLE control structure. In addition, procedures such as PATH, FIND, ENTRY, and DELETE are disabled by the omission of the corresponding ACTIVITIES option of the SCREEN statement.

Example

The following example illustrates how the DISABLE verb is used to remove a standard procedure from QUICK's processing cycle. In the following example, the DELETE procedure is disabled with the DISABLE verb.

```
> SCREEN INVINFO &
>   ACTIVITIES ENTRY, FIND, CHANGE
>
> FILE INVOICES PRIMARY
> FILE INVOICESTATS DETAIL OCCURS 10
ITEM INVOICENO initialized (fixed) to INVOICENO OF
INVOICES.
>
> FIELD INVOICENO OF INVOICES REQUIRED NOCHANGE &
>    LOOKUP NOTON INVOICES
> ALIGN (1,4,21) (,,45)
> FIELD COMPANYNUMBER OF INVOICES
> FIELD INVOICEDATE OF INVOICES
> SKIP 2
> CLUSTER OCCURS WITH INVOICESTATS FOR 2,40
>   ALIGN (1,4,13) (,,23)
>   FIELD PARTNUMBER OF INVOICESTATS
>   FIELD QUANTITY OF INVOICESTATS
> CLUSTER
> BUILD LIST DETAIL
.
.
.
> PROCEDURE DELETE
>   DISABLE
.
.
.
```

DISPLAY

Displays a value in a field.

Syntax

DISPLAY [FIELD] item [FROM [ITEM] item]
[OF [FILE] record-structure]

item

Names the field where the value is displayed.

FROM [ITEM] item

Specifies that the displayed value is retrieved from the named item rather than the item normally associated with the field.

OF [FILE] record-structure

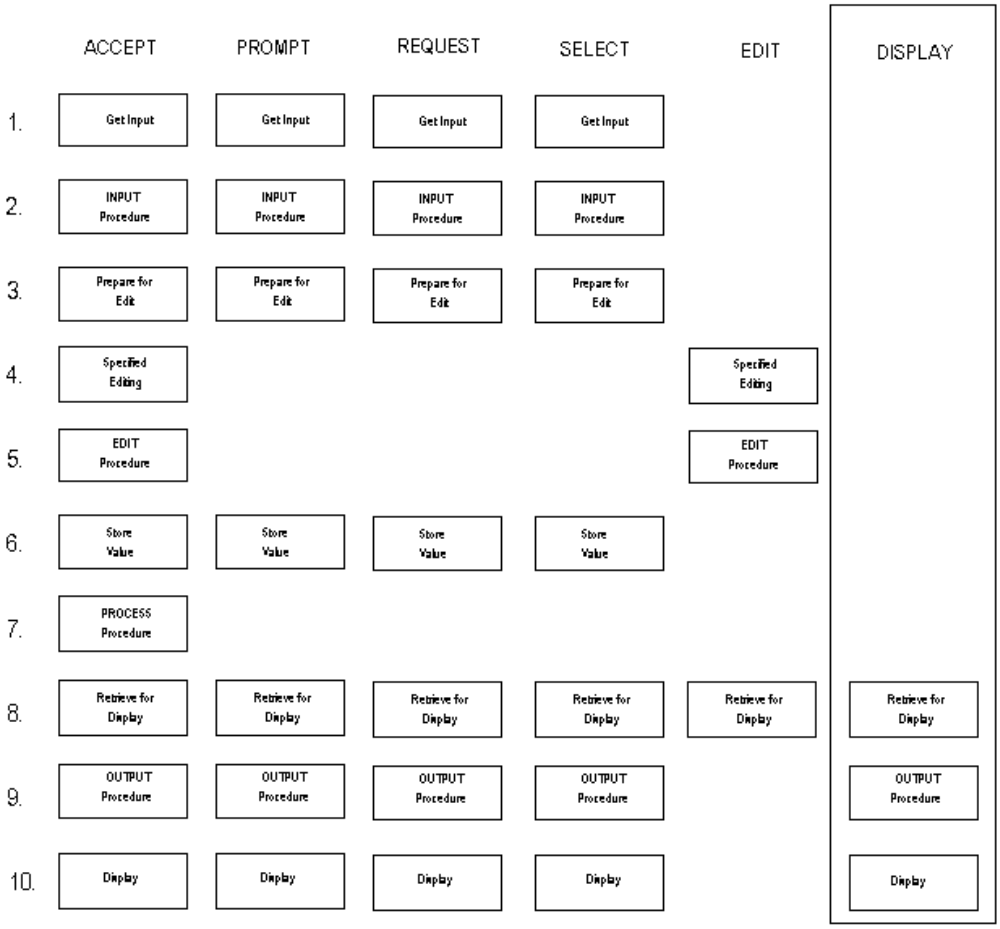
Specifies that the displayed field is retrieved from the named record structure.

Discussion

The DISPLAY verb initiates the activities that cause QUICK to display a value in the named field. The item is retrieved, formatted, and displayed.

Note: For information about verb and procedure compatibility, see (p. 239).

The following figure illustrates the steps that are initiated by the DISPLAY verb, and contrasts these steps to similar steps performed by other field processing verbs:



For details about each of the steps in the preceding diagram, see (p. 364).

The DISPLAY verb also performs its tasks

- during predisplay of initialized values
- when displaying fields after the FIND and DETAIL FIND procedures
- when redisplaying REFRESH fields on return from an invoked screen

QDESIGN automatically generates DISPLAY verbs rather than ACCEPT verbs in the default ENTRY procedure for defined items and for fields with the DISPLAY option.

Inclusion of a DISPLAY verb overrides the FIELD statement options: DISPLAY, FIXED, IF, NOENTRY, and OMIT.

Using DISPLAY Verbs in the FIND Procedure

If the DISPLAY verb is used in a FIND procedure, the display may be subsequently overwritten by the automatic display of fields at the end of the FIND procedure. The DISPLAY verb works as expected if the corresponding FIELD statement includes the OMIT ON FIND option.

Fields That Are Not in the Occurrence Window

QUICK does not move the occurrence window even if QUICK is displaying a field that is not currently in the occurrence window. The change in display will be visible once the user scrolls to that field.

If the field is in the occurrence window, then the user will see the changed field immediately.

DO BLOB

The DO BLOB verb executes an external program to handle the contents of the BLOB.

Syntax

DO BLOB blob [options]...

blob

The name of the BLOB.

Options

DO BLOB options:		
BINARY|TEXT	CLEAR	COMMAND
FILE	INPUT	NOWARN
ON ERROR	REFRESH	

BINARY|TEXT (OpenVMS, Windows)

If a BLOB is copied to or from a file using the wrong mechanism, it can be corrupted. The BINARY|TEXT option allows you to specify the contents of the BLOB and how it should be copied. The default on Windows is BINARY, and the default on OpenVMS is TEXT.

This statement copies the contents of a BLOB named file_blob in binary format to a file named prmhcp.pdf:

```
> DO BLOB file_blob BINARY command " " FILE "prmhcp.pdf"
```

This statement copies the contents of a BLOB named file_blob in text format to a file named pmhcp.txt.

```
> DO BLOB file_blob TEXT command " " FILE "pmhcp.txt"
```

This option is not necessary on UNIX or MPE/iX, which internally handle both binary and text data.

CLEAR ALL|SCREEN|[LINES] n [TO m]

Clears an area of terminal memory before the program is called. Any terminal writes from the external program appear, starting on the first line of the cleared area. Lines cleared are refreshed automatically when the screen is reactivated and QUICK is ready to prompt the user. The CLEAR option doesn't require the **restore=lines** program parameter to be used.

ALL

Clears the entire terminal memory.

SCREEN

Clears the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears the area between and including lines n to m, numbering from the first line of terminal memory. If the TO option is not used, a single line is cleared.

COMMAND string1|item1

Name of the external program to be run with the contents of the BLOB as input.

OpenVMS: If the command is not specified, PowerHouse uses the command defined in the symbol PH_BLOBEDIT. By default, PH_BLOBEDIT has the value EDIT/EDT.

UNIX, Windows: If a command is not specified, then the value of the environment variable, PH_BLOBEDIT, is used.

FILE string2|item2

A string or character item containing the name of the file where BLOB contents are loaded prior to invoking the utility specified by the command string. If no file specification is given, QUICK generates a temporary file specification. If a file is specified, it is permanent.

INPUT B|C|SAME (MPE/iX)

Puts the terminal in the specified input mode prior to executing the external program. The terminal is put back into the original mode after completion of the external program.

B

Starts the external program in Block mode if the external program can be run in Block mode.

C

Starts the external program in character mode.

SAME

Starts the external program in the same input mode as the calling screen.

Default: SAME

NOWARN (MPE/iX)

Specifies that if a command returns a non-zero status (and the ON ERROR CONTINUE option has been specified), QUICK will not issue a warning message after executing the command. However, any message issued by the command itself will be displayed.

ON ERROR CONTINUE|TERMINATE

Specifies the action to be taken if an operating system error occurs during the execution of a command. TERMINATE is the default value of the option. If TERMINATE is in effect, an operating system error causes QUICK to process the error as it would for an ERROR verb. If CONTINUE is specified, an operating system error is ignored and processing continues as if the error had not occurred.

REFRESH ALL|SCREEN|[LINES] n [TO m]

Clears and rewrites an area of the terminal memory when the screen is reactivated and QUICK is ready to prompt the user. REFRESH options are performed before, and in addition to, an automatic refresh from any CLEAR option.

The screen area options are as follows:

ALL

Clears and rewrites the entire terminal memory.

SCREEN

Clears and rewrites the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears and rewrites the area between and including lines n to m, numbering from the first line of terminal memory. If the TO option is not used, a single line is refreshed.

Discussion

The DO BLOB verb works as follows:

1. The contents of the BLOB (if any) are loaded into the file specified by the designer, or into a temporary file if no file was specified.
2. The program specified by the command string is invoked with the file as input unless a blank string is used, in which case no command is run, and the BLOB is simply copied to the file. The filename is concatenated to the end of the command prior to running the command.

   OpenVMS: If the command is not specified, PowerHouse uses the command defined in the symbol PH_BLOBEDIT. By default, PH_BLOBEDIT has the value EDIT/EDT.

   UNIX, Windows: If a command is not specified, then the value of the environment variable, PH_BLOBEDIT, is used.
3. PowerHouse then determines if the file has been modified. If it has, the contents of the BLOB are copied into an internal structure, and the file is deleted (if it was temporary).
4. If an update is performed, a new BLOB is created into which the contents from the internal structure are copied.

Examples

The DO BLOB verb can be used to interchange data between BLOBs and files. For example, the following statement copies the data from a file, descript (**MPE/iX**) or descript.txt (**OpenVMS, UNIX, Windows**) into a BLOB named description:

MPE/iX:	`DO BLOB description COMMAND "copy descript "`
OpenVMS:	`DO BLOB description COMMAND "copy descript.txt "`
UNIX:	`DO BLOB description COMMAND "cp descript.txt "`
Windows:	`DO BLOB description COMMAND "copy descript.txt "`

When used in this fashion, a temporary file is used to store the BLOB contents. The name of that temporary file is automatically substituted as part of the copy command. The space after the filename is required to preserve this automatic functionality.

Alternatively, the following statement copies the contents of a BLOB named description into a file, descript (**MPE/iX**) or descript.txt (**OpenVMS, UNIX, Windows**):

MPE/iX:	`DO BLOB description COMMAND " " FILE "descript"`
OpenVMS, UNIX, Windows:	`DO BLOB description COMMAND " " FILE "descript.txt"`

In this example, a blank string is entered after the COMMAND option so that no command is executed.

DO EXTERNAL (MPE/iX)

For DO EXTERNAL (**OpenVMS**), see (p. 398).

For DO EXTERNAL (**UNIX**), see (p. 406).

For DO EXTERNAL (**Windows**), see (p. 412).

Executes an external program.

Syntax

DO EXTERNAL [CM|INTRINSIC|NM] namelstring
[PASSING filelitem [,filelitem]...] [INPUT B|C|SAME]

or

DO EXTERNAL [CM|INTRINSIC|NM] namelstring (parm [,parm]...)

INPUT B|C|SAME

Puts the terminal in the specified input mode prior to executing the external program. The terminal is put back into the original mode after completion of the external program.

B

Puts the terminal in Block mode.

C

Puts the terminal in Character mode.

SAME

Puts the terminal in the same input mode as the calling screen.

Default: SAME

Discussion

For information about verb and procedure compatibility, see (p. 239).

Linking to External Subroutines

The DO EXTERNAL verb is provided for those occasions when you want to use localized processing with traditional programming languages. External subroutines should not be used to replace QUICK's standard functions such as finding or updating the PRIMARY file of a screen, because they interfere with QUICK. Similarly, input from or output to the terminal from localized processing is not recommended, since QUICK needs to know what is displayed on the screen at all times. Using Basic is not recommended, since its method of storing data items is not compatible with QUICK or the other languages. There are two methods used to call external subroutines from QUICK: COBOL and FORTRAN.

The DO EXTERNAL verb

The DO EXTERNAL verb supports four program call standards:

- Native Mode Cobol-compatible calling convention
- Native Mode Fortran-compatible calling convention
- Compatibility Mode Cobol-compatible calling convention
- Compatibility Mode Fortran-compatible calling convention

The Fortran-compatible calling convention uses parentheses to bracket parameters, while the Cobol-compatible calling convention uses the PASSING keyword.

QDESIGN can distinguish between Cobol and Fortran-compatible calling conventions based on the format of the parameter list. If a parameter list is not supplied, the Cobol-compatible calling convention is assumed.

Note: The following sections use only Cobol examples.

Linking Method One

The syntax of the first linking method is:

DO EXTERNAL [CM|INTRINSIC|NM] namelstring [PASSING
parm 1 [parm2]...]

where "name" is the name of the external subroutine and each "parm" is a record, record item, or temporary item that is being passed to the routine. The three keywords (CM, INTRINSIC, or NM) are used to describe the type of function to be called. CM represents Compatibility Mode and NM stands for Native Mode. The Keyword INTRINSIC indicates that the function to be carried out is a system intrinsic. The default is NM.

QUICK passes three Cobol-compatible addresses:

- common area
- common area size
- pass list

The common area is allocated and initialized to binary zeros (but not otherwise used) by QUICK. It can be used to communicate between different external subroutines or between different calls to the same subroutine.

The common area size is a one-word integer containing the size (in words) of the common area, as specified in QKGO's common area size parameter.

The pass list is the address of a data structure containing a copy of all records and items specified in the parameter list (the PASSING option). Within the data structure, the parameters are in the order specified in the parameter list and are word aligned. When a repeating item is specified in the parameter list, only the address of the first occurrence will be passed.

Note: If you use Linking Method One with CM, the QKGO parameter, Common Area Size, must be greater than zero. Otherwise, a run-time error occurs.

In the linking methods, "namelstring" is used to obtain the function name. Native mode C language allows mixed case function names. In order to prevent automatic shifting being applied to the function name, use "string."

Linking Method Two

The second method of linking to external subroutines is to omit the keyword PASSING, and to list all parameters in parentheses. The parameters listed are passed, except for the common area or common area size parameters. The syntax of the second linking method is:

DO EXTERNAL [CM|INTRINSIC|NM] namelstring
(parm 1[,parm2]...)

In this case "parm" is one of the following:

file

Passes the word address of the record buffer for the file.

BYTE (file)

Passes the byte address of the record buffer for the file.

REFERENCE (file)

Passes the word address of the record buffer for the file.

WORD (file)

Passes the word address of the record buffer for the file. This is the default address type for files.

item

Passes the word address of the item.

BYTE (item)

Passes the byte address of the item.

REFERENCE (item)

Passes the word address of the item.

WORD (item)

Passes the word address of the item. This is the default address type for items.

VALUE (item)

Passes the value of the item (can be type INTEGER, FLOAT, NUMERIC, DATE, PHDATE, JDATE, PACKED, ZONED, INTERVAL).

The three keywords CM, INTRINSIC, and NM are used to describe the type of function to be called. CM represents Compatibility Mode and NM stands for Native Mode. The keyword INTRINSIC indicates that the function to be carried out is a system intrinsic. The default is NM.

Several of the previous parameter options will produce similar results. The parameter options allow users to follow application standards. For a table of compatible item datatypes used with subroutines, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

The Native Mode Cobol and Fortran compilers automatically downshift all supplied function names except intrinsics, which are upshifted. The string option is provided for Native Mode C, which supports mixed case function names. It can also be employed to allow the use of characters, such as (') in CM SPL routines.

Cobol routines must be compiled with $CONTROL DYNAMIC. Any WORKING-STORAGE is initialized each time that the subroutine is called. To maintain values from one call to the next, use the common area.

Subroutines are compiled into object files, and then placed in an Executable Library (XL). The names of the XLs must be specified on the MPE/iX RUN statement for QUICK.

By default, QUICK allows one external subroutine and zero words in the common area. Changing QKGO parameters increases both of these settings.

If the called subroutine uses the common area, it should check the passed common area size to ensure that a sufficient size was specified in QKGO.

It is not necessary to specify a pass list, in which case the addresses for the common area, the common area size, and an empty pass list are passed.

All external subroutines within a system of screens should use the same layout for the common area, if it is passed.

Parameters passed can be records, record items, or temporary items. Not all item formats available in QUICK are Cobol-compatible. A maximum of 16 parameters can be included in any pass list. Predefined items (FIELDTEXT, FIELDVALUE, PATH) and defined items cannot be passed. If an external subroutine is called in an EDIT procedure, the address of the record item is passed. Record item names that are used as parameters are not trapped and replaced by the contents of FIELDTEXT or FIELDVALUE. To pass the value in FIELDTEXT or FIELDVALUE, use a temporary item.

Temporary areas are set up for each passed parameter. On return to QUICK, the values of all passed parameters are compared to their original values before the call. If any have changed, the original value is altered to reflect the change. If record items are changed, the record status is reset to reflect this fact. If the items in an empty record buffer (status of New, Unchanged, Undeleted) are changed, the status is New, Changed, Undeleted on return to QUICK. Subsequent processing can be affected if a subroutine is used to retrieve an existing record and place it in an empty buffer.

If the same item is passed more than once in the same call, only the last occurrence of that item is checked for value changes on return to QUICK.

If a subroutine must access a database or file, the subroutine must open and close that database or file. The considerations are the same as those for file sharing between many processes. For instance, when accessing an IMAGE database, the database can be opened and closed from the subroutine that must access it. In the case of different subroutines accessing the same database, separate open and close subroutines should be called from the main procedure (with the database name that is returned from the DBOPEN passed to the other subroutines in the common area).

Errors in an external subroutine may cause problems such as a bounds violation or memory problems in QUICK. Because an external subroutine is not controlled by QUICK, an error may corrupt QUICK's internal tables. If a subroutine involves accepting data, editing should be provided; otherwise data conversion errors can cause a process to abort.

HP system INTRINSICs with "option variable" (variable option lists) or INTRINSICs that return a value cannot be called as external subroutines from QUICK.

For Native Mode (NM) external subroutines, the data alignment will be 64-bit. All NM external subroutines must be aware of this data alignment.

If the system uses external routines, you must use the XL parameter of the RUN command to invoke QUICK.

```
: RUN QUICK.CURRENT.COGNOS;XL="MYXL1.MYGROUP.MYACCOUNT, MYXL2..."
```

MYXL1 and MYXL2 contain the external routines required to run the application.

Examples

Subroutines that Use External Subroutine Linking (Method One)

The following code is an example of the use of Linking Method One. This method uses the PASSING keyword and each parameter is a record, record item, or temporary item that is being passed to the subroutine:

```
> SCREEN XBQO
> FILE DATA-FILE
> FILE COBOL-DATA DESIGNER
> TEMP NM-MEAN FLOAT SIZE 4
> TEMP NM-SDEV FLOAT SIZE 4
> TEMP CM-MEAN NONIEEE FLOAT SIZE 4
> TEMP CM-SDEV NONIEEE FLOAT SIZE 4
> TEMP COUNT INTEGER SIZE 2 INITIAL 10 RESET AT STARTUP
> TEMP CMEAN PACKED SIGNED SIZE 5
> TEMP CSDEV PACKED SIGNED SIZE 5
> TITLE "Do External Demo System - COBOL Passing Method" &
>    CENTERED AT 2,40
> SKIP TO 4
> ALIGN (1,4,21) (41,44,61)
> FIELD DATA+-KEY REQUIRED LOOKUP NOTON DATA-FILE
> FIELD DATA-NAME
> SKIP TO 6
> TITLE "NM-Data" CENTERED AT ,40
> ALIGN (1,,4)
> CLUSTER OCCURS WITH NM-Data FOR 1,16 AT ,1
> FIELD NM-DATA
> SKIP TO 9
> TITLE "CM-Data" CENTERED AT ,40
>  CLUSTER OCCURS WITH CM-DATA FOR 1,16 AT 10, 1
> FIELD CM-DATA
> CLUSTER
> SKIP TO 13
> ALIGN (1,4,21)
> FIELD NM-MEAN &
>    PICTURE " ^^^^.^^^^" OUTPUT SCALE 4 &
>     LEADING SIGN "-"  &
>     SIGNIFICANCE 6 DISPLAY NOID
> FIELD NM-SDEV &
>    SIGNIFICANCE 6 DISPLAY NOID
> SKIP TO 13
> ALIGN (41,44,61)
```

```
> FIELD CM-MEAN &
>    PICTURE "^^^^.^^^^" OUTPUT SCALE 4 &
>    LEADING SIGN "-" &
>    SIGNIFICANCE 6 DISPLAY NOID
> FIELD CM-SDEV &
>   PICTURE "^^^^.^^^^" OUTPUT SCALE 4 &
>   LEADING SIGN "-" &
>   SIGNIFICANCE 6 DISPLAY NOID
>
> TITLE "50 NM C" AT 16,1
> TITLE "51 NM COBOL" AT 17,1
> TITLE "52 NM FORTRAN" AT 18,1
> TITLE "70 CM SPL " AT 16,41
> PROCEDURE PREUPDATE
>   BEGIN
>    IF ALTEREDRECORD OF COBOL-DATA
>     THEN BEGIN
>      DELETE COBOL-DATA
>      END
>  END
>  PROCEDURE UPDATE
>   BEGIN
>    PUT DATA-FILE
>    PUT COBOL-DATA
>    END
> PROCEDURE INTERNAL DISPLAY-RESULTS
>   BEGIN
>    DISPLAY NM-MEA
>    DISPLAY NM-DSEV
>    DISPLAY CM-MEAN
>    DISPLAY CM-SDEV
>   END
>  PROCEDURE INTERNAL SETUP-COBOL
>   BEGIN
>    ; CAN'T FORGET ABOUT DECIMAL ALIGNMENT HERE!
>    FOR 10
>     BEGIN
>     LET NM-DATA OF COBOL-DATA = NM-DATA &
>       OF DATA FILE*10000
>     LET CM-DATA OF COBOL-DATA = CM-DATA &
>       OF DATA-FILE*10000
>       END
>   END
> ;CALL A NATIVE MODE C EXTERNAL
> PROCEDURE DESIGNER 50
>  BEGIN
>    DO EXTERNAL "nbc setup" PASSING COUNT
>    DO EXTERNAL "nbc SDEV" PASSING DATA-FILE
>    DO EXTERNAL "nbc getresults" PASSING NM-MEAN, &
>      NM-SDEV
>    DO INTERNAL DISPLAY-RESULTS
>  END
>
> PROCEDURE DESIGNER 51
>  BEGIN
>   DO INTERNAL SETUP-COBOL
>   DO EXTERNAL NBBSETUP PASSING COUNT
>   DO EXTERNAL NBBSDEV PASSING COBOL-DATA
>   DO EXTERNAL NBBGETRESULTS PASSING CMEAN, CSDEV
>   LET NM-MEAN = CMEAN / 10000
>   LET NM-SDEV = CSDEV / 10000
>   DO INTERNAL DISPLAY-RESULTS
>  END
> PROCEDURE DESIGNER 52
>  BEGIN
>   DO EXTERNAL NBFSETUP PASSING COUNT
>   DO EXTERNAL NBFSDEV PASSING DATA-FILE
>   DO EXTERNAL NBFGETRESULTS PASSING NM-MEAN, NM-SDEV
```

```
>   DO INTERNAL DISPLAY-RESULTS
>  END
> PROCEDURE DESIGNER 70
>  BEGIN
>  DO EXTERNAL CM "CBS'SETUP" PASSING COUNT
>  DO EXTERNAL CM "CBS'SDEV" PASSING DATA-FILE
>  DO EXTERNAL CM "CBS'GETRESULTS" PASSING CM-MEAN, CMSDEV
>   DO INTERNAL DISPLAY RESULTS
>  END
> BUILD
```

The following sections show the Cobol subroutines that are called from the DO EXTERNAL verbs in DESIGNER procedure 51 in the previous QDESIGN program.

The Cobol Subroutine NBBSETUP

```
$CONTROL DYNAMIC
 IDENTIFICATION DIVISION.
 PROGRAM-ID. NBBSETUP.
*
* NBBSETUP initializes the common area structure used
* by the standard deviation function. All that is
* required is that the count field be set.
ENVIRONMENT DIVISION.
DATA DIVISION.
LINKAGE SECTION.
01  COMMON-AREA
    05 COM-COUNTPIC 9(4)COMP.
    05 COM-MEANPIC S9(4)V9(4)COMP-3.
    05 COM-SDEVPIC S9(4)V9(4)COMP-3.
01  COMMON-AREA-SIZEPIC S9 (4)COMP.
01 LINK-PARMS
    05 COUNT-PARMPIC S9(4)COMP.
PROCEDURE DIVISION
    USING COMMON-AREA, COMMON-AREA-SIZE, LINK-PARMS.
*
* First ensure that the common area size is large
* enough for your application needs. If not, the
* QKGO file needs to be changed to accommodate a
* larger size. Remember that the common area
* size is in 16-bit words.
*
* 0000-START.
   IF COMMON-AREA-SIZE IS LESS THAN 7
    CALL INTRINSIC "QUIT" USING -1.
*
* Size is ok, initialize.
*
  1000-WORK.
   MOVE COUNT-PARM TO COM-COUNT.
*
* Return to QUICK
*
  9000-EXIT.
  GOBACK.
```

The Cobol Subroutine NBBSDEV

```
$CONTROL DYNAMIC
 IDENTIFICATION DIVISION
 PROGRAM-ID. NBBSDEV.
*
* NBBSDEV calculates the mean and standard deviation
* for an array of numbers. The structure COMMON-AREA
* is used as a communication block for the number of
* elements in the array as well as the results of
* this function.
ENVIRONMENT DIVISION.
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
01 SUM-OF-ELEMENTSPIC S9(4)V9(4)COMP-3.
01 SUM-OF DIFF-SQPIC S9(4)V9(4)COMP-3.
01 SUM-OF-DIFF-SQ-OVER-COUNTPIC S9(4)V9(4)COMP-3.
01
LINKAGE SECTION
01 COMMON AREA.
   05 COM-COUNTPIC 9(4)COMP.
   05 COM-MEANPIC S9(4)V9(4)COMP-3.
   05 COM-SDEVPIC S9(4)V9(4)COMP-3.

01 COMMON-AREA-SIZEPIC S9(4)COMP.

01 LINK-PARMS
   05 COBOL-DATA.
      10 DATA-KEYPIC S9(4)COMP.
      10 DATA-NAMEPIC X(14).
      10 NM-DATAPIC S9(4)V9(4)COMP-3
       OCCURS 10 TIMES.
      10 CM-DATAPIC S9(4)V9(4)COMP-3
       OCCURS 10 TIMES.
PROCEDURE DIVISION
 USING COMMON-AREA, COMMON-AREA-SIZE, LINK-PARMS.
* First ensure that the common area size is
* large enough for your application needs. If
* not, the QKGO file for the application must
* be changed to accommodate a larger size.
* Remember that the common area size is in
* 16-bit words.
*
 0000-START
  IF COMMON-AREA-SIZE IS LESS THAN 7
   CALL INTRINSIC "QUIT" USING -1.
1000-INIT.
   MOVE 0 TO SUM-OF-DIFF-SQ-OVER-COUNT.
   MOVE 0 TO SUM-OF-ELEMENTS.
   MOVE 0 TO SUM-OF-DIFF-SQ.

2000-MEAN
   PERFORM 9100-SUM-OF-DATA VARYING I FROM 1 BY 1
     UNTIL I IS GREATER THAN COM-COUNT.
   DIVIDE SUM-OF-ELEMENTS BY COM-COUNT GIVING COM-MEAN.

3000-SDEV.
   PERFORM 9200-SUM-DIFF-SQ VARYING I FROM 1 BY 1
     UNTIL I IS GREATER THAN COM-COUNT.
   DIVIDE SUM-OF-DIFF-SQ BY COM-COUNT.
     GIVING SUM-OF-DIFF-SQ-OVER-COUNT.
   COMPUTE COM-SDEV = SUM-OF-DIFF-SQ-OVER-COUNT ** 0.5.
*
* Return to QUICK

*
 9000-EXIT.
   GOBACK.
*
* Accumulate NM-Data
*
 9100-SUM-DATA
  ADD NM-DATA(I) TO SUM-OF-ELEMENTS.
*
* Accumulate sum of differences square
*
 9200-SUM-DIFF-SQ.
   COMPUTE SUM-OF-DIFF-SQ = SUM OF DIFF-SQ
     (NM-DATE(I) - COM-MEAN * (NM-DATA(I) - COM-MEAN).
```

The Cobol Subroutine NBBGETRESULTS

```
$CONTROL DYNAMIC
 IDENTIFICATION DIVISION
 PROGRAM-ID. NBBGETRESULTS.
*
* NBBGETRESULTS moves the results of the standard
* deviation calculation into the supplied results
* parameters.
*
 ENVIRONMENT DIVISION.
 DATA DIVISION
 LINKAGE SECTION.
 01 COMMON AREA.
 05 COM-COUNTPIC 9(4)COMP.
 05 COM-MEANPIC S9(4)V9(4)COMP-3.
 05 COM-SDEVPIC S9(4)V9(4)COMP-3.
 05 COMMON-AREA-SIZEPIC S9(4)COMP.
 05 LINK-PARMS.
 05   MEAN-PARMPIC S9(4)V9(4)COMP-3.
 05   FILLERPIC X(3).
 05   SDEV-PARMPIC S9(4)V9(4)COMP-3.
 PROCEDURE DIVISION
  USING COMMON-AREA, COMMON-AREA-SIZE, LINK-PARMS.
*
* First ensure that the common area size is large enough
* for your application requirements. If not, the QKGO
* file for the application needs to be changed to
* accommodate a larger size. Remember that the common
* area size is in 16-bit words.
*
 0000-START.
  IF COMMON-AREA SIZE IS LESS THAN 7
   CALL INTRINSIC "QUIT" USING -1.
*
* Size is ok, so copy out the results.

*
 1000-WORK.

   MOVE COM-MEAN TO MEAN-PARM.
   MOVE COM-SDEV TO SDEV-PARM.
*
* Return to QUICK
*
 9000-EXIT.
   GOBACK.
```

DO EXTERNAL (OpenVMS)

For DO EXTERNAL (**MPE/iX**), see (p. 390).

For DO EXTERNAL (**UNIX**), see (p. 406).

For DO EXTERNAL (**Windows**), see (p. 412).

Executes an external program.

Syntax

DO EXTERNAL name|string [PASSING file|item [,file|item]...]

or

DO EXTERNAL name|string (parm [,parm]...)

parm

Describes what is being passed to a subroutine, as follows:

file

Passes the address of the records buffer for the file.

item

Passes the address of the item.

REFERENCE (file|item)

Passes the address of the record buffer for the file or the address of the item.

VALUE (item)

Passes the value of the item (can be type INTEGER, FLOAT, VMSDATE, INTERVAL, NUMERIC, DATE, PHDATE, or JDATE). Types less than 32 bits are extended to 32 bits. INTEGER size 6 is extended to 8 bytes (64 bits).

DESCRIPTOR (file|item)

Passes the address of the standard OpenVMS descriptor for the record buffer of the file or the address of the standard OpenVMS descriptor of the item.

PASSING file|item [,file|item]

Passes the address of the common area, the address of a 2-byte integer containing the common size area, and the address of the pass list to the program. When a repeating item is specified in the parameter list, only the address of the first occurrence will be passed.

Several of the preceding parm options produce equivalent results. The various options allow designers to follow application standards.

Discussion

Use the DO EXTERNAL verb when you want to use an external subroutine written in COBOL, FORTRAN, C, or other programming languages that use the standard OpenVMS calling convention. A maximum of 100 parameters (consisting of files or items) can be passed. (See the next section, "Linking to External Subroutines", for a detailed discussion.)

Any record buffer for an Oracle Rdb relation that is passed to an external routine via DO EXTERNAL has a 2-byte integer null flag for each item in the record buffer that allows null values. A value of 0 means the item exists; a non-zero value means the item is null.

Note: For information about verb and procedure compatibility, see (p. 239).

Linking to External Subroutines

The DO EXTERNAL verb lets you perform editing or other localized processing in traditional programming languages. External subroutines shouldn't be used to replace QUICK's standard functions such as finding or updating the PRIMARY file of a screen; such activity interferes with QUICK. Similarly, input or output to the terminal isn't recommended since QUICK needs to know what is displayed on the screen at all times. This section provides examples of C, COBOL, and FORTRAN subroutines called from QUICK.

The DO EXTERNAL verb calls a separate program that performs the subroutine calls. The default program name is QKDRIVER, but another name can be used if specified in the QKGO file. This program can be constructed using the BUILDEXTERNAL command procedure.

For more information about the BUILDEXTERNAL command procedure, see (p. 404).

Because the DO EXTERNAL driver is a shared library that's run from within the QUICK process, it uses the run-time library routine, LIB$FIND_IMAGE_SYMBOL, to locate routines to call from QUICK.

If QUICK is installed with privileges, then the shared QKDRIVER module must be installed. System managers should ensure that a QKDRIVER module doesn’t violate system security before installing it.

When PowerHouse is installed with privileges, the external driver (default name QKDRIVER) must also be installed. If the image isn't located in the SYS$SHARE directory, a unique name that points to the installed image must be defined in the system logical name table.

Since the QKDRIVER runs in the same process as QUICK, it shares the same memory and resources. Therefore, external modules should be tested outside of QUICK before being used in a QKDRIVER module. Dynamic memory should be obtained from LIB$GET_VM and released with LIB$FREE VM. If event flags are needed, they should be obtained with LIB$GET_EF and returned with LIB$FREE_EF. Errors can be returned to QUICK by using LIB$SIGNAL. QUICK formats and displays the message vector and flags an error condition.

Program Sections

Some language compilers create Program Sections (PSECTs) that are writable and sharable. In such cases, you must install QKDRIVER with shared write attributes. However, this can lead to problems when the image is shared between several processes: when QKDRIVER is installed with shared write attributes, the processes can overwrite each other's data values.

To prevent processes from overwriting other processes' data values, you can restrict the attributes of specific PSECTs to WRT and NOSHR. If any PSECTs remain that have WRT and SHR attributes, the QKDRIVER image must be installed with shared write attributes. Follow these steps to restrict the attributes of any (or all) PSECTs:

1. Use the BUILDEXTERNAL command. If you specified the /NOSHARE option, you must include both the /MAP and /FULL options following the link options list, as in

   ```
   $ BUILDEXTERNAL EDPART1, EDPART2
   $_   EDPART1,EDPART2/MAP/FULL
   ```

   If you specified the /SHARE option, the MAP and OPTION files are built automatically.

   The share options, /SHARE and /NOSHARE, are explained in greater detail in the "Building the Subroutine Driver Program" section in this chapter. If you require more information about the BUILDEXTERNAL command, help and examples are available by entering

   ```
   BUILDEXTERNAL ?
   ```

   at the DCL prompt.
2. Look at the Program Section Synopsis of the MAP file and locate any PSECTs with attributes listed as SHR and WRT. Record the names of those PSECTs for use in Step 3.
3. For each PSECT with the attributes SHR and WRT, determine whether or not you want the PSECT to share the data within that section. If the data is to be shared, keep the original shared write attributes of the PSECT. If you don't want the data shared, note the name of that PSECT for use in Step 4.
4. Using the text editor, create a link-options file with the extension code OPT. The link-options file must contain one line for each PSECT that you recorded in Step 3 (that is, for each program section to which you want to assign WRT and NOSHR attributes), in this form:

```
PSECT_ATTR=<Psectname>,WRT,NOSHR
```

5. Using the link-options file, perform the BUILDEXTERNAL a second time, as in

```
$ BUILDEXTERNAL EDPART1, EDPART2 -
$_   EDPART1,EDPART2/MAP/FULL, <option file>/OPT
```

6. If you allowed any of the PSECTs to retain shared write attributes, the QKDRIVER image must also be installed with the /SHARE and /WRITE attributes.

Calling External Subroutines from QUICK

There are two methods for calling external subroutines from QUICK: the method that uses a common area, and the direct method.

To call an external subroutine using the common area method, use

DO EXTERNAL name [PASSING filelitem [,filelitem]...]

where name is the name of the external subroutine and each parm is a record, record item, or temporary item that is being passed to the routine.

QUICK passes three COBOL-compatible addresses: common area, common area size, and pass list.

The common area is allocated and initialized to binary zeros (but not otherwise used) by QUICK. It may be used to communicate between different external subroutines or between different calls to the same subroutine.

The common area size is a one-word integer containing the size (in words) of the common area as specified in QKGO.

The pass list is the address of a data structure containing all records and items specified in the parameter list. Within the data structure, the parameters are in the order specified in the parameter list and are longword aligned (four bytes). The external subroutine must allow for any slack bytes required.

To call an external subroutine using the direct method, use

DO EXTERNAL name (filelitem [,filelitem]...)

The direct method of calling external subroutines is to omit the keyword PASSING and list the parameters in parentheses as file or item addresses, descriptor addresses, or item values. The parameters listed are passed, but the common area or common area size isn't.

Additional Notes on External Subroutines

Whichever method you use, the called subroutine expects to receive the parameters passed to it in an acceptable form. For instance, FORTRAN programs require the use of descriptors when you pass character strings to them. The following example shows the QDESIGN code necessary to pass a character item named A and a numeric item named B to a FORTRAN program:

```
DO EXTERNAL subroutinename (DESCRIPTOR(A),B)
```

For both methods, if the same item is passed more than once in the same call, the last occurrence of that item determines the final value of the item in QUICK.

By default, QUICK allows zero words in the common area. This setting can be reallocated by changing the QKGO parameter common area size. If the called subroutine uses the common area, you should check the passed common area size to ensure that sufficient size was specified in QKGO.

It's not necessary to specify a parameter passing list. In such a case, the common area address, common area size, and a dummy parameter address are passed, in that order.

If you're using the direct method of calling subroutines with optional parameters, you can pass empty parameter lists.

All subroutines within a system of screens should use the same layout for the common area (if passed).

Parameters passed may be records, record items, or temporary items. Not all item type formats available in QUICK are available in the called languages. A maximum of 100 parameters can be included in any pass list.

Temporary areas are set up for each passed parameter. On return to QUICK, the values of all passed parameters are compared to their original values before the call. If any have changed, the original value is altered to reflect the change. If record items are changed, the record status is set to reflect this, provided that the parameters were passed by REFERENCE or DESCRIPTOR.If the items in an empty record buffer (status of New, Unchanged, Undeleted) are changed, the status will be New, Changed, Undeleted on return to QUICK. This may affect subsequent processing if the subroutine is used to retrieve an existing record and place it in an empty buffer.

If the subroutine accesses files, it must open and close them. The considerations are the same as those for file sharing between processes.

Building the Subroutine Driver Program

The DO EXTERNAL verb requires a driver program that contains a look-up table of subroutine names and addresses and all user-callable subroutines used by a system of QUICK screens. This program is built using the BUILDEXTERNAL command procedure supplied with the QUICK installation package.

Examples

The Screen TRANSACT in this example makes use of the DO EXTERNAL verb to invoke an external program to perform an edit check:

```
> SCREEN TRANSACT
>
> TEMPORARY STATUS NUMERIC*4
>
> FILE TRANSACTIONS
>
> SKIP TO LINE 4
> HILITE TITLE INVERSE
> TITLE "Transaction Screen" CENTRED
> DRAW FROM 3,15 TO 5,65
> SKIP 2
>
> GENERATE NOLIST
>
> PROCEDURE EDIT TRANS-NO
>      BEGIN
>          LET STATUS_CODE = 0
>          DO EXTERNAL EDITTRANS PASSING TRANS-NO, &
>              STATUS_CODE
>          IF STATUS CODE NE 0
>              THEN ERROR "Entered number failed edit check."
>      END
>
> BUILD
```

In the preceding example, the external program EDITTRANS is called to perform an edit check of the current transaction number. The temporary variable, STATUS_CODE, is set by the external program EDITTRANS to indicate whether or not the edit was successful. If the edit fails, (that is, if a non-zero status code is returned), the user receives an error message.

The following example illustrates calling a C subroutine using the direct method. The syntax for the direct method does not use the PASSING keyword and the parameters are enclosed in parentheses:

```
> SCREEN EDPART1
> FILE ORDER-INFO
> TEMP PROD-OK CHARACTER*2
> TEMP TEMP-NO INTEGER*4 SIZE 2
> TEMP ERR-MSG CHARACTER*12
>
> FIELD INVOICE-NUMBER OF ORDER-INFO
> FIELD PROD-NUMBER OF ORDER-INFO
> FIELD QTY-ORDERED OF ORDER-INFO
>
> PROCEDURE EDIT PROD-NUMBER
```

```
>      BEGIN
>         LET PROD-OK = "Y"
>         LET TEMP-NO = PROD-NUMBER
>         DO EXTERNAL EDPART1, (PROD-OK, ERR-MSG, &
>            VALUE(TEMP-NO))
>       IF PROD-OK <> "Y"
             THEN ERROR = "Product Number invalid" &
                + ERR-MSG + " " + ASCII (PROD-NUMBER)
> END
```

The following example is the C code for the subroutine called using the direct method:

```
/*
  C example using direct method.
*/
edpart1(prod_ok, err_msg, prod_no)
   char         prod_ok[];
   char         *err_msg;
   short int    prod_no;
   {
/*
   The following routine checks for a product number greater than 7777 and, if
found, flags an error to be returned to the calling QUICK Screen.
*/
   if (prod_no > 7777){
/*
      Flag an error on the product number.
*/
      strncpy(err_msg, "too big", 13);
      prod_ok[0]='N';
      prod_ok[1]='\0';      /* Null terminate string */
```

The following example illustrates the calling of a FORTRAN subroutine using the direct method. The syntax for the direct method does not use the PASSING keyword and the parameters are enclosed in parentheses:

```
> SCREEN EDPART2
> FILE ORDER-INFO
> TEMP PROD-OK CHARACTER*2
> TEMP TEMP-NO INTEGER*4 SIZE 2
> TEMP ERR-MSG CHARACTER*13
> FIELD INVOICE-NUMBER OF ORDER-INFO
> FIELD PROD-NUMBER OF ORDER-INFO
> FIELD QTY-ORDERED OF ORDER-INFO
>
> PROCEDURE EDIT PROD-NUMBER
>      BEGIN
>         LET PROD-OK = "Y"
>         LET TEMP-NO = PROD-NUMBER
>         DO EXTERNAL EDPART2 (DESCRIPTOR(PROD-OK), &
>            DESCRIPTOR(ERR_MSG),TEMP-NO)
>         IF PROD-OK <> "Y"
>         THEN ERROR = "Product Number invalid" &
>            + ERR-MSG + " " + ASCII(PROD-NUMBER)
> END
```

The following example is the FORTRAN code for the subroutine called using the direct method:

```
SUBROUTINE EDPART2(PROD_OK,ERR_MSG,PROD_NO)
CHARACTER*2 PROD_OK
CHARACTER*13 ERR_MSG
INTEGER*2 PROD_NO

IF (PROD_NO .GT. 7777) THEN
  ERR_MSG = 'too big      '
  PROD_OK = 'N'
END IF

RETURN
END
```

The following example illustrates calling a COBOL subroutine that uses a common area. The common area method uses the PASSING keyword and each parameter is a record, record item, or temporary item that is being passed to the subroutine:

```
> SCREEN EDPART3
> FILE ORDER-INFO
> TEMP PROD-OK CHARACTER*2
> TEMP TEMP-NO INTEGER*4 SIZE 2
> TEMP ERR-MSG CHARACTER*13
> FIELD INVOICE-NUMBER OF ORDER-INFO
> FIELD PROD-NUMBER OF ORDER-INFO
> FIELD QTY-ORDERED OF ORDER-INFO
> PROCEDURE EDIT PROD-NUMBER
>
> BEGIN
>      LET PROD-OK = "Y"
>      LET TEMP-NO = PROD-NUMBER
>      DO EXTERNAL EDPART3 PASSING PROD-OK, ERR-MSG, &
>          TEMP-NO
>      IF PROD-OK <> "Y"
>      THEN ERROR = "Product Number invalid" &
>          + ERR-MSG + " " + ASCII(PROD-NUMBER)
> END
```

The following example calls the COBOL subroutine using the direct method. The syntax for the direct method does not use the PASSING keyword and the parameters are enclosed in parentheses:

```
> SCREEN EDPART4
> FILE ORDER-INFO
> TEMP PROD-OK CHARACTER*2
> TEMP TEMP-NO INTEGER*4 SIZE 2
>  TEMP ERR-MSG CHARACTER*13
> FIELD INVOICE-NUMBER OF ORDER-INFO
> FIELD PROD-NUMBER OF ORDER-INFO
> FIELD QTY-ORDERED OF ORDER-INFO
> PROCEDURE EDIT PROD-NUMBER
>      BEGIN
>          LET PROD-OK = "Y"
>          LET TEMP-NO = PROD-NUMBER
>          DO EXTERNAL EDPART3 (PROD-OK, ERR-MSG, TEMP-NO)
>      IF PROD-OK <> "Y"
>      THEN ERROR = "Product Number invalid" &
>          + ERR-MSG + " " + ASCII (PROD-NUMBER)
> END
```

The following example is the COBOL code for the subroutine called using both the direct and common area methods. It performs a simple edit.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EDPART3.
ENVIRONMENT DIVISION.
DATA DIVISION.
LINKAGE SECTION.
01 COMMON-AREA              PIC X.
01 COMMON-AREA-SIZE         PIC S9(4) COMP.
01 PASS-LIST.
   05 PROD-OK               PIC X(2).
   05 FILLER                PIC X(2).
   05 ERR-MSG               PIC X(13).
   05 FILLER                PIC X(3).
   05 PROD-NO               PIC S9(4) COMP.
PROCEDURE DIVISION USING COMMON-AREA
                         COMMON-AREA-SIZE
                         PASS-LIST.
A00.
   IF PROD-NO IS GREATER THAN 7777
     MOVE "too big" TO ERR-MSG
     MOVE "N"       TO PROD-OK.
   EXIT PROGRAM.
```

Note: Code for other DO EXTERNAL examples can be found $powerhouse:[<version>.demo.external].

The BUILDEXTERNAL Command

The BUILDEXTERNAL command builds the driver image required for DO EXTERNAL routines in QUICK.The syntax takes the general form:

BUILDEXTERNAL [share-option] subroutine-list
link-options [driver-name]

share-option

The share-option is one of the following:

/SHARE

Indicates that a shared image is to be built. Images that can be shared are linked dynamically at run-time and no subprocess is spawned. The file extension of the external driver must be .EXE.

/NOSHARE

Indicates that an executable image is to be built. When this option is used to build the external routine, a call to an external routine spawns a subprocess and executes the external routine within the subprocess. The file extension of the external driver must be .PHEXE.

You can build and debug external routines by using /NOSHARE. Once the application is ready for production runs, it can be converted to a shared image for greater performance.

Limit: The options /NOSHARE and /SHARE must be entered immediately after the BUILDEXTERNAL command.

Default: /SHARE

subroutine-list

Lists the names of the external routines called from QUICK. The name may be listed in a file which can then be used with the at-sign (@).

Limit: Separate the names with a comma; do not include spaces.

link-options

Specifies the options necessary to build the image. The options include:

- object files containing external routines
- object libraries
- link commands (for example, /DEBUG)
- link option files

driver-name

Names the external driver image.

Default: QKDRIVER

The BUILDEXTERNAL driver name extensions must be either .EXE or .PHEXE. Any other extension is invalid and causes an error. The file extensions of the driver name specified in QKGO must be either .EXE or .PHEXE.

Creating Subroutines

The BUILDEXTERNAL command must include a list of subroutines and link options that designate the subroutine locations. Additional link options (see the OpenVMS Linker Utility Manual) and the driver program name can also be supplied.

The subroutine-list parameter may be a single name or a list of names separated by commas but with no intervening spaces. Each name may be either a subroutine entry point name or, when preceded by an at-sign (@), a text file (default extension code .DAT) containing a list of subroutine entry point names (one per line), as in

```
$ BUILDEXTERNAL EDPART1,EDPART2,EDPART3,-
$_ EDPART4 EDPART1,EDPART2,EDPART3,EDPART4
```

Each routine is compiled into a separate object file. The compiled subroutines listed in the link-options are assumed to have the file extension .OBJ. The compiled subroutines EDPART1.OBJ, EDPART2.OBJ, EDPART3.OBJ, and EDPART4.OBJ are used by Linker to resolve the subroutine entry points. The driver program name defaults to QKDRIVER.

Creating an Object Library

The following commands create an object library and add each module to it. The BUILDEXTERNAL command then uses the object library to resolve the subroutine entry points.

```
$ CC      EDPART1
$ FORTRAN EDPART2
$ COBOL   EDPART3
$ COBOL   EDPART4
$ LIBRARY/CREATE EDPART EDPART1, EDPART2, -
$_     EDPART3, EDPART4
$ BUILDEXTERNAL EDPART1, EDPART2, EDPART3, EDPART4 -
$_     EDPART/LIB
```

If the file EDPART.DAT contains EDPART1, EDPART2, EDPART3, and EDPART4, and if the driver name is QUICKSUB, then the command

```
$ BUILDEXTERNAL @EDPART EDPART/LIB QUICKSUB
```

can be used. This would search the library EDPART for the names listed in the text file EDPART.DAT.

DO EXTERNAL (UNIX)

For DO EXTERNAL (**MPE/iX**), see (p. 390).

For DO EXTERNAL (**OpenVMS**), see (p. 398).

For DO EXTERNAL (**Windows**), see (p. 412).

Executes an external subroutine.

Syntax

DO EXTERNAL name|string ([parm [,parm]...])

or

DO EXTERNAL name|string [PASSING file|item [,file|item]...]

name|string

The name of the external subroutine.

parm

Describes what is being passed to the subroutine.

The parm can be:

file

Passes the address of the record buffer for the file.

item

Passes the address of the item.

REFERENCE (file|item)

Passes the address of the record buffer for the file or the address of the item.

VALUE (item)

Passes the value of the item. The item datatype must be one of INTEGER, FLOAT, NUMERIC, INTERVAL, DATE, JDATE, or PHDATE.

PASSING file|item [,file|item]...

Passes the address of the common area (as declared in the QKGO file), the address of an integer of size 2 containing the common area size, and the address of the pass list to the subroutine. The pass list is a data structure consisting of the files and items specified. The files and items may be aligned according to machine-specific requirements. When a repeating item is specified in the parameter list, only the address of the first occurrence will be passed.

Note: Parameter buffers to an external subroutine are always aligned correctly for any valid datatype, but items within record buffers are not normally aligned with respect to earlier items.

The common area is allocated and initialized to binary zeros by QUICK; otherwise, it is not used. It can be used to communicate between different external subroutines or between different calls to the same subroutine.

The common area size is an integer of size 2 containing the size (in words) of the common area as specified in QKGO's common area size parameter. By default, QUICK allows one external subroutine and one word in the common area. Use QKGO parameters to increase both of these settings. For more information, see (p. 255).

Limit: Any value returned by an external routine will be discarded.

Note: There is no limit on the number of external subroutines that can be invoked.

Limit: A maximum of 16 files, items, or parms can be passed. However, the effective limit may be less than this in some cases, in particular when a 64-bit floating-point parameter is passed by value. If you do specify parameters over the 16-parameter limit, QUICK issues an error.

Discussion

An external subroutine is written in COBOL, FORTRAN, C, Pascal, or another traditional programming language and is not declared in the screen design.

Use the DO EXTERNAL verb when you want to use localized processing with traditional programming languages. Because they interfere with QUICK, external subroutines should not be used to replace QUICK's standard functions, such as finding or updating the PRIMARY file of a screen. Similarly, input or output to the terminal from localized processing is not recommended, since QUICK needs to know what is displayed on the screen at all times.

In order for QUICK to link to the external routine, a distinct process must be created using a script called BuildExternal. For more information about the BuildExternal script, see (p. 407).

QUICK aligns all data to be passed to external routines on machine alignment requirements. If the language uses alignment keywords (as Pascal does), be careful when passing entire data records. Parameter buffers to an external routine are always aligned correctly for any valid datatype. However, items within record buffers are not normally aligned with respect to earlier items.

If the invoked subroutine uses the common area, it should check the passed common area size to ensure that a sufficient size was specified in QKGO.

You don't have to specify a pass list. If you don't, the addresses for the common area, the common area size, and an empty pass list are passed.

All external subroutines within a system of screens should use the same layout for the common area, if it is passed.

Parameters passed can be records, record items, or temporary items. A maximum of 16 parameters can be included in any pass list. Predefined items (FIELDTEXT, FIELDVALUE, PATH) and defined items cannot be passed. If an external subroutine is invoked in the EDIT procedure, the address of the record item is passed. Record item names that are used as parameters are not trapped and replaced by the contents of the FIELDTEXT or FIELDVALUE predefined item. To pass the value in the FIELDTEXT or FIELDVALUE predefined item, use a temporary item.

Temporary areas are set up for each passed parameter. On return to QUICK, the values of all passed parameters are compared to their values before being invoked. If any values have changed, the original value is altered to reflect the change. If record items are changed, the record status is reset to reflect this fact. If the items in an empty record buffer (status of New, Unchanged, Undeleted) are changed, the status is New, Changed, Undeleted on return to QUICK. Subsequent processing can be affected if a subroutine is used to retrieve an existing record and place it in an empty buffer.

If the same item is passed more than once in the same call, only the last occurrence of that item is checked for value changes on return to QUICK.

If a subroutine must access a file, the subroutine must open and close that file. The considerations are the same as those for file sharing between many processes. If the external routine returns a value, it is ignored by QUICK, and you cannot access it.

QUICK can accommodate programming languages that support lowercase names for functions or subroutines. When QUICK searches for an identifier in the symbol table, it can recognize a name that is all in lowercase, uppercase, or mixed case.

Note: For information about verb and procedure compatibility, see (p. 239).

The BuildExternal Script

In order to link to the external routine, QUICK must create a distinct process. Included with PowerHouse is a script named BuildExternal. BuildExternal is a C Shell (*csh*) script that takes routine names, object files, and an optional executable (driver) name and builds a program to communicate with QUICK in order to accomplish processing.

BuildExternal takes a variety of arguments and transforms them into a driver program. The two required arguments which you must supply to BuildExternal are a list of function or procedure names and a list (or library) of object files. The optional argument is the name of the executable driver that QUICK starts when it runs.

Note: If the driver option is specified, an environment variable must be set to indicate to QUICK that a name other than qkdriver is being used as in

```
setenv QKDRIVER ./routines.out
```

The syntax of the BuildExternal script takes the general form:

BuildExternal routine_names object_files [driver_name]

routine_names

The argument routine_names is either a comma-separated list of function/procedure names or a filename that contains a list of routine names. In the latter case, the filename begins with the "@" character.

The following are examples of routine_names where change is the name in the C language declaration in the examples in the previous section.

```
BuildExternal change ...
BuildExternal @routines ...
```

object_files

The argument object_files is either a comma-separated list of filenames with .o extensions or an archived library containing contents of object files. (See the UNIX command *ar(1)* for archiving. See *f77(1)* and *cc(1)* for discussions on object files.)

Since PowerHouse 5GL is a 32-bit application, object files must also be 32 bit. If not, BuildExternal will issue errors.

The lib/<name>.a option is the method to supply an archived library to BuildExternal.

The following are examples of object files:

```
BuildExternal change change.o
BuildExternal @routines lib/routines.a
```

driver_name

The argument driver_name is optional. It is a UNIX executable file containing the communication code and linked copies of your compiled external routines. By default, its name is qkdriver.

The following examples show the use of a driver name

```
BuildExternal change change.o
```

creating an executable named qkdriver.

```
BuildExternal @routines lib/routines.a routines.out
```

creating an executable named routines.out.

Examples

The screen TRANSACT in this example makes use of the DO EXTERNAL verb to invoke an external program (written in C) to perform an edit check:

```
> SCREEN TRANSACT
>
> TEMPORARY STATUS_CODE NUMERIC*4
>
> FILE TRANSACTIONS
>
> SKIP TO LINE 4
> HILITE TITLE INVERSE
> TITLE "TRANSACTION SCREEN" CENTERED
> DRAW FROM 3,15 TO 5,65
> SKIP 2
>
> GENERATE NOLIST
>
> PROCEDURE EDIT TRANS-NO
>   BEGIN
>     LET STATUS_CODE = 0
>     DO EXTERNAL EDITTRANS PASSING &
>        TRANS-NO, STATUS_CODE
>     IF STATUS_CODE NE 0
```

```
>       THEN ERROR "ENTERED TRANSACTION NUMBER DID" &
>          "NOT PASS EDIT CHECK."
>     END
>
> BUILD
```

In the preceding example, the external program EDITTRANS is called to perform an edit check of the current transaction number. The temporary variable, STATUS_CODE, is set by the external program EDITTRANS to indicate whether or not the edit was successful. If the edit fails (that is, if a non-zero status code is returned), the user receives an error message.

External Subroutine Linking (Method One)

The syntax of the first linking method is

DO EXTERNAL namelstring [PASSING filelitem [,filelitem]...]

The following example illustrates how to invoke a subroutine that performs a part-number edit using the first method of external subroutine linking. A flag is passed in order to check the results.

The screen design is as follows:

```
> SCREEN LINK
> FILE ORDER
> TEMPORARY TEST-FLAG INTEGER
> TEMPORARY TEMP-NO CHARACTER*10
> FIELD ORDER-NO OF ORDER
> FIELD PART-NO OF ORDER
> FIELD ORDER-QTY OF ORDER
> PROCEDURE EDIT PART-NO
> BEGIN
>   LET TEST-FLAG = 0
>   LET TEMP-NO = PART-NO
>   DO EXTERNAL EDPART PASSING TEST-FLAG, TEMP-NO
>   IF TEST-FLAG <> 0
>    THEN ERROR = "INVALID PART NUMBER:ERROR TYPE " &
>       + ASCII (TEST-FLAG)
>   END
> BUILD
```

The FORTRAN subroutine is called:

```
      subroutine edpart (com, comlem, pass)
      integer comlen
      integer com(comlen)
      integer pass(6)
      character*10 tempno
      integer string(6)
      equivalence (tempno, string(2))
      flag = pass(1)
      do 10 i = 2, 6
10    string(i) = pass(i)
.
.
.
      pass(1) = flag
      do 20 i = 2, 6
20    pass(i) = string(i)
      return
      end
```

External Subroutine Linking (Method Two)

The second method of linking to external subroutines is omitting the PASSING keyword and listing all parameters in parentheses. The listed parameters are passed, but the common area and common area size are not passed. The syntax of the second linking method is

DO EXTERNAL namelstring ([parm [,parm]...])

The screen design needed for the second method of invoking external subroutines is identical to the first method, except that the PASSING option is not used on the DO EXTERNAL verb:

```
> SCREEN LINK
```

```
> FILE ORDER
> TEMPORARY TEST-FLAG INTEGER
> TEMPORARY TEMP-NO CHARACTER*10
> FIELD ORDER-NO OF ORDER
> FIELD PART-NO OF ORDER
> FIELD ORDER-QTY OF ORDER
> PROCEDURE EDIT PART-NO
> BEGIN
>   LET TEST-FLAG = 0
>   LET TEMP-NO = PART-NO
>   DO EXTERNAL EDPART (TEST-FLAG, TEMP-NO)
>   IF TEST-FLAG <> 0
>    THEN ERROR = "INVALID PART NUMBER:ERROR TYPE " &
>       + ASCII (TEST-FLAG)
>   END
> BUILD
```

The FORTRAN subroutine using the second method is:

```
      subroutine edpart (flag, tempno)
      integer flag
      integer tempno(5)
      integer string(5)
      character*10 partno
      equivalence (partno, string(1)
      do 10 i = 1, 5
10    string(i) = tempno(i)
      .
      .
      .
      do 20 i = 1, 5
20    tempno(i) = string(i)
      return
      end
```

Additional External Procedure Examples

The following example illustrates how to pass a temporary variable to an external C procedure.

The screen design is as follows:

```
> SCREEN QKCHANGE MENU
> TEMPORARY ARGUMENT CHARACTER*15
.
.
.
> BEGIN
>   DO EXTERNAL CHANGE (ARGUMENT)
>   DISPLAY ARGUMENT
>   END
> BUILD
```

The C routine to accept the temporary item is as follows:

```
extern char *memcpy();
extern int  memcmp();
void change( value )
char *value; /* Reference of argument as
              * to byte ptr.
              */
{
   if( ( memcmp( value, "Hello World", 11 )) == 0)
       (void)memcpy( value, "Hello Quick", 11 );
   else
       (void)memcpy( value, "GoodBye Quick", 13 );
}
```

The following example illustrates how to invoke an external C procedure to accept the BILLINGS record.

The screen design is as follows:

```
> SCREEN QKCHANGE
```

```
> FILE BILLINGS PRIMARY
> FIELD EMPLOYEE OF BILLINGS REQUIRED NOCHANGE
> FIELD MONTH OF BILLINGS
> FIELD PROJECT OF BILLINGS REQUIRED NOCHANGE
> FIELD BILLING OF BILLINGS
.
.
.
> PROCEDURE DESIGNER 10
> BEGIN
>   DO EXTERNAL CHANGE( BILLINGS )
>   DISPLAY EMPLOYEE OF BILLINGS
>   DISPLAY MONTH OF BILLINGS
>   DISPLAY PROJECT OF BILLINGS
>   DISPLAY BILLING OF BILLINGS
>   END
> BUILD
```

C Language statements which accept the BILLINGS record are

```
void change( buffer)
int *buffer;
{
/* The structure defined below and pointer(*record)
 * is cast from the record that QUICK passes.
 * See the C language guide for more information
 * on structures and casting.
 */
struct a {
   short     int employee;
   int    month;
   int    project;
   int    billing;   /* as packed signed */
} *record = (struct a *)buffer;

 record->employee =  42; /* Change the employee number. */
 record->project  =  42; /* Change the project. No format */

}
```

QDESIGN - DO EXTERNAL (Windows)

For DO EXTERNAL (**MPE/iX**), see (p. 390).

For DO EXTERNAL (**OpenVMS**), see (p. 398).

For DO EXTERNAL (**UNIX**), see (p. 406).

Executes an external subroutine.

Syntax

DO EXTERNAL [C|PASCAL] name|string ([parm [,parm]...])

or

DO EXTERNAL [C|PASCAL] name|string [PASSING file|item [,file|item]...]

C|PASCAL

Names the type of calling convention used.

The PASCAL format reads parameters right to left, the standard order expected internally by Windows.

Special care must be taken upon return from a function when using the PASCAL parameter passing convention. The calling function must allow for sixteen parameters because a variable number of parameters is not supported. The called function may expect anywhere up to and including sixteen parameters. It is the responsibility of the called function to remove the parameters off of the program stack. Since the caller must provide sixteen parameters and the called function may or may not handle that many parameters it is necessary to have the caller make sure that the stack is properly reset upon return.

Default: C

name

The name of the external subroutine.

PowerHouse looks for the specified name. If it does not find the name, it will look for it in the .QKI file, in QKGO or in the QKDRIVER environment variable.

string

A string naming the external subroutine.

Describes the library and the entry point to call in the form "library-name@entry-point". The library name can be fully qualified. If specified, the library file is loaded from the named folder; otherwise, Windows searches for the library file using standard Windows search rules. If the string contains only a name, it is treated as a ?name parameter. By default, PowerHouse for NT/2000/XP looks for the external subroutine in the .QKI file, in QKGO or in the QKDRIVER environment variable.

parm

Describes what is being passed to the subroutine.

The parm can be:

file

Passes the address of the record buffer for the file.

item

Passes the address of the item.

REFERENCE (file|item)

Passes the address of the record buffer for the file or the address of the item.

VALUE (item)

Passes the value of the item (can be type INTEGER, FLOAT, NUMERIC, DATE, PHDATE, or JDATE). Types less than 32 bits are extended to 32 bits. INTEGER size 6 is extended to 8 bytes (64 bits).

PASSING file|item [,file|item]...

Passes the address of the common area (as declared in the QKGO file), the address of an integer of size 2 containing the common area size, and the address of the pass list to the subroutine. The pass list is a data structure consisting of the files and items specified. The files and items may be aligned according to machine-specific requirements. When a repeating item is specified in the parameter list, only the address of the first occurrence will be passed.

Note: Parameter buffers to an external subroutine are always aligned correctly for any valid datatype, but items within record buffers are not normally aligned with respect to earlier items.

The common area is allocated and initialized to binary zeros by QUICK; otherwise, it is not used. It can be used to communicate between different external subroutines or between different calls to the same subroutine.

The common area size is an integer of size 2 containing the size (in words) of the common area as specified in QKGO's common area size parameter. By default, QUICK allows one external subroutine and one word in the common area. Use QKGO parameters to increase both of these settings.

Limit: Any value returned by an external routine will be discarded.

Note: There is no limit on the number of external subroutines that can be invoked.

Windows limits the number of passed parameters to sixteen files or items. The effective limit may be less in some cases, in particular when a 64-bit floating point parameter is passed by value. That single parameter would use four of the available sixteen parameter positions.

Discussion

Windows allows programs to link to a library of routines dynamically through dynamic link libraries (DLLs). PowerHouse 4GL for NT/2000/XP allows the programmer to use these functions using the DO EXTERNAL procedural verb.

DO EXTERNAL supports the standard C calling convention and the PASCAL (or FORTRAN) calling convention. Your external function can be written in any language whose compiler supports these calling conventions.

Steps

1. Create a function and place it in a Windows Dynamic Link Library.
2. Create and compile PowerHouse screens with a DO EXTERNAL command.
3. Place the Windows DLL in the Windows folder.
4. Run the PowerHouse screen.

Formats of Do External Calls

Form 1: Using the PASSING Keyword

DO EXTERNAL [C|PASCAL] name|string [PASSING file|item [,file|item]...]

This format of DO EXTERNAL issues the call to the specified external routine and passes the following parameters:

- Address of the common area was been initialized to binary zeros as declared in the .QKI file, in QKGO, or in the QKDRIVER environment variable.
- Address of a two-byte integer containing the size in words of the common area.
- Address of the pass list, which is a data structure consisting of the structures and items specified. Entries in the pass list are aligned on word boundaries (16 bits).

By default PowerHouse allows one external subroutine and one word in the common area. These values can be customized in the .QKI file, in QKGO, or in the QKDRIVER environment variable.

General External Function Format:

C:

```
void FAR extfuncname(int FAR * com_area, int FAR * com_size, int FAR *
passlist);
```

PASCAL:

```
void FAR PASCAL extfuncname(int FAR * com_area, int FAR * com_size, int FAR *
passlist);
```

All pointer references are FAR pointers in both forms of the external function call.

Form 2: Not Using the Passing Keyword

The second method of linking to external subroutines is to omit the PASSING keyword and to list all the parameters in parentheses. The syntax of this form is

DO EXTERNAL [C|PASCAL] name|string ([parm [,parm]...])

This format of DO EXTERNAL dynamically loads the DLL corresponding to the specified name and a call is issued to the function with the name specified in the syntax. The common area and the common area size are not passed to the external function. As specified in the calling syntax, either the address or the value of the structure or the item is passed to the external subroutine.

General External Function Format:

C:

```
void FAR extfuncname(int FAR * buffer, ... );
```

PASCAL:

```
void FAR PASCAL extfuncname(int FAR * buffer1, int FAR * buffer2, ... , int FAR
* buffer16,);
```

All pointer references are FAR pointers in both forms of the external function call.

Examples

Example: The DLL (Dynamic Link Library)

The C source code and the definition file for a simple dynamic link library that could be used as a template for your own DLL routines.

Inside the C source code there are two other functions, LibMain and WEP. LibMain is the entry point of the library; it is called by Windows to initialize the DLL.

The WEP (Windows Exit Procedure) function performs clean up for the DLL before the library is unloaded.

The definition file is similar to the definition file for all Windows programs, with the exception of the first statement. In Windows EXE definition files, the first statement is the NAME statement, but in DLLs the first statement is the LIBRARY statement. The name you give your DLL can be anything you want, but it must be unique throughout your system.

C Source Code

```
#include <stdio.h>
#include <conio.h>
/*Function Prototype*/
void __declspsec( dllexport ) SimpleFunction (char[20],char[20]);

/*Function Implementation*/
void __declspsec( dllexport ) SimpleFunction (char text[20],char title[20])
{
   int ch;
```

```
   text[19]='\0';
   title[19]='\0';

   printf ("\nText=[%s]",text
   printf ("\nTitle=[%s]",text
   printf ("\nPress<ENTER>to continue");

   ch=getch();
   return;
}
```

Example: Calling AX_DLL from a Screen

The following example is used to display the use of external calls from within PowerHouse 4GL:

```
SCREEN X
TEMPORARY title CHAR SIZE 20
TEMPORARY text CHAR SIZE 20
...
PROCEDURE DESIGNER x NODATA
BEGIN
        LET title = "DoExternal"
        LET text = "This is the text"
        DO EXTERNAL "DOEXT@SimpleFunction" (text,title)
END
```

The actual DO EXTERNAL call is one that requires a closer look. The format of the command has changed from other forms of the PowerHouse command set.

```
DO EXTERNAL "<DLL NAME>@_<ROUTINE NAME>" (<PARAMETER LIST>)
```

PowerHouse passes the address of the variable to the external function only by default. The source code for the external routine accepts both parameters as addresses. You can pass the value of the parameters by using the VALUE option.

Example: Calling GetProfileString

This code shows a call to the GetProfileString function in KERNEL, by using the function to determine the location of Excel according to the win.ini configuration file. The PASCAL option is added to the DO EXTERNAL verb so that the parameters are read in the order required by Windows. If the location of Excel is found, its path is displayed. Otherwise, the default return value 'Excel not found' is displayed.

```
SCREEN getpriv MENU
TEMP iNull INTEGER SIZE 2 INITIAL 0 RESET AT STARTUP
TEMP szNull CHARACTER SIZE 1 INITIAL char(iNull) RESET AT STARTUP
;These are the temporary variables used as parameters to pass to
;the function GetProfileString. See the Windows 3.1 SDK for
;details.
TEMP lpszSection CHARACTER SIZE 30 RESET AT STARTUP INIT " "
TEMP lpszEntry CHARACTER SIZE 30 RESET AT STARTUP INIT " "
TEMP lpszDefault CHARACTER SIZE 30 RESET AT STARTUP INIT " "
TEMP lpszReturn CHARACTER SIZE 128 RESET AT STARTUP INIT " "
TEMP iReturn INTEGER SIZE 2 RESET AT STARTUP INIT 128
TEMP length INTEGER SIZE 2 RESET AT STARTUP INIT 0
TEMP t_excel CHARACTER SIZE 128 RESET AT STARTUP
PROCEDURE DESIGNER excl NODATA
BEGIN
   LET lpszSection = "Extensions" + szNull
   LET lpszEntry = "xls" + szNull
   LET lpszDefault = "Excel not found" + szNull
```

This is the actual call to the function. As a general rule, it works best if you pass strings by reference and numbers by value.

```
DO EXTERNAL PASCAL "Kernel@GetProfileString" (REF(lpszSection),
REF(lpszEntry), REF(lpszDefault), REF(lpszReturn), VALUE(iReturn))
   IF lpszReturn <> ("Excel not found" + szNull)
   THEN BEGIN
      LET length = INDEX(lpszReturn, " ")
```

```
    END
    ELSE BEGIN
       LET length = INDEX(lpszReturn, szNull)
    END
    LET length = length - 1
    LET t_excel=lpszReturn[1:length]
    DISPLAY t_excel
END
```

Windows Search Rules

Windows searches in this order:

1. current directory
2. directory containing the executable file for the current task
3. 32-bit Windows system directory.

   The function GetSystemDirectory can be used to get the path of this directory which is named SYSTEM32.
4. 16-bit Windows directory named SYSTEM
5. Windows directory.

   The following function can be used to get the path of this directory:

   ```
   GetSystemDirectory<JavaScript:hhobj 5. Click()>
   ```
6. directories in your path

DO INTERNAL

Executes an internal procedure.

Syntax

DO [INTERNAL] name

name

Names an existing INTERNAL procedure.

Limit: 64 characters

Discussion

An INTERNAL procedure is a subroutine written in QDESIGN procedural code. The INTERNAL procedure must be declared in the procedure section of the screen design before the procedure in which it is invoked. The corresponding procedure executes when this statement is encountered.

You can make recursive procedure calls within INTERNAL procedures.

Example

The do internal verb references a previously declared internal procedure, as shown in the following example:

```
> SCREEN INVCHK
> FILE INVOICES
>
> FIELD INVOICENO OF INVOICES &
>       REQUIRED NOCHANGE LOOKUP NOTON INVOICES
> FIELD QUANTITY OF INVOICES
> FIELD PRICE OF INVOICES
> FIELD TAX OF INVOICES
> FIELD TOTAL OF INVOICES
> PROCEDURE INTERNAL STANDARDCHECK
>   BEGIN
>     IF TOTAL NE QUANTITY * (PRICE + TAX)
>       THEN ERROR "Check-digit edit failed"
>     END
> PROCEDURE EDIT TOTAL
>   BEGIN
>     DO INTERNAL STANDARDCHECK
>     END
>
> BUILD
```

EDIT

Edits a value in a field.

Syntax

EDIT field

field

Names the field that corresponds to the item to be edited.

Discussion

The EDIT verb initiates activities that perform value checks and lookups, and executes EDIT and PROCESS procedures for the named field.

Do not use the EDIT verb within the EDIT procedure of the same field. This will cause QUICK to loop.

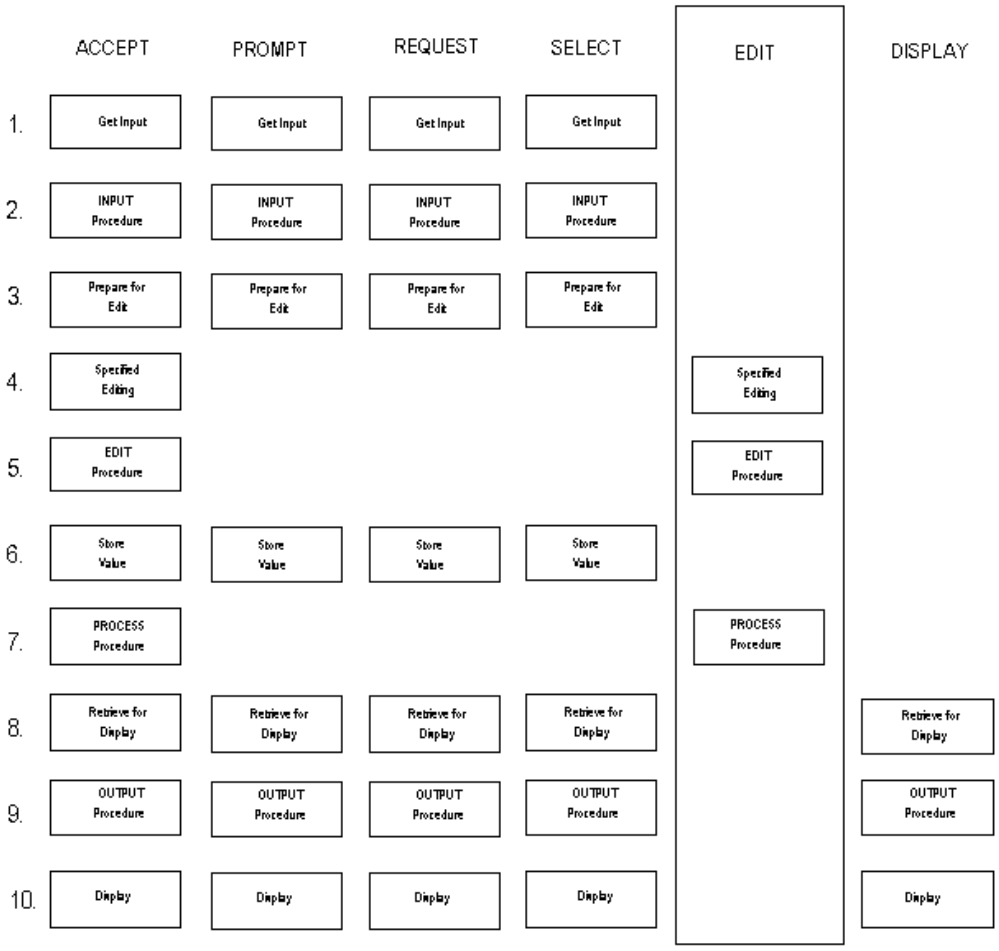
Note: For information about verb and procedure compatibility, see "Verb and Procedure Compatibility" (p. 239).

The EDIT Verb and SILENT Fields

The EDIT verb is normally used in conjunction with SILENT fields. The EDIT verb neither accepts data from the QUICK screen user nor displays data to the screen. Rather, the EDIT verb retrieves the value of the item from its associated buffer and moves it into the FIELDTEXT predefined item and (where appropriate) the FIELDVALUE predefined item. The content of the FIELDTEXT predefined item is consistent with an acceptable entry of the corresponding item type. QDESIGN then performs

- specified FIELD and dictionary editing
- the EDIT procedure for that field
- the PROCESS procedure for that field

The following figure illustrates the steps that are initiated by the EDIT verb, and contrasts these steps to similar steps performed by other field processing verbs:



For information about the steps in the preceding figure, see "Processes Initiated by the ACCEPT Verb" (p. 364).

Inclusion of EDIT overrides the following FIELD statement options: DISPLAY, FIXED, IF, OMIT, and NOENTRY.

Example

In the following example, an employees file is used to develop a screen that contains an employee's surname and first name. The field employee-NUMBER is a silent field that is used for lookup only, and QDESIGN generates an edit verb for this field in the default entry procedure. The EDIT verb causes QUICK to perform a lookup, ensuring that an EMPLOYEENUMBER for the specified LASTNAME and FIRSTNAME exists.

```
> SCREEN EMPNAMES
> FILE EMPLOYEES PRIMARY
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> FIELD FIRSTNAME OF EMPLOYEES
> FIELD EMPLOYEENUMBER OF EMPLOYEES &
>       SILENT LOOKUP ON EMPLOYEES
>
> PROCEDURE ENTRY
>   BEGIN
>   ACCEPT LASTNAME OF EMPLOYEES
>   ACCEPT FIRSTNAME OF EMPLOYEES
>   EDIT EMPLOYEENUMBER OF EMPLOYEES
>     END
> BUILD
```

ERROR

Stops processing and issues an error message.

Syntax

ERROR [MESSAGE] string|=string-expression|n

string

Defines the contents of the message using a string.

=string-expression

Defines the contents of the message using a string expression.

n

Defines a message number that corresponds to a message in the designer message file, which is the designated file QKMSGDES.

For more information about QDESIGN's message files, see Chapter 4, "Messages in PowerHouse", in the *PowerHouse Rules* book.

Discussion

The ERROR verb instructs QUICK to display the specified message on the message line. The message is displayed if the message line is empty or if it contains a WARNING or INFORMATION message. The ERROR verb doesn't overwrite a SEVERE verb message. The ERROR verb has the second highest priority (after SEVERE) in the four-level hierarchy of messages.

If an error occurs and the action is to back up to a previous ACCEPT or PROMPT verb, or a previous BLOCK TRANSFER control structure, QUICK follows this process:

1. Looks at each verb that was executed before the error occurred, starting from the verb immediately prior to the error.
2. If the verb is an ACCEPT, PROMPT, or BLOCK TRANSFER, QUICK executes it and continues processing.
3. If the verb is a DO INTERNAL verb, QUICK searches the internal procedure for the first ACCEPT, PROMPT, or BLOCK TRANSFER, executes it, and continues processing.
4. If there is no ACCEPT, PROMPT, or BLOCK TRANSFER in the internal procedure, QUICK continues searching as in Step 1.
5. If no ACCEPT, PROMPT, or BLOCK TRANSFER exists, QUICK prompts at the Action field.

If an error occurs and the action is to prompt at the Action field, the prompt is immediate; no other procedure is executed unless specifically stated in the procedure.

The following table describes error processing in all QUICK procedures except EDIT, INPUT, PROCESS, and OUTPUT, which are discussed in the next table.

Procedure	Action
APPEND	QUICK backs up to a previous ACCEPT or PROMPT verb, or to a previous BLOCK TRANSFER control structure. If there is no such verb or control structure, and the APPEND procedure was executed using a PERFORM APPEND verb in the ENTRY or MODIFY procedure, QUICK backs up to a previous ACCEPT, PROMPT, or BLOCK TRANSFER in the ENTRY or MODIFY procedure. QUICK prompts in the Action field if there is no such verb or control structure, or if the APPEND procedure was executed using the Append (A) or Update (U) Action field commands, or via an AUTOUPDATE option on the FIELD statement.

Procedure	Action
BACKOUT	The rest of the procedure is skipped, and QUICK continues backout processing.
DELETE	The rest of the procedure is skipped, and QUICK prompts at the Action field.
DESIGNER	QUICK backs up to a previous ACCEPT or PROMPT verb, or to a previous BLOCK TRANSFER control structure. If no such verb or control structure exists, QUICK prompts at the Action field.
DETAIL DELETE	The rest of the procedure is skipped, and QUICK prompts at the Action field.
DETAIL FIND	QUICK backs up to the last executed GET verb for the DETAIL file. If no such verb exists, QUICK prompts at the Action field without displaying any retrieved data.
DETAIL POSTFIND	The rest of the procedure is skipped, QUICK displays the data retrieved by the FIND and DETAIL FIND procedures, and prompts at the Action field.
ENTRY	QUICK backs up to the previous ACCEPT or PROMPT verb, or to a previous BLOCK TRANSFER control structure or DO INTERNAL call. If no such verb, control structure, or call exists, QUICK prompts at the Action field.
EXIT	QUICK backs up to a previous ACCEPT or PROMPT verb, or to a previous BLOCK TRANSFER control structure. If no such verb or control structure exists, control returns to the invoking screen.
FIND	QUICK backs up to the last GET verb for the PRIMARY file. If no such verb exists, QUICK prompts at the Action field without displaying retrieved data.
INITIALIZE	QUICK backs up to a previous ACCEPT or PROMPT verb, or to a previous BLOCK TRANSFER control structure. If no such verb or control structure exists, QUICK executes the EXIT procedure (if it exists), and returns control to the invoking system.
INTERNAL	QUICK follows the error processing for the procedure containing the DO INTERNAL verb.
MODIFY	QUICK backs up to the last ACCEPT or PROMPT verb, or the previous INTERNAL procedure, or to the last BLOCK TRANSFER control structure. If no such verb or control structure exists, QUICK prompts the user at the Action field.
PATH	The rest of the procedure is skipped, and QUICK prompts at the Action field without completing the retrieval cycle.
POSTFIND	The rest of the procedure is skipped, and QUICK prompts at the Action field without displaying retrieved data.
POSTPATH	The rest of the procedure is skipped, and QUICK prompts at the Action field without executing the retrieval cycle.
POSTUPDATE	QUICK skips the rest of the procedure and prompts at the Action field.
PREENTRY	QUICK backs up to the previous ACCEPT or PROMPT verb, or to a previous BLOCK TRANSFER control structure. If no such verb or control structure exists, QUICK prompts at the Action field without executing the ENTRY procedure.
PREUPDATE	The rest of the procedure is skipped, and QUICK prompts at the Action field without executing the UPDATE procedure.

Procedure	Action
SELECT	QUICK backs up to a previous ACCEPT or PROMPT verb, or to a previous BLOCK TRANSFER control structure. If no such verb or control structure exists, QUICK prompts at the Action field.
UDPATE	QUICK skips the rest of the procedure, rolls back any updates done in the UPDATE procedure to the point where the error occurred, and prompts at the Action field.

Errors that are issued from within INPUT, EDIT, PROCESS, and OUTPUT procedures executed by an ACCEPT, DISPLAY, PROMPT, REQUEST, or SELECT verb take precedence over normal error processing in the procedure containing that verb.

Errors issued from within EDIT and PROCESS procedures executed by an EDIT verb do not override normal error processing.

Error Processing for Procedures Initiated by Verbs

Procedure	Called by the Verb	Action
INPUT	ACCEPT, PROMPT, REQUEST, SELECT	The rest of the procedure is skipped and QUICK prompts at the current field.
EDIT	ACCEPT	The rest of the procedure is skipped and QUICK prompts at the current field.
	EDIT	The rest of the procedure is skipped and QUICK continues error processing for the procedure containing the EDIT verb.
PROCESS	ACCEPT	The rest of the procedure is skipped and QUICK prompts at the current field.
	EDIT	The rest of the procedure is skipped and QUICK continues error processing for the procedure containing the EDIT verb.
OUTPUT	ACCEPT, DISPLAY, PROMPT, REQUEST, SELECT	The rest of the procedure is skipped, QUICK displays the overflow character (by default, the crosshatch #) and QUICK continues processing.

Note: For information about verb and procedure compatibility, see (p. 239).

Example

This example counts the skills of selected company personnel. An error message appears if no data record associated with the user-entered employee number is available for retrieval. The value entered by the screen user is echoed back by concatenating it into the message string

```
> SCREEN EMPSKILL
>
> TEMPORARY COUNTER NUMERIC*3 INITIAL 0
>
> FILE EMPLOYEES PRIMARY
> FILE SKILLS DETAIL OCCURS 12
Item EMPNUM initialized (fixed) to EMPNUM OF EMPLOYEES.
> FIELD EMPNUM OF EMPLOYEES REQUIRED NOCHANGE &
>   LOOKUP NOTON EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> CLUSTER OCCURS WITH SKILLS
>   FIELD SKILLCODE
```

```
> CLUSTER
>
> PROCEDURE FIND
>     BEGIN
>       IF PATH = 1
>       THEN
>         BEGIN
>           GET EMPLOYEES VIA EMPNUM
>           IF NOT ACCESSOK
>             THEN ERROR = "Sorry," + &
>                  ASCII(EMPNUM) + &
>                  " is not an employee number."
>         END
>     IF PATH = 2
>       THEN GET EMPLOYEES VIA LASTNAME
>     IF PATH = 3
>       THEN GET EMPLOYEES SEQUENTIAL
>     END
>
> PROCEDURE DETAIL FIND
>   BEGIN
>     FOR SKILLS
>       BEGIN
>         GET SKILLS OPTIONAL
>         IF ACCESSOK
>           THEN LET COUNTER = COUNTER + 1
>           ELSE INFORMATION = "This employee has "&
>           + ASCII(COUNTER) + " skill(s)."
>         END
>     END
> BUILD
```

[SQL] FETCH

Retrieves the next row of data for the specified cursor.

Syntax

[SQL] FETCH cursor-reference [OPTIONAL]

cursor-reference

A cursor-name or table-name named in a CURSOR statement.

Discussion

The cursor will be opened if it is not already open. In this case, only the default substitution will take place. The execution of the FETCH retrieves the next row of data for the specified cursor.

The SQL FETCH can only be applied to relational tables, views or cursors declared with a CURSOR statement.

The predefined condition ACCESSOK behaves the same for FETCH as it does for GET.

FOR

Repeats procedural statements a number of times.

Syntax

FOR n

FOR [EACH] [ITEM] item

FOR [EACH] [DISPLAYED] [FILE] record-structure

FOR MISSING [ITEM] item

FOR MISSING [DISPLAYED] [FILE] record-structure

FOR n

Repeats the procedural statement following the FOR control structure n times.

Limit: Within a procedure, only one FOR control structure can be active at a time.

n

A number between 1 and 255.

FOR [EACH] [ITEM] item

Repeats the procedural statement following the FOR control structure once for each occurrence of the specified item.

EACH

For documentation only.

ITEM

For documentation only.

item

Specifies what occurring item the FOR control structure is to iterate over.

FOR [EACH] [DISPLAYED] [FILE] record-structure

For the specified record-structure, repeats the procedural statement following the FOR control structure once for each record buffer in the cache.

EACH

For documentation only.

DISPLAYED

Repeats the procedural statement only for those record buffers that are in the occurrence window. That is, QUICK starts the iteration at the first record buffer currently displayed on the screen and ends at the last displayed record buffer.

FILE

For documentation only.

record-structure

Specifies what record-structure the FOR control structure is to iterate over.

FOR MISSING [ITEM] item

Starts the iteration after the last non-empty occurrence of an item. It ends when the last record buffer in the cache is reached.

ITEM

For documentation only.

item

Specifies what occurring item the FOR control structure is to iterate over.

FOR MISSING [DISPLAYED] [FILE] record-structure

Starts the iteration of the FOR control structure after the last non-empty record buffer in the cache. It ends when the last record buffer in the cache is reached.

DISPLAYED

Repeats the procedural statement only for those empty record buffers in the occurrence window. That is, QUICK starts the iteration after the last non-empty record buffer currently displayed on the screen and ends at the last displayed record buffer.

FILE

For documentation only.

record-structure

Specifies what record-structure the FOR control structure is to iterate over.

Discussion

The FOR control structure allows repeated execution of a procedural statement.

FOR MISSING Control Structure

The FOR MISSING control structure is generated in the default MODIFY procedure, and it repeats as many times as there are empty record occurrences on the screen. A FOR MISSING control structure allows users to perform Append processing when changing data records using the Modify (M) Action field command.

Nesting FOR Control Structures

FOR control structures cannot be explicitly nested, but they can invoke an INTERNAL procedure that contains another FOR control structure. Similarly, the WHILE RETRIEVING control structure can't be explicitly nested within a FOR control structure.

Block Transfer and Cached Records

A BLOCK TRANSFER inside a FOR control structure where the associated record-structure is CACHED is not allowed and will cause a syntax error.

FOR Control Structure with Primary or Detail Record Structures

When the FOR control structure is used with a primary or detail record-structure which has an OCCURS or a CACHE option specified, the FOR control structure will end when

- in Entry mode, the cache is full or the screen user stops the entry sequence with a Skip All (//) or Backout (^) command. This occurs when you use the FOR control structure in the ENTRY and APPEND procedures.
- in Find or Select mode, all records are read or the cache is full. This occurs when you use the FOR control structure in the DETAIL FIND and FIND procedures.
- in Change or Correct mode, the last non-empty record buffer is reached.

- if no mode is active, or executed before the data records are entered or found, such as in the INITIALIZE procedure, the loop executes for the number of occurrences of the record-structure.

FOR Control Structure with Secondary or Designer Record Structures

For secondary and designer record-structures, the number of processing repetitions performed is equal to the number of occurrences declared on the FILE or CURSOR statement, regardless of the number of data records entered or retrieved.

FOR Control Structure with Repeating Items

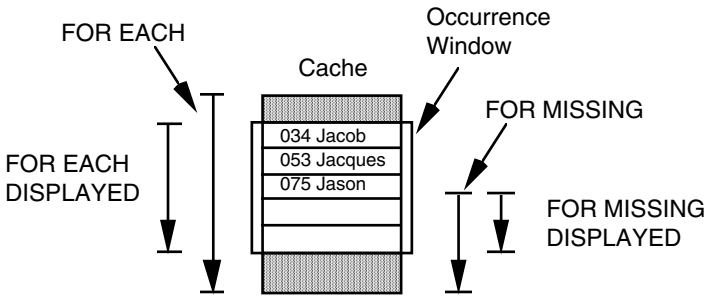
For repeating items, the number of processing repetitions performed is equal to the number of occurrences declared for that item, regardless of the number of items entered or assigned.

FOR EACH, FOR MISSING, and the DISPLAYED Option

These options of the FOR control structure control what part of the cache and what part of the occurrence window are to be iterated over.

FOR EACH

When you use the FOR EACH option, the FOR loop starts at the top of the cache and ends at the bottom of the cache. The DISPLAYED option limits the iteration to all record buffers currently in the occurrence window.



FOR MISSING

When you use the FOR MISSING option, the FOR control structure starts after the last non-empty record buffer and ends at the end of the cache. If the cache is full, the FOR control structure does not execute.

The FOR MISSING DISPLAYED option limits the iteration to all empty record buffers currently in the occurrence window. If there are no empty record buffers currently in the occurrence window (that is, the occurrence window is positioned above the empty record buffers in the cache), then the FOR control structure does not execute.

The FOR MISSING control structure is generated in the default MODIFY procedure. The FOR MISSING control structure allows QUICK screen users to perform Append processing when changing data records using the Modify (M) command. During MODIFY procedure processing, the FOR MISSING control structure starts Append processing and accepts records until ended with a Skip All (//) or Backout (^) command or the cache becomes full.

DISPLAYED

The designer may write code that iterates over only the record buffers in the cache currently on screen by using the DISPLAYED option.

Iterations in a FOR Control Structure

The number of processing repetitions in a FOR control structure is determined by one of the following specifications:

- the OCCURS option of the appropriate FILE, CURSOR, or TEMPORARY statement together with the number of repetitions of the item, the file, or the cursor

- the number specified in the OCCURS option

Examples

The following screen is used to browse through employees. The POSITION field indicates which records are currently being viewed. In this example:

- NUM_EMPLOYEES is the number of employee records in the cache.
- The FOR DISPLAYED loop is used to determine the first and last records displayed.
- The first FOR loop is used to determine the number of records entered or retrieved.
- The second FOR loop is used to initialize the record counter.
- The internal procedure UPDATE_POSITION is called to set the initial value of POSITION. The POSTSCROLL procedure is not called when the records are initially displayed, only when the records are actually scrolled.
- The POSTSCROLL procedure is invoked every time the records are scrolled. This procedure uses the internal procedure UPDATE_POSITION to update the POSITION field.

```
> SCREEN SCROLL_EMPLOYEES
>
> FILE EMPLOYEES OCCURS 15 CACHE 40
>
> TEMPORARY EMPLOYEE_NUM  INTEGER*3 UNSIGNED   &
>   OCCURS WITH EMPLOYEES
> TEMPORARY FIRST_EMPLOYEE RESET AT STARTUP
> TEMPORARY LAST_EMPLOYEE  RESET AT STARTUP
> TEMPORARY NUM_EMPLOYEES  RESET AT STARTUP
>
> DEFINE POSITION CHARACTER*60 = SUBSTITUTE   &
>   ( "You are viewing records ^ to ^ out of ^.", &
>             ASCII( FIRST_EMPLOYEE ), &
>             ASCII( LAST_EMPLOYEE  ), &
>             ASCII( NUM_EMPLOYEES  ) )
>
> ALIGN (11,,15) (,,20) (,,28) (,,43)
> TITLE "Record  Employee    First             Last"  &
>   AT 4,11
> TITLE "Number   Number      Name              Name"  &
>   AT 5,11
> CLUSTER OCCURS WITH EMPLOYEES
> FIELD EMPLOYEE_NUM PICTURE "^^^" SIGNIFICANCE 3 DISPLAY
> FIELD EMPLOYNO OF EMPLOYEES REQUIRED NOCHANGE  &
>   LOOKUP NOTON EMPLOYEES
> FIELD FIRSTNAME OF EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> CLUSTER
> SKIP
> ALIGN (,,11)
> FIELD POSITION ID 99
>
> PROCEDURE INTERNAL UPDATE_POSITION
>   BEGIN
>     LET FIRST_EMPLOYEE = 0
>     FOR DISPLAYED EMPLOYEES
>       BEGIN
>         IF FIRST_EMPLOYEE = 0
>         THEN
>           LET FIRST_EMPLOYEE = OCCURRENCE
>         ELSE
>           LET LAST_EMPLOYEE  = OCCURRENCE
>       END
>     FOR EMPLOYEES
>       BEGIN
>         LET NUM_EMPLOYEES = OCCURRENCE
>       END
>     DISPLAY POSITION
>   END
```

```
> PROCEDURE POSTFIND
>   BEGIN
>     FOR EACH EMPLOYEES
>       BEGIN
>         LET EMPLOYEE_NUM = OCCURRENCE
>       END
>     DO INTERNAL UPDATE_POSITION
>   END
>
>
>
> PROCEDURE POSTSCROLL
>   BEGIN
>     DO INTERNAL UPDATE_POSITION
>   END
```

```
MODE:F ACTION:

        Record  Employee    First            Last
        Number   Number      Name            Name
        01  016   1015   LINDA          RYAN
        02  017   1016   KEILA          NIKKANEN
        03  018   1017   DONNA          MCPHERSON
        04  019   1018   HEATHER        SAMPSON
        05  020   1019   GLENN          WALKER
        06  021   1020   JUDY           CLARK
        07  022   1021   PAUL           LARSON
        08  023   1022   LINDA          MANNING
        09  024   1023   THOMAS         WILLIAMS
        10  025   1024   SAMANTHA       KELLY
        11  026   1025   MAURICE        PHILLIPS
        12  027   1026   EDWARD         MITCHELL
        13  028   1027   MALCOLM        TIERNEY
        14  029   1028   CLAYTON        HARWOOD
        15  030   1029   SUZANNE        KNIGHT

        You are viewing records 16 to 30 out of 37.
```

How QUICK Sets the OCCURRENCE System Function

The setting of the OCCURRENCE system function is localized to the scope of the FOR loop in which it occurs. The following example illustrates the value of OCCURRENCE in an implicitly nested FOR control structure:

```
> 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 10 times. The following list shows how the item OCCURRENCE is set within each of these iterations:

Iteration	Value of OCCURRENCE
1	1,1,2,3,4,5,6,7,8,9,10,1
2	2,1,2,3,4,5,6,7,8,9,10,2
3	3,1,2,3,4,5,6,7,8,9,10,3
4	4,1,2,3,4,5,6,7,8,9,10,4
5	5,1,2,3,4,5,6,7,8,9,10,5
6	6,1,2,3,4,5,6,7,8,9,10,6
7	7,1,2,3,4,5,6,7,8,9,10,7
8	8,1,2,3,4,5,6,7,8,9,10,8
9	9,1,2,3,4,5,6,7,8,9,10,9
10	10,1,2,3,4,5,6,7,8,9,10,10

At any time, there is only one setting of the OCCURRENCE function.

If an item has multiple occurrences within a record-structure that has multiple occurrences, the OCCURRENCE function always refers to the record-structure, so the first item of the current data record is used. You can't address all the occurrences of a repeating item within a repeating record-structure on the same screen. Instead, you must pass each occurrence of the record-structure in turn to a subscreen and process the repeating item there.

GET

Retrieves a data record.

Syntax

GET record-structure [option]...

record-structure

Names the data record to be retrieved. The record-structure must be declared in the data dictionary.

Options

BACKWARDS

Reverses the sequence in which the data records are normally read.

Limit: Valid only for C-ISAM, DISAM, RMS ISAM, and IMAGE datasets with keyed access.

Limit: The BACKWARDS and SEQUENTIAL options cannot be used together for RMS ISAM files.

GENERIC|NOGENERIC

GENERIC allows partial index retrieval. NOGENERIC prevents partial index retrieval.

Limit: Not valid for IMAGE indexes, unless they are B-Tree or OMNIDEX indexes.

Default: GENERIC

OPTIONAL

Continues processing even if the access fails. With the exception of PRIMARY and DETAIL type files, retrieval is required unless the OPTIONAL keyword is used. If a data record isn't found, a data record is created containing initial values for each item. These values are taken from the data dictionary and any ITEM statements. If no initial values are specified in the data dictionary or an ITEM statement, CHARACTER items are initialized to spaces, and NUMERIC and DATE items are initialized to zero.

ORDERBY item [ASCENDING|DESCENDING] [,item[ASCENDING|DESCENDING]]...

Allows the ordered retrieval of records in a relational table or view by any column (or combination of columns) defined in the table or view.

If the ORDERBY option occurs with the VIAINDEX option, ordering is performed according to the columns of the ORDERBY option and the ordering imposed by the VIAINDEX option is ignored.

Limit: Valid only for relational structures.

Default: ASCENDING

SEQUENTIAL

Accesses the data records sequentially.

Limit: The SEQUENTIAL and USING options cannot be used in the same GET verb.

Limit: The BACKWARDS and SEQUENTIAL options cannot be used together for RMS ISAM files.

UNIQUE

Forces a re-evaluation of the USING expression and a "get first" access for each data record read. UNIQUE overrides chained-type access to indexed files and IMAGE datasets. For record-structures in direct and relative files, the option allows calculation of the data record number for each individual data record.

Limit: Applies only to PRIMARY, SECONDARY, and DETAIL files.

USING expression [,expression]...

Accesses the data records of the specified record-structure using the results of the specified expression as

- the value for corresponding segments
- the data record number for record-structures in direct files or relative files
- the column value in a relational table

For indexed files or IMAGE datasets, there can be more than one value (for segmented indexes), but QUICK interprets the values as a single index value in that file.

If the record-structure belongs to a DIRECT file, there can only be one expression which must be numeric. Otherwise, a series of expressions may be specified which correspond one-to-one with the segments established by either the VIA or VIAINDEX options.

If neither the VIA nor the VIAINDEX option is specified, and the record-structure has only one associated index structure, this index structure is used as if the VIAINDEX option had been specified except that index retrieval order will not be enforced.

If a record-structure is a relational table, there can be several values in the USING option which QDESIGN interprets as the values of the columns in the table. The VIA or VIAINDEX options must be used to indicate which columns the values belong to if more than one index is in use or if no index is used.

If the VIA option is specified, the number of expressions specified must correspond one-to-one with the number of segments specified on the VIA option.

If the VIAINDEX option is specified and the VIA option isn't specified, the number of expressions specified may be less than or equal to the number of segments contained within the index structure specified. There must be at least one expression.

Limit: 255 expressions.

Limit (**MPE/iX**): IMAGE does not support retrieval via an initial subset of the segments of a multi-segment index, unless the index is a B-Tree or OMNIDEX index. An expression must be specified for every segment of the index.

VIAINDEX indexname

Names an index of an indexed file, or IMAGE dataset, or relational table. When VIAINDEX is used with the USING option, there can be as many USING values as there are segments in the index, or fewer values than the index segments. In the latter case, the values are matched to the index segments in order, starting from the first segment; the leftover segments are not used. When using VIAINDEX, the retrieval always follows the order specified by that index.

Use VIA instead of VIAINDEX with relational tables. By explicitly referencing an index with the VIAINDEX option, it becomes harder to change the database definitions. If the index is deleted, the source code must be modified. If VIA is used instead, the index can be deleted and the screen continues to work properly.

VIA linkitem [,linkitem]...[ORDERED[ASCENDING| DESCENDING]]

Accesses the record-structure via the specified segments. Linkitem is a segment in an index for an indexed file, or a column in a relational database.

When a VIA list is used in combination with the USING option, there must be a one-to-one match between the USING expressions and VIA linkitems. This option is valid only for indexed files, IMAGE datasets, and relational tables.

For indexed files, and IMAGE datasets, the series of linkitems declared must define a series of segments contained within the index structure associated with the record-structure. In this case, the first linkitem is the first segment within the index structure, the second linkitem is the second segment, and so on.

For relational tables, a series of linkitems may represent any series of columns in a table as long as the VIAINDEX option is not specified. If VIAINDEX is specified, a series of linkitems must be a series of segments contained within a specific index structure: match the first linkitem to the first segment, the second linkitem to the second segment, and so on.

THE ORDERED option allows ordered retrieval of records in a relational table or view by any column or combination of columns defined in a table of view.

The ORDERED option is a convenient method of specifying ORDERBY items when the specified items are the same as those in the VIA list.

If ORDERED option occurs with the VIAINDEX option, which also imposes an ordering, the ordering is done by the columns of the VIA option. The implicit ordering imposed by the VIAINDEX option is ignored.

Limit: 255 segments.

Limit (**MPE/iX**): IMAGE does not support retrieval via an initial subset of the segments of a multi-segment index, unless the index is a B-Tree or OMNIDEX index. The series of linkitems must include all of the segments in the index.

Discussion

The GET verb retrieves one data record of the named file and moves it into the screen's record buffer area. From the record buffer, the data record can be displayed, changed, or deleted. If more than one access path is possible for a given record-structure, you must provide sufficient information (preferably by using the ACCESS statement) to specify the access path explicitly.

The GET verb returns only data records that pass the selection conditions established by either a SELECT statement or a selection value for the record-structure, as specified in the data dictionary.

When a file has a selection value, and a GET OPTIONAL is done, only the first record is read. To force reading all records until the selection passes or end of file, GET SEQUENTIAL OPTIONAL must be specified.

QUICK locks indexed files for record retrieval. By default, the GET verb locks the file, retrieves the record, and unlocks the file. The locking is conditional unless overridden by the designer. The number of lock attempts and the interval between attempts can also be controlled by using QKGO parameters.

If the lock cannot be obtained, an error condition is flagged and the effect is the same as if an ERROR verb had been issued. An exception occurs in the FIND procedure with SECONDARY and DETAIL files. If a lock cannot be obtained (using the appropriate number of attempts) on a SECONDARY or DETAIL file, the error is flagged and the procedure stops. Any data retrieved to that point is displayed along with the lock error message. The user can make changes to the data, even though the logical transaction may be incomplete. This occurs even if retrieval is required (the OPTIONAL keyword has been removed).

Normally, if an error is issued in a FIND procedure, QUICK backs up to the last GET verb for the PRIMARY file, if one exists. QUICK does not back up when a lock cannot be obtained because, if the SECONDARY or DETAIL file remains locked, PRIMARY file records could be retrieved without anything ever being displayed to the user. The wait could be considerable.

To work around this situation:

- To lock all SECONDARY and DETAIL files (indexed only) that are retrieved in the FIND procedure, add a LOCK verb in the FIND procedure before the GET verb for the PRIMARY file. An UNLOCK verb should be added to unlock the files at the end of the FIND procedure. With the LOCK verb, the files are specifically locked prior to any attempt at retrieval. If the locks cannot be obtained, the appropriate message is issued and the cursor returns to the Action field.

- **MPE/iX, UNIX, Windows:** Use the QKGO parameter LOCK UNCONDITIONAL to specify that the first lock in a series of locks is unconditional. The lock around the read would then be unconditional as long as no other file or database is locked at read time.

Note: For information about verb and procedure compatibility, see (p. 239).

GET Locks the File for Concurrent Read (OpenVMS)

The GET verb uses VMS Lock Management Services (LMS) to lock the file for concurrent read when it retrieves a record from a file. Concurrent read allows other users read or write access to the file and provides the requestor of the lock (in this case, the GET verb) with read access only. The GET verb releases the current read lock after the record is read.

Example

This example counts the skills of selected company personnel. In this example:

- GET retrieves data records from EMPLOYEES via the index EMPNUM.
- SEQUENTIAL specifies the default file access when the QUICK screen user makes a null entry in Find mode.
- OPTIONAL allows processing to continue when a data record from the SKILLS file isn't retrieved.

```
> SCREEN EMPSKILL
> TEMPORARY COUNTER NUMBERIC*3 INITIAL 0
> FILE EMPLOYEES PRIMARY
> FILE SKILLS DETAIL OCCURS 12
Item EMPNUM initialized (fixed) to EMPNUM OF EMPLOYEES.
> FIELD EMPNUM OF EMPLOYEES &
>       REQUIRED NOCHANGE LOOKUP NOTON EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> CLUSTER OCCURS WITH SKILLS
>   FIELD SKILLCODE OF SKILLS
> CLUSTER
> PROCEDURE FIND
>   BEGIN
>     IF PATH = 1
>       THEN BEGIN
>         GET EMPLOYEES VIA EMPNUM OPTIONAL
>         IF NOT ACCESSOK
>           THEN ERROR = &
>             "Sorry, " + &
>              ASCII(EMPNUM) + "is not an" &
>               + " employee number."
>         END
>     IF PATH = 2
>       THEN GET EMPLOYEES VIA LASTNAME
>     IF PATH = 3
>       THEN GET EMPLOYEES SEQUENTIAL
>     END
> PROCEDURE DETAIL FIND
>   BEGIN
>     FOR SKILLS
>       BEGOM
>         GET SKILLS OPTIONAL
>         IF ACCESSOK
>           THEN LET COUNTER = COUNTER + 1
>           ELSE INFORMATION = &
>             "This employee has " + &
>             ASCII(COUNTER) + " skill(s)."
>         END
>     END
>
> BUILD
```

IF

Establishes a conditional statement.

Syntax

IF condition
THEN procedural statement
[ELSE procedural statement]

condition

States a condition to be evaluated.

THEN procedural statement

Executes the specified procedural statement if the condition is satisfied.

ELSE procedural statement

Executes the specified procedural statement if the condition isn't satisfied.

Discussion

The IF control structure allows conditional execution of a procedural statement.

Note: IF control structures can be nested.

An IF option of a FIELD statement causes an IF control structure to be included in the ENTRY procedure.

The THEN and ELSE keywords must each begin on a separate line.

Example

This example counts the skills of selected company personnel. In this example:

- The IF structure establishes the condition that must be met before the THEN structure begins execution of the following block.
- IF NOT ACCESSOK establishes a condition that must be satisfied before the next procedural statement is executed. If the employee number entered doesn't match a record in SKILLS, QUICK displays an error message.

```
> SCREEN EMPSKILL
>
> TEMPORARY COUNTER NUMERIC*3 INITIAL 0
> FILE EMPLOYEES PRIMARY
> FILE SKILLS DETAIL OCCURS 12
Item EMPNUM initialized (fixed) to EMPNUM OF EMPLOYEES.
>
> FIELD EMPNUM OF EMPLOYEES REQUIRED NOCHANGE &
>    LOOKUP NOTON EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> CLUSTER OCCURS WITH SKILLS
>   FIELD SKILLCODE
> CLUSTER
>
> PROCEDURE FIND
>   BEGIN
>     IF PATH = 1
>       THEN BEGIN
>         GET EMPLOYEES VIA EMPNUM
>         IF NOT ACCESSOK
>           THEN ERROR = &
>             "Sorry, " + ASCII(EMPNUM) + &
>             " is not a valid employee number."
>         END
```

```
>     IF PATH = 2
>       THEN GET EMPLOYEES VIA LASTNAME
>     IF PATH = 3
>       THEN GET EMPLOYEES SEQUENTIAL
> END
>
> PROCEDURE DETAIL FIND
>   BEGIN
>     FOR SKILLS
>       BEGIN
>         GET SKILLS OPTIONAL VIA EMPNUM
>         IF ACCESSOK
>           THEN LET COUNTER = COUNTER + 1
>         ELSE INFORMATION = &
>           "This employee has " + &
>           ASCII(counter) + " skill(s)."
>         END
>     END
>
> BUILD
```

For additional examples of how to use the IF control structure, see (p. 319) and (p. 335).

INFORMATION

Issues an informational message.

Syntax

INFORMATION [MESSAGE] string|=string-expression|n
[NOW [RESPONSE]]

string

Defines the contents of the message using a string.

=string-expression

Defines the contents of the message using a string expression.

n

Defines a message number that corresponds to a message in a designer message file. The designated file is QKMSGDES.

For more information about message files, see "Messages in PowerHouse" in the *PowerHouse Rules* book.

NOW [RESPONSE]

Forces the display of the specified message on the screen when the INFORMATION verb is encountered in a procedure. Otherwise, the message isn't displayed until the next Input or Output action. When you use the NOW option, QUICK writes the contents of the display buffer to the screen. This action refreshes the screen and function key labels.

RESPONSE temporarily erases the top line of the screen, and prompts the screen user to press [Return].

Discussion

An INFORMATION message has the lowest priority in the four-level message hierarchy. The INFORMATION verb instructs QUICK to display the stated message, but has no effect on processing unless the RESPONSE option is used. (There is one exception: if the string is greater than the screen width, you are prompted to press Return even if the RESPONSE option is not used.) After the message is displayed, the message line in the display buffer is cleared.

An INFORMATION message is displayed only when the message line is empty, unless the NOW option is included.

Note: For information about verb and procedure compatibility, see (p. 239).

[SQL] INSERT

Adds new rows to a table.

Syntax

```
    [SQL[IN database]
    [TRANSACTION transaction_name
        [FOR {CONSISTENCY|{[CONCURRENCY]
            phase-option [,phase-option]...}}]]]
INSERT INTO tablespec
    [(column-name [,column-name]...)]
query-expression|{VALUES (sql-expression | NULL
[,sql-expression|NULL]...) [RETURNING DBKEY INTO :item]}
```

IN database

The name of the database. The database must be attached to the current data dictionary.

TRANSACTION transaction_name [FOR {CONSISTENCY| {[CONCURRENCY] phase-option[,phase-option]...}}]

If the TRANSACTION option is not used, one of PowerHouse's default transactions is used. Default transactions are associated with every file, table or SQL statement. The default transactions are Query, Update and Consistency.

TRANSACTION

Specifies that the transaction is associated with the SQL statement.

transaction_name

Any valid PowerHouse name.

FOR CONSISTENCY

Determines that the SQL statement is associated with a particular transaction in Consistency model.

Limit: Only one transaction association can be specified.

FOR [CONCURRENCY] phase-option [,phase-option]...

Determines that the SQL statement is associated with a particular transaction or transactions in Concurrency model.

Limit: Up to three transaction associations can be specified.

phase-option

Specifies the screen phase with which the transaction is associated.

Phase option	Description
PROCESS	The phase in which you are entering, correcting, or changing data records on the screen.
QUERY	The phase in which data is retrieved from the database.
UPDATE	The phase in which data is updated.

For more information about transactions, see the *PowerHouse and Relational Databases* book.

INSERT INTO tablespec

Identifies the table where the new rows are to be added. The syntax for tablespec is:

[[server-name.]database-name.][owner-name.]table-name

If server-name is included in a Sybase tablespec, double quotation marks are required for the server-name and database-name, such as

```
"dbsvr01.accnt".manager.billings_tbl
```

For Oracle, the syntax is

[owner-name.]table-name[@database-linkname]

If the database-linkname is included, it is treated as part of the table-name, and double quotes are required. For example,

```
manager."billings_tbl@dblnk01"
```

Oracle synonyms may be used for table-names. For more information about how PowerHouse uses Oracle synonyms, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

[column-name[,column-name]...]

Identifies columns of the table. If no columns are listed, it is the equivalent of specifying all the columns in the order that they appear in the table.

query-expression

The query-expression is a query-specification or the union of two or more query-specifications.

If a query-expression is used, multiple rows may be inserted into the table.

VALUES ({sql-expression|NULL }[,{sql-expression|NULL}]...)

If a VALUES option is used, a single row is inserted into the table. The values are assigned to the columns by position. For example, the first value is inserted into the first column in the column list. Inserted rows will have a null value in each column that is not specified.

```
> INSERT INTO DBO.AUTHORS (FIRSTNAME, LASTNAME) &
>      VALUES ('SCOTT', 'CRAIG')
> INSERT INTO DBO.AUTHORS &
>      SELECT * FROM DBO.NEWEMPS &
>      WHERE TITLE = 'AUTHOR'
```

A value of NULL can also be specified for a column.

[RETURNING DBKEY INTO item]

Allows you to get the DBKEY of the inserted record so you can use it in a subsequent SQL UPDATE statement. DBKEY is available only if the underlying database supports it.

Limit: DBKEY is not supported for Sybase.

Discussion

The INSERT verb acts directly on a table or view in the database and is never generated by PowerHouse.

LET

Sets the value of an item equal to an expression.

Syntax

LET item = expression

item

Names the record item, temporary item, or predefined item to be assigned a value.

expression

Specifies the expression that provides the value.

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

Discussion

The LET verb assigns a value to a record item, a temporary item, or a predefined item. The value is the result of an expression. The expression must yield a value of the same type (CHARACTER, NUMERIC, or DATE) as the target item.

Note: For information about verb and procedure compatibility, see (p. 239).

Effects of the LET Verb on Record Status

If a LET verb changes the value of a record item and the screen is in ENTRYMODE, CORRECTMODE, or CHANGEMODE, the data record status of that data record is set to Changed. Data record status is also changed if there is no current mode (as in the INITIALIZE procedure).

The LET verb doesn't change the data record status in FINDMODE.

If a LET verb changes the value of an item that has an ITEM statement with the SUM option associated with it, the value obtained by subtracting the old value from the new value is summed.

Example

This course-enrollment screen doesn't allow more than a set maximum number of students to be booked into a course.

In the following example, the LET verb sets the value of PRESENTENROLL to equal PRESENTENROLL plus one. The counter PRESENTENROLL keeps track of the number of students enrolled in a course.

```
> SCREEN CRSENRL
> FILE BOOKING PRIMARY
> FILE COURSEMANAGE DESIGNER
> FILE COURSEMANAGE REFERENCE &
>   ALIAS COURSEPRESENT
>
> FIELD COURSENUMBER OF BOOKING REQUIRED NOCHANGE
> FIELD STARTDATE OF BOOKING REQUIRED NOCHANGE &
>   LOOKUP ON COURSEPRESENT VIA COURSEKEY &
>   USING NCONVERT(COURSENUMBER) + &
>   NCONVERT(STARTDATE)
> FIELD STUDENTNAME OF BOOKING REQUIRED
> FIELD COURSEKEY OF BOOKING SILENT &
>       LOOKUP NOTON BOOKING
>
> PROCEDURE PREUPDATE
>   BEGIN
>     GET COURSEMANAGE VIA COURSEKEY &
>       USING NCONVERT(COURSENUMBER) +   &
```

```
>             NCONVERT(STARTDATE)
>     IF PRESENTENROLL >= COURSEMAXIMUM
>       THEN ERROR = "This course is full." + &
>         "Please try another date."
>       ELSE LET PRESENTENROLL = PRESENTENROLL + 1
>     END
>
> PROCEDURE UPDATE
>   BEGIN
>     PUT BOOKING
>     PUT COURSEMANAGE
>     END
>
> BUILD
```

LOCK

Locks a file, an IMAGE database or dataset, an ALLBASE/SQL table or an Oracle table.

Syntax

**LOCK [FILE] {record-structure [record-options] |
tablename [table-options]}
[,[FILE]{record-structure [record-options] |
tablename [table-options]}]...**

record-structure[record-options]

Names the file to be locked.

The record-options are:

- BASE
- FILE
- RECORD

Defaults: BASE for IMAGE datasets; FILE for MPE and KSAM files (**MPE/iX**); FILE (**OpenVMS, UNIX, Windows**).

BASE (MPE/iX)

Tells QUICK to lock IMAGE datasets at the database level.

FILE

Tells QUICK to lock files or datasets at the file level.

MPE/iX:	IMAGE datasets are locked at the dataset level. MPE and KSAM files are always locked at the file level.
OpenVMS:	Applies an exclusive LMS lock to the named file.
UNIX:	UNIX IO files are always locked at the file level.

RECORD

Tells QUICK to lock at the record level.

MPE/iX:	Locks IMAGE datasets at the record level. The LOCK RECORD option affects only IMAGE files. If the LOCK RECORD option is specified for other file systems, QUICK displays an error message.
OpenVMS:	Applies an LMS concurrent write lock to the named file and uses RMS locking to lock records as needed. Applies to all non-relational file types.
UNIX:	Record-level locking is supported for C-ISAM files on all UNIX platforms and Net-ISAM files on Sun SPARC.
Windows:	Record-level locking is supported for DISAM files.

tablename[table-options]

The name of a table defined in an ALLBASE/SQL or Oracle relational database. The tablename must have been previously referenced on a FILE or CURSOR statement.

The table-options are:

- EXCLUSIVE
- PROTECTED
- SHARED

EXCLUSIVE|PROTECTED|SHARED

An EXCLUSIVE lock prevents other transactions from reading or writing to the table. A PROTECTED lock allows other transactions to read but not change the table. A SHARED lock allows other transactions to read or write to the table.

Default: If the table is READ ONLY, the lock type is PROTECTED; otherwise, the lock type is EXCLUSIVE.

Discussion

For information about verb and procedure compatibility, see (p. 239).

Locking Tables

With the exception of ALLBASE/SQL and Oracle, relational tables cannot be locked using the LOCK verb. The LOCK verb is ignored for all other supported databases. Relational tables are protected by the database software against conflicting updates. For more information about rollback and error processing when using relational record-structures, see Chapter 2, "Relational Support in QDESIGN", in the *PowerHouse and Relational Databases* book.

The UNLOCK verb is ignored for all database products. Locks are released when the transaction ends.

Locking Files

To maintain data integrity, QUICK automatically locks files before performing a write or update. Indexed files are locked before performing a write, update, or read.

Unless otherwise specified, QUICK opens all files in shared mode and applies default locking around or upon the GET and PUT verbs.

The FILE statement exclusivity options can be employed to override this default and make locking unnecessary.

File-level locks are only released when

- a corresponding UNLOCK verb is executed
- QUICK is ready to accept input from the terminal
- QUICK is ready to send output to the terminal
- QUICK links to a lower-level screen
- QUICK exits from the current screen

File-level locks are left in place for external routine calls and command processing. The INFORMATION and WARNING verbs with the NOW option display messages immediately and do not cause file-level locks to be released. However, file-level locks are released if the RESPONSE option is used, because that will request user input. As well, file-level locks are released if the message is greater than the screen width, because the user is prompted to continue even if the RESPONSE option is not specified.

File-level Locks: Exclusive Locks (OpenVMS)

The FILE options of the LOCK verb uses LMS to apply an exclusive lock to the named file. An exclusive lock grants write access to the file and prevents the file from being shared by any other readers or writers.

If a PUT verb is issued for a file locked with the LOCK option, RMS record-level locks are applied, but the exclusive file-level lock is not released.

Controlling Locking with the SCREEN Statement

When building a screen with QDESIGN, the designer can take control of file locking by using the LOCK option of the SCREEN statement. If the LOCK option is specified, QDESIGN generates default LOCK and UNLOCK verbs in the UPDATE procedure.

Using the default locking strategy provides more concurrency than using the SCREEN statement LOCK option to lock files.

OpenVMS, UNIX, Windows: When you use default locking, the PUT verb locks just one data record at a time, updates it, and then unlocks it.

MPE/iX: When you use default locking, the PUT verb locks the file, does the update and unlocks the file.

By using the LOCK option, you can specify that all files that can be updated are locked for the entire UPDATE procedure, thereby ensuring the reliability of rollback during the UPDATE procedure.

For more information, see (p. 185).

Using the LOCK Verb

Explicit locks are required only to improve throughput or to meet user-specific requirements that are not satisfied by QUICK's default locking strategy. Use the LOCK verb to specify options that are consistent with your particular needs.

MPE/iX: To lock more than one file concurrently, QUICK uses Multiple RIN Capability. For more information, refer to the Hewlett-Packard documentation for your operating system.

OpenVMS: QUICK engages the VMS Lock Management Services (LMS) for file-level locking and Record Management Services (RMS) for record-level locking.

Deadlock Protection (MPE/iX, UNIX, Windows)

For information about deadlock protection in OpenVMS, see "Deadlock Protection (OpenVMS)" (p. 444).

A "deadly embrace" can occur if two or more processes attempt to lock the same files using unconditional locking. To avoid this, QUICK uses special logic when locking multiple files:

1. QUICK attempts to lock the first file in the statement with a conditional lock.
2. If another process has locked the file, QUICK uses the same strategy as default locking, retrying the lock every two seconds for a total of 16 lock attempts. However, once the first file is locked, QUICK does not retry locks for subsequent files in the same LOCK statement.
3. If a subsequent lock is not immediately successful, all files locked in the current LOCK statement are unlocked. QUICK then starts the locking process again from the first lock, taking into consideration the total number of lock attempts allowed on the first file.

QUICK also uses a similar logic to avoid deadly embraces if the same file is opened twice. When QUICK issues a lock on one open, all other opens are marked as locked. For example, if a file is opened once for read-only access, again for update access, and then the designer locks through the original read-only open, the update open is marked locked.

The designer can control locking options using QKGO file parameters. The file parameter LOCK ATTEMPTS controls the maximum number of lock attempts (the default is 16). The length of time between lock attempts (the default is two seconds) can be controlled with LOCK RETRY INTERVAL. Whether the first lock attempt is conditional or unconditional is controlled using the LOCK UNCONDITIONAL parameter. If LOCK UNCONDITIONAL is enabled, the first lock is unconditional only if no other files are locked by this QUICK process. For more information, see (p. 255).

Deadlock Protection (OpenVMS)

For information about deadlock protection in MPE/iX, UNIX, and Windows, see "Deadlock Protection (MPE/iX, UNIX, Windows)" (p. 444).

A deadly embrace can occur when two or more processes attempt to lock the same set of files. QUICK supplements the deadlock protection provided by LMS to ensure that deadly embraces do not occur.

If you attempt to lock a file via the GET, PUT, or LOCK verbs, or the LOCK option of the SCREEN statement, and that file has already been locked by another process, the lock request is queued and the message "The file is busy. Please wait..." is issued. Use the QKGO parameter LOCK MESSAGE WAIT to alter the time delay (in seconds) between a lock request for file access and the display of the message "The file is busy. Please wait...". This message is displayed until the maximum lock request specification has been exhausted. At this time, if the lock request has not been granted, the message "Unable to complete the requested action at this time" is displayed.

The maximum number of seconds that a lock request is queued before an error message is issued can be set by the screen designer with the QKGO parameter LOCK REQUEST WAIT. The default is 30 seconds and the maximum wait is 255 seconds.

For more information on QKGO parameters, see (p. 255).

In addition, QUICK uses special logic to avoid the deadlock that can occur if the same file is opened twice with different file names (as may be the case with files that have been assigned aliases, for example). When QUICK issues a lock on one open, all other opens are marked as locked.

System Locking Compatibility (OpenVMS)

The locking strategies available in QUICK are consistent with those in the other PowerHouse components. However, because LMS locking is cooperative, there is no guaranteed protection against non-PowerHouse 4GL program access. To enable other programs to use the same LMS locking strategies as PowerHouse 4GL, a documented set of routines is provided with the (OpenVMS server) PowerHouse 4GL installation package in the PH_DOC_LOCATION directory.

Record Level Locking and Unlocking

The following conditions cause all records to be unlocked by QUICK:

- execution of an UNLOCK RECORD verb for a given file
- start of a new entry sequence
- start of a new find sequence
- QUICK exit from the current screen
- completion of a PUT verb if using default locking (**UNIX, Windows**)

There is no way to unlock any one record. Any condition that causes a record to be unlocked will also cause all records locked in the same file by the calling process to be unlocked. It is therefore possible that when a lock on a particular screen is released, locks on another screen associated with the same process may also be released. This is also true of the UNLOCK when QUICK performs checksum verification during the execution of any PUT verb.

To preserve data integrity, QUICK always performs the reread/ checksum calculation before updating the record; that is, if you try to update a "locked" record that has since been unlocked by an unrelated UNLOCK, the usual update lock is applied, the record is reread, and the checksum is calculated as if the record had not been locked.

Since QUICK applies locks during the reread and checksum calculation, when QUICK terminates the lock at the end of the verification, all records locked in that file by that process will be unlocked.

QDESIGN issues a message warning you about this issue when the UNLOCK verb is used with the RECORD option.

OpenVMS: The RECORD option of the LOCK verb causes LMS to apply a concurrent write lock to the named file.

QUICK maintains a rollback buffer where it stores images of updated records, record locks, and unlocks. Since record locks and unlocks take up extra space, you must increase the amount of space reserved for rollback buffers. Failure to do so may result in an error message indicating that there is insufficient page space and that the lock is being abandoned. To increase the reserved space, use the QKGO parameter Rollback Buffer.

All record occurrences for a repeating file can be locked by including a LOCK verb with the RECORD option in the FOR construct that accepts or retrieves data for the file. If the LOCK verb is outside the FOR construct, only the first occurrence will be locked.

OpenVMS: If multiple LOCK verbs lock the same file, but use different options, the result is an exclusive file-level lock. If the file-level lock follows the record-level lock, the concurrent write file lock is converted to an exclusive lock. If the reverse is the case, the exclusive lock is left in place.

Record Level Locking with IMAGE (MPE/iX)

Specifying Lock Items

PowerHouse determines internally which item to use as the lock item, using the following rules:

- If a primary index exists, the item associated to that index is used.
- If no primary index exists but there is an alternating/repeating index, the first alternating/repeating index is used.

As a result, you can determine which item is to be used as the lock item only through the data dictionary specification for the database.

This makes it particularly important to determine an effective locking strategy when designing an application, in order to minimize potential problems. For example, it is possible to lock nonexistent records by specifying a value that does not exist in the dataset. If that value is subsequently added to the dataset, the lock is automatically applied, resulting in confusion for other users.

Read chains can be handled by record level locking but you must take care to lock the entire chain, otherwise a broken chain can occur.

Record level locking gives you exclusive write access to records in the dataset that are associated with the locked item. The value for the item in the record that is open at the time of the lock request is the value used.

If there are multiple occurrences of the same item value in a dataset, all records containing that value are locked. For example, if the lock item value is the last name "Smith", the records for all the "Smiths" in that dataset are locked. Records in other datasets are not affected.

Other users can access the locked records, but cannot update them until the lock is removed.

Limitations

Some database operations that involve master set records and critical items require dataset locking as the minimum level. As a result, record level locking cannot be used with the following operations:

- addition or deletion of master set records
- updates to critical items associated with a master set record (key items)
- updates to critical items associated with a detail set record (key items)

Using Multiple LOCK Verbs

The results of multiple LOCK verbs are combined. For example,

```
> LOCK A FILE, C FILE
> LOCK B FILE, D FILE
```

results in files A, B, C, and D being locked. You can get the same results with the following verbs:

```
> LOCK A FILE, C FILE
> LOCK A FILE, B FILE, C FILE, D FILE
```

In the last example, the locks on files A and C in the second LOCK verb are ignored and the action is the equivalent of simply locking file B and D. If file B or D cannot be locked, the locks on file A and C are not removed, as they would have been if files A and C had been previously locked.

If a lock attempt fails on any file or record in a set, all other successful locks in the set are unlocked, unless they were successfully locked by a previous LOCK verb.

Using Multiple Lock Verbs with IMAGE (MPE/iX)

If multiple LOCK verbs lock datasets and records in the same IMAGE database, QUICK ensures that the correct level (BASE, FILE or RECORD) of lock is applied. If locks that affect the same dataset are requested at different levels, only the highest level lock is applied, where BASE is the highest level and RECORD is the lowest level.

If a higher level lock already exists when a lower level lock is requested, the lower level lock has no effect. For example, A, B and C are datasets in the same database and are locked as follows:

```
> LOCK A BASE
> LOCK B FILE
> LOCK C RECORD
```

The LOCK verbs for B and C will both be ignored because the entire database was already locked by the first LOCK verb.

If a lower level lock exists when a higher level lock is requested, QUICK must release the lower level lock before applying the higher level lock. Here is an example:

```
> LOCK B RECORD
> LOCK B FILE
> LOCK A BASE
```

The second LOCK verb in this example forces QUICK to release the existing record-level locks before it can apply a file-level lock to the same dataset. The third LOCK statement forces QUICK to release the file-level lock on B before it can lock the entire database.

File-level and record-level locks can both be used, if they are on different datasets. For example, C and B are locked as follows:

```
> LOCK C RECORD
> LOCK B FILE
```

In this example, the record-level locks on C are compatible with the file-level lock on B. QUICK does not have to release the record-level locks before applying the file-level lock because the locks are on different datasets.

It is not possible to unlock individual files or records in an IMAGE database. If anything is unlocked, then all locks that are held through the same physical open of the database are released at the same time. This can have unexpected results if file-level, record-level, or a combination of file-and record-level locks exist on more than one dataset in the database when the unlock is done.

The unlock could be the result of any of the following:

- an UNLOCK verb,
- upgrading to a higher lock level as outlined above,
- hitting one of the processing points where QUICK releases all file or record level locks
- QUICK releasing the lock that it automatically applies around the PUT if a record or file is not already locked when a PUT verb is executed,.

For more details, see the UNLOCK verb.

Example

```
> SCREEN STAFF LOCK FILE FOR UPDATE
> FILE EMPLOYEES PRIMARY
> FILE SKILLS DELETE
Item EMPLOYEE initialized (fixed) to EMPLOYEE OF EMPLOYEES
> FILE POSITIONS REFERENCE
> FILE BRANCHES REFERENCE
.
.
.
> BUILD LIST
>
> PROCEDURE UPDATE
> BEGIN
> LOCK EMPLOYEES FILE, SKILLS FILE
> PUT SKILLS
> PUT EMPLOYEES
> UNLOCK ALL
> END
```

MEMOLOG (MPE/iX)

Writes a message to an IMAGE log file.

Syntax

MEMOLOG filespec string-expression

filespec

Names a file (that is a dataset in an IMAGE database) for which the message is logged.

string-expression

Defines a string expression.

Discussion

The MEMOLOG verb places a message into an IMAGE log file associated with an IMAGE database using a DBMEMO call. The verb is ignored for other types of files. The name of the file in the desired database can be used.

Note: For information about verb and procedure compatibility, see (p. 239).

NULL

Performs a null action.

Syntax

NULL

Discussion

The NULL verb is typically used in the ELSE portion of a nested IF control structure or to perform a NULL procedure.

Example

The edit procedure for the transact screen uses a nested if control structure to evaluate the status code returned from the external procedure edittrans. The addition of the null verb helps to make it clear to the user that the entered transaction number passed the edit check.

```
> SCREEN TRANSACT
>
> TEMPORARY STATUSCODE NUMBER*4
>
> FILE TRANSACTIONS
>
> SKIP TO LINE 4
> HILITE TITLE INVERSE
> TITLE "Transaction Screen" CENTERED
> DRAW FROM 3,15 TO 5,65
> SKIP 2
>
> GENERATE NOLIST
.
.
.
> PROCEDURE EDIT TRANSNO
>  BEGIN
>   IF STATUSCODE = 0
>    THEN NULL  ; PASSED EDIT CHECK
>    ELSE BEGIN
>     IF STATUSCODE < 0
>     THEN ERROR = "Entered transaction number didn't " + &
>          "pass edit check."
>   ELSE WARNING = "Entered transaction number refers to " + &
>      "out-of-date stock."
>     END
>    END
> BUILD
```

[SQL] OPEN

Opens a cursor and gets the result rows ready to be accessed with subsequent SQL FETCH statements.

Syntax

[SQL] OPEN cursor-name [sql-substitution...]

cursor-name

The name of a cursor defined by the PowerHouse [SQL] DECLARE CURSOR statement.

sql-substitution...

The syntax for an sql-substitution is:
substitution-variable (text)

Discussion

If the cursor is already open, it will be closed and re-opened.

The CURSOROPEN predefined condition is used in the generated FIND procedure to check if an OPEN needs to be executed before data is retrieved from the cursor.

PERFORM APPEND

Executes the APPEND procedure.

Syntax

PERFORM APPEND

Discussion

QDESIGN automatically generates the PERFORM APPEND verb in an ENTRY procedure when repeating PRIMARY or DETAIL files are declared on a screen. Similarly, for PANEL screens, QDESIGN generates a PERFORM APPEND within a FOR MISSING loop in the default MODIFY procedure.

Note: For information about verb and procedure compatibility, see (p. 239).

Example

This example invoice screen creates detail data records for the INVOICES file. In this example, the PERFORM APPEND verb executes the APPEND procedure to accept values for the repeating DETAIL file after values are entered for the PRIMARY file.

```
> SCREEN INVINFO
>
> FILE INVOICES PRIMARY
> FILE INVOICEDETAIL DETAIL OCCURS 10
Item INVOICENO initialized (fixed) to INVOICENO OF
INVOICES.
>   ACCESS VIA INVOICENO
>
> FIELD INVOICENO OF INVOICES REQUIRED NOCHANGE &
>   LOOKUP NOTON INVOICES
> ALIGN (1,4,21) (,31,45)
> FIELD CUSTNO OF INVOICES
> FIELD INVOICEDATE OF INVOICES
> SKIP 2
> CLUSTER OCCURS WITH INVOICEDETAIL FOR 2,40
>   ALIGN (1,4,20) (,,28)
>   FIELD PRODNO OF INVOICEDETAIL
>   FIELD QTYSHIPPED OF INVOICEDETAIL
> CLUSTER
>
> PROCEDURE APPEND
>   BEGIN
>     ACCEPT PRODNO OF INVOICEDETAIL
>     ACCEPT QTYSHIPPED OF INVOICEDETAIL
>     END
>
> PROCEDURE ENTRY
>   BEGIN
>     ACCEPT INVOICENO OF INVOICES
>     ACCEPT CUSTNO OF INVOICES
>     ACCEPT INVOICEDATE OF INVOICES
>     FOR INVOICEDETAIL
>       BEGIN
>         PERFORM APPEND
>         END
>     END
> BUILD
```

PROMPT

Prompts for and displays a value in a field.

Syntax

PROMPT field [INTO item] [REQUIRED]

field

Names the field where the user is prompted for input.

INTO item

Instructs QUICK to move the value to the named item rather than the item normally associated with the field. The item may be a record item or temporary item.

REQUIRED

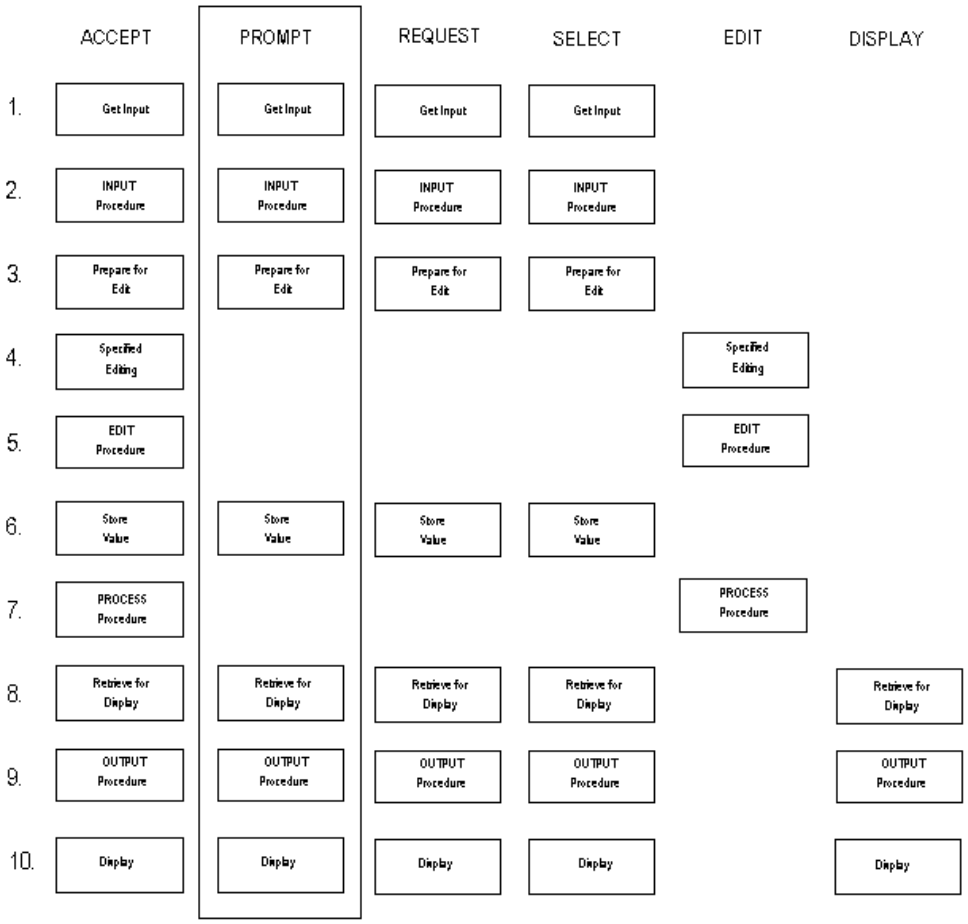
Declares that a null response from the screen user isn't accepted. A valid entry must be supplied.

Discussion

The PROMPT verb initiates the activities that prompt the screen user at the named field, perform size and type checking on the screen user's response, and redisplay the value in the field with any format options applied.

Note: For information about verb and procedure compatibility, see (p. 239).

The following figure illustrates the steps that are performed by the PROMPT verb, and contrasts these steps to similar steps performed by other field processing verbs:



The PROMPT verb follows the same steps as the ACCEPT verb, except that QUICK skips Steps 4, 5, and 7 (editing and processing). For more information about the steps that are initiated by the PROMPT verb, see (p. 364).

The PROMPT verb ignores the DEFAULT, DUPLICATE, and REQUIRED options of the FIELD statement, but does recognize the user-entered Duplicate command (_). However, the PROMPT verb doesn't save the entered value in the duplicate buffer.

Inclusion of PROMPT overrides the following FIELD statement options: DISPLAY, FIXED, IF, OMIT, and NOENTRY.

Example

This order-management screen prompts the user for confirmation before deleting data records from a file.

The PROMPT verb prompts for confirmation in the CONFIRM field before deleting any data records. The REQUIRED option prevents the QUICK screen user from making a null response.

```
> SCREEN ORDERMGT
>
> TEMPORARY CONFIRM CHARACTER*1
>
> FILE CUSTOMERS PRIMARY
> FILE ORDERS DESIGNER
> FILE ORDERS ALIAS ORDERERASE DELETE
Item ORD-NUM initialized (fixed) to ORD-NUM OF CUSTOMERS.
> FIELD CUSTNO OF CUSTOMERS REQUIRED NOCHANGE &
>       LOOKUP NOTON CUSTOMERS
> FIELD CUSTNAME OF CUSTOMERS
> FIELD CONFIRM NOID NOLABEL UPSHIFT
> PROCEDURE DELETE
>   BEGIN
>     GET ORDERS VIA ORDNUM &
>       USING ORDNUM OF CUSTOMERS
>     IF ACCESSOK
>       THEN BEGIN
>         WARNING "Records on file. Do you wish to delete?"
>         PROMPT CONFIRM REQUIRED
>         IF CONFIRM = "Y"
>           THEN BEGIN
>             DELETE CUSTOMERS
>             DELETE ORDERERASE
>             END
>         END
>       ELSE DELETE CUSTOMERS
>     END
>
> BUILD
```

Although you must exercise caution when using a PROMPT verb in a DELETE procedure, in this case, QUICK prompts into a temporary item to confirm the deletion only; the data the user enters in response to the prompt doesn't have to be saved.

For more examples about how to use the PROMPT verb, see (p. 319).

PUSH

Places one or more QUICK commands on a Pending Screen Input Buffer (PSIB).

Syntax

PUSH conditional-command-list

conditional-command-list

Specifies what command(s) the PUSH verb executes and, optionally, under what conditions.

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. The general form of a command list is:

command [, command]...

Limit: Only Action commands and Action and Data commands can be specified by the PUSH verb, not Data commands.

For a list of the available commands, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

condition

A condition is a logical test that has the general form:

[NOT] condition [AND|OR [NOT] condition]...

For more information about conditions or conditional command lists, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Discussion

The PUSH verb pushes a conditional command list onto the Pending Screen Input Buffer (PSIB). Although the PSIB has a 'LIFO' (Last In=First Out) configuration, pushed commands are executed in the order you specify them in the conditional command list.

You can specify several PUSH verbs in a row for execution under procedural control, as in

```
> PUSH LAST RECORD
> PUSH NEXT DATA
```

Be aware, however, that this is different from entering

```
> PUSH LAST RECORD, NEXT DATA
```

In the first example, LAST RECORD is put on the PSIB first, followed by NEXT DATA. Due to the PSIB's LIFO configuration, the commands are removed for processing in the order NEXT DATA, LAST RECORD. This is the opposite of the second example, where the commands are processed in the order in which they're specified, that is, LAST RECORD, NEXT DATA.

For more information, see (p. 49).

Note: For information about verb and procedure compatibility, see (p. 239).

PUT

Updates the data record.

Syntax

PUT [DELETED|NEW|NOTDELETED]
record-structure|cursor-reference
[AT numeric-expression] [RESET]

DELETED|NEW|NOTDELETED

Specifies the type of data records to update.

DELETED

Specifies that only data records marked for deletion are processed by the PUT verb.

NEW

Specifies that only new data records are processed by the PUT verb.

NOTDELETED

Specifies that data records not marked for deletion are processed by the PUT verb.

record-structure

Names the record-structure to be updated.

cursor-reference

A cursor-name or table-name named in a CURSOR statement.

AT numeric-expression

Stores the data record at the data record number indicated by the numeric expression.

Limit: Applies only to record-structures in direct or relative files, and only if the data record status is New.

RESET

Specifies that the record buffers and status are reset after the update. The buffers are reset to default and initial values, and the data record status is reset to New, Unchanged, Undeleted.

Discussion

The PUT verb can refer to a data structure referenced in a FILE or CURSOR statement. A PUT verb updates one data record on the named data-structure. What the PUT verb does in each circumstance depends on the status of the data record in the screen's record buffer.

The PUT verb for a CURSOR only issues SQL DML statements for the first table in the query-specification and its associated columns. A column can be omitted from being updated by giving it an alias name with the AS option.

The following table lists the effects of the PUT verb under different circumstances.

Data record source	Record status	Effect of PUT verb
Brought into record buffer from file	Unchanged or empty record buffer	No effect

Data record source	Record status	Effect of PUT verb
	Changed or Deleted	Updates or deletes the data record in the file to correspond to the record in the record buffer
Brought into record buffer from a DELETE file	Marked for deletion	Deletes all data records that match record access criteria

The PUT verb is recommended for use only in BACKOUT and UPDATE procedures, or procedures using the RECOVERABLE options (available for the PREUPDATE, POSTUPDATE, and INITIALIZE procedures.) For more information about verb and procedure compatibility, see (p. 239).

Details of PUT Verb Processing

Processing of the PUT verb for any data record relies on

- the record status: Old|New, Changed|Unchanged, Deleted|Undeleted
- a NEED option on the FILE statement
- the INITIAL FIXED or FINAL options of ITEM statements

The following table describes the standard sequence of activities when QUICK processes a PUT verb.

Step	Under what conditions	Activity performed
1. Evaluate the NEED option.	NEED option specified on FILE statement that applies to the file	Treat the data record as changed for the purpose of update.
2. Compute the FINAL values.	OLD, UNDELETED, or NEW, CHANGED, UNDELETED	Assign the ITEM INITIAL FIXED and FINAL values to the data record in the order specified. (This may change the record status.)
3. Check for a conflicting update.	OLD, CHANGED	Reread the existing data record. Compare the current value to the original value of the data record.
4. Update.	NEW, CHANGED, UNDELETED (indexed file)	Add the new data record to the file.
	NEW, CHANGED, UNDELETED (SEQUENTIAL file)	Place the data record at the end of the file.
	NEW, CHANGED, UNDELETED (DIRECT file)	Evaluate the AT option and place the data record at the indicated location in the file. If there is no AT option, place the data record at the end of the file.
	OLD, CHANGED, UNDELETED (indexed or DIRECT file)	Replace the old data record in the file.

Step	Under what conditions	Activity performed
	OLD, CHANGED, UNDELETED (SEQUENTIAL file)	Error condition.
	OLD, CHANGED, DELETED (indexed file)	Delete the data record from the file.
	OLD, CHANGED, DELETED (DIRECT or SEQUENTIAL file)	Error condition.
5. Reset buffers	CHANGED, UNDELETED	Update the rollback buffers.
6. Reset record status	NEW, CHANGED, UNDELETED	Set the data record status to OLD, UNCHANGED, UNDELETED.
	CHANGED	Set the data record status to UNCHANGED.
	RESET option of PUT verb	Set the data record status to NEW, UNCHANGED, UNDELETED, and initialize the buffers to default and initial values.
7. Back out balancing	DELETED	Reverse the COUNT or SUM operations from this data record (applies only if the file marked as deleted is a DELETE file). For other file types, the COUNT and SUM operations are backed out by the DELETE verb.

Notes on the PUT Verb

Any errors detected by QUICK in processing a PUT verb cause further processing to stop.

All PUT verbs for DELETE record-structures cause repeated retrievals and deletions of all data records that meet the access conditions for the file.

QDESIGN doesn't include PUT verbs in the UPDATE procedure of AUDIT, DESIGNER, or REFERENCE files. Data records in AUDIT files are automatically written in conjunction with their associated file. Data records in REFERENCE files are opened for read only access.

If the file processed by a PUT verb has an AUDIT record-structure associated with it, ITEM INITIAL FIXED and FINAL option values for the AUDIT file are also calculated in Step 2. In addition, Steps 3 through 6 are performed for the AUDIT file, after they have been completed for the associated file.

Non-Relational Rollback

PUT verbs used in recoverable procedures can take advantage of PowerHouse's full rollback for non-relational files. If any error conditions are detected in the middle of a multiple record update, the rollback facility restores the files to the state that they were in before the update began. This applies to all PUT verbs in recoverable procedures, PUTs to designer files and PUTs executed in an INTERNAL procedure while a recoverable procedure is active.

The UPDATE procedure is recoverable by default. The INITIALIZE, POSTUPDATE and PREUPDATE procedures can be set to recoverable by using the RECOVERABLE option.

If you add the RECOVERABLE option to existing INITIALIZE, PREUPDATE and POSTUPDATE procedures, any BACKOUT procedures that you have coded to handle rollback of PUT verbs in those procedures must be reviewed to ensure that a "double rollback" is not performed.

Outside recoverable procedures, PUT verbs to non-relational files are never rolled back.

If the RECOVERABLE option is used, the changed behavior of PUT verbs in each procedure is as described in this manual. If the RECOVERABLE option is not used, non-relational rollback of PUT verbs is unchanged.

PUT Verbs and Relational Tables

With relational tables, PUT verbs are handled by relational transactions. The way a program behaves depends upon what transaction model is used. For more information, see the *PowerHouse and Relational Databases* book.

Implications of PUT Verbs in Designer-written Procedures

A PUT verb in a designer-written procedure can result in data being placed in the file prior to the use of any of the Update commands. Data integrity of non-relational files can thus be compromised if a QUICK screen user backs out of a screen without updating. In such cases, there is no automatic rollback procedure to cancel the effects of PUT verbs outside of recoverable procedures. If PUT verbs are used in procedures other than the UPDATE procedure or procedures with the RECOVERABLE option, then you can write a BACKOUT procedure that nullifies the effects of the PUT verbs. Designer-written BACKOUT procedures can prevent undesired updates, such as when a QUICK screen user backs out of a screen without entering an Update command.

Example

QDESIGN generates a PUT verb in the UPDATE procedure for most file types used in a screen design. For example,

```
> SCREEN EMPSKILL
> TEMP COUNTER NUM*3 INITIAL 0
> TEMP CONFIRM CHAR*1
> FILE EMPLOYEE PRIM
> FILE SKILLS DETAIL OCCURS 10
.
.
.
> PROCEDURE UPDATE
>      BEGIN
>          PUT EMPLOYEE
>          FOR SKILLS
>             BEGIN
>                PUT SKILLS
>          END
>      END
.
.
.
```

REFERENCE files do not generate PUT verbs, as in:

```
> SCREEN EMPLABEL
> FILE EMPLOYEE PRIMARY
> FILE SKILLS DETAIL OCCURS 5
> FILE LABELS REFERENCE
> ACCESS VIA LASTNAME USING LASTNAME OF EMPLOYEE
.
.
.
> PROCEDURE UPDATE
>   BEGIN
>      PUT EMPLOYEE
>      FOR SKILLS
>        BEGIN
>          PUT SKILLS
>          END
>      END
```

REFRESH

Clears and rewrites (refreshes) an area of terminal memory.

Syntax

REFRESH ALL|SCREEN|[LINES] n [TO m]

ALL

Clears and rewrites the entire terminal memory.

SCREEN

Clears and rewrites the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears and rewrites the area between and including lines n to m, numbering from the first line of terminal memory. LINE n by itself refreshes line n only.

Discussion

The REFRESH verb only works if QUICK is run with the **restore = lines** program parameter. This program parameter changes the default behavior of screen refreshing.

The REFRESH verb instructs QUICK to clear and rewrite all or part of the terminal memory. The options allow the designer to restrict the portion that is rewritten. One option must be given. The effects of the REFRESH verb are not apparent until QUICK is ready to prompt the user, and may be negated by other REFRESH or CLEAR verbs or options.

Note: For information about verb and procedure compatibility, see (p. 239).

REQUEST

Prompts for a value required for record retrieval.

Syntax

REQUEST field

field

Names the field where the user is prompted for input.

Discussion

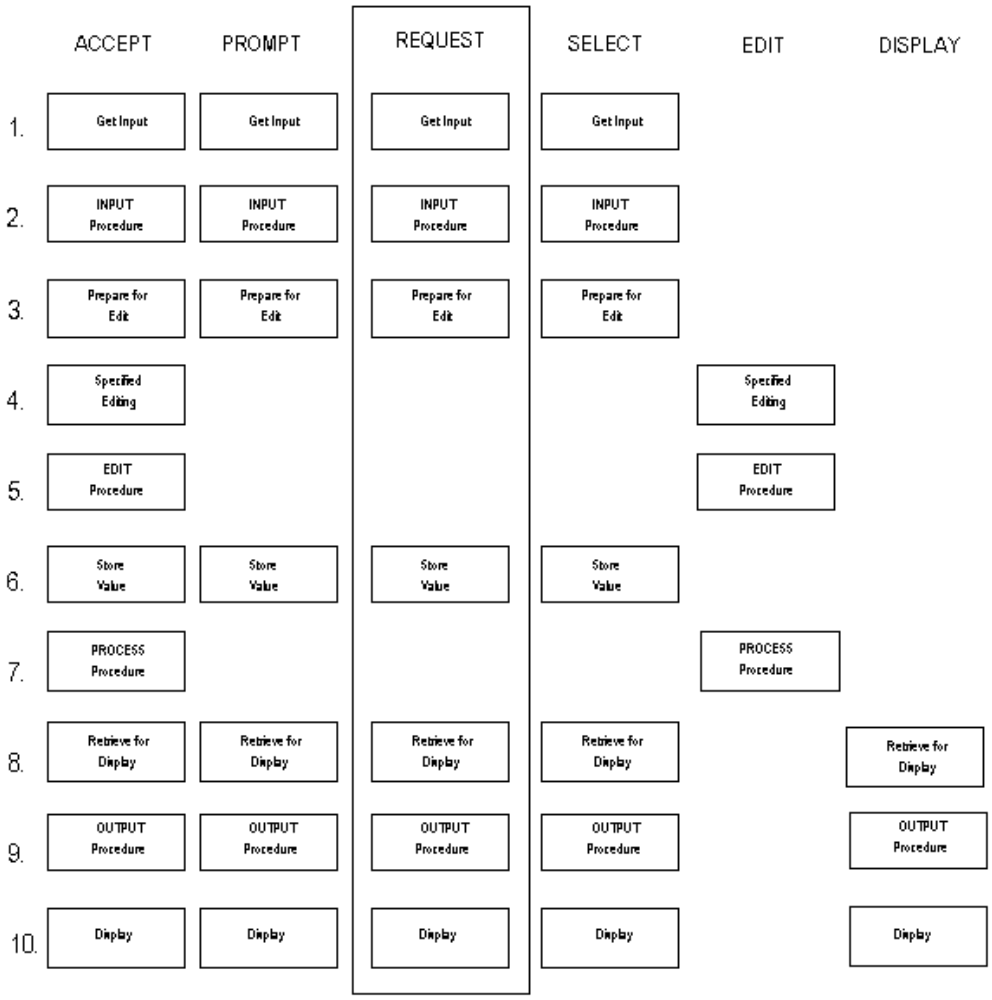
The REQUEST verb relates to processing required by the PATH procedure. QDESIGN generates one REQUEST verb in the PATH procedure for each segment in each index in the record-structure of the PRIMARY file of the screen design.

The accepted value is stored in the request buffer. Until the next Entry mode or Find mode initialization, the stored value is used to initialize the item. Once a request has been made for a given field, additional requests for that field return the value first accepted.

Each time record initialization takes place, the requested value is placed into the record buffer, as is done for an ITEM statement with an INITIAL option. Only the first request for any field goes to the user for input. Subsequent requests set both the value and the PROMPTOK predefined condition based on the response given to the first request. The request buffer is cleared during Entry and Find mode initialization.

Note: For information about verb and procedure compatibility, see (p. 239).

The following figure illustrates the steps that are initiated by the REQUEST verb, and contrasts these steps with other field processing verbs:



The REQUEST verb follows the same steps as the ACCEPT verb except that Steps 4, 5, and 7 (editing and processing) are bypassed. For details about the steps listed in the preceding figure, see (p. 364).

The REQUEST verb ignores the DUPLICATE, REQUIRED, and DEFAULT field options but does recognize the user-entered Duplicate command `(_)`. However, the REQUEST verb does not save the entered value in the duplicate buffer.

Inclusion of REQUEST overrides the following FIELD statement options: DISPLAY, FIXED, IF, NOENTRY, and OMIT.

Differences Between the REQUEST and PROMPT Verbs

Both the REQUEST and PROMPT verb prompt the QUICK screen user for data in a field. However, the REQUEST verb places the response entered in the request buffer for the current screen. The values in the request buffer are used to retrieve data based on the PATH and FIND procedures. In contrast, the PROMPT verb simply prompts for a value and stores the entered response into the field that's referenced by the PROMPT verb.

Always use the REQUEST verb rather than the PROMPT verb when you prompt for segment values to be used in record retrieval.

Example

QDESIGN generates one REQUEST verb in the PATH procedure for each segment in each index in the record-structure of the PRIMARY file of the screen design.

```
> SCREEN EMPSKILL
> TEMP COUNTER NUM*3 INITIAL 0
```

```
> TEMP CONFIRM CHAR*1
> FILE EMPLOYEE PRIM
> FILE SKILLS DETAIL OCCURS 10
> FIELD EMPLOYEE OF EMPLOYEE REQUIRED NOCHANGE &
>  LOOKUP NOTON EMPLOYEE
> FIELD LASTNAME OF EMPLOYEE REQUIRED NOCHANGE
> CLUSTER OCCURS WITH SKILLS
> FIELD SKILL OF SKILLS
> CLUSTER
> SKIP TO LINE 23
> ALIGN (,,1)
> FIELD CONFIRM NOID NOLABEL UPSHIFT
.
.
.
> PROCEDURE PATH
>     BEGIN
>         REQUEST EMPLOYEE OF EMPLOYEE
>         IF PROMPTOK
>             THEN LET PATH = 1
>             IF PATH = 0
>                 THEN BEGIN
>                     REQUEST LASTNAME OF EMPLOYEE
>                     IF PROMPTOK
>                       THEN LET PATH = 2
>                 END
>                 IF PATH = 0
>                     THEN BEGIN
>                      LET PATH = 3
>                 END

>     END
```

For more examples of how to use the REQUEST verb, see (p. 335).

RETURN

Exits a screen.

Syntax

RETURN

Discussion

The RETURN verb forces an immediate return to the invoking screen.

If the current screen is the initial screen specified in an active QKGO file, the RETURN verb terminates QUICK. If no initial screen is specified and the user is on the highest-level screen, the RETURN verb returns the user to the Screen ID prompt. If there is changed data on the screen, the RETURN verb executes the BACKOUT procedure.

Note: For information about verb and procedure compatibility, see (p. 239).

Example

The following example includes a numbered DESIGNER procedure that contains a RETURN verb. Although QUICK screen users can always return to a higher-level screen by pressing ^, you can use the following method to display a menu option on your screens for returning to higher-level screens.

```
> SCREEN PARTMAIN MENU  &
>   NOMODE  &
>   ACTIVITIES ENTRY  &
>   ACTION  &
>   LABEL "==>" AT 20,1
>
> TITLE "Parts Maintenance" CENTERED AT 4,1
>
> ALIGN (20,25,)
> SKIP TO LINE 8
> SUBSCREEN ADDPART   &
>   MODE E  &
>   LABEL "Add a New Part"
> SUBSCREEN ADDVAR   &
>   MODE E  &
>   LABEL "Add a New Part Variant"
> SUBSCREEN MODPART  &
>   MODE F &
>   LABEL "Change or Delete a Part"
>
> SKIP 1
>
> SUBSCREEN PARTLIST &
>   MODE F &
>   LABEL "Check Inventory Level for a Part"
>
> SUBSCREEN PRTPRICE &
>   MODE F  &
>   LABEL "Check Part Pricing"
>
> SKIP 1
> SUBSCREEN PARTRPT  &
>   ID 20  &
>   LABEL "Generate Parts Summary Reports"
>
> SKIP 2
>
> TITLE "60   Return to Previous Menu" AT ,20
>
> PROCEDURE DESIGNER 60 NODATA
```

```
> BEGIN
>   RETURN
>   END
```

ROLLBACK

Restores the data affected by an update to the state that it was in before the transaction was started.

Syntax

**ROLLBACK [[TRANSACTION] transaction_name
[,transaction_name]...]**

TRANSACTION

An optional keyword for documentation purposes only.

transaction_name

Names the transaction with which the ROLLBACK verb is associated.

Discussion

The ROLLBACK verb (without a transaction list) lets you duplicate QUICK's automatic rollback processing by rolling back all locally active transactions and returning to a stable point in screen processing. The SEVERE verb can also be used to rollback locally active transactions.

In contrast to QUICK's automatic rollback processing, the ROLLBACK verb with a transaction list issues a rollback to all listed transactions, whether or not they are locally active. The ROLLBACK verb with a transaction list should be used with extreme care, because no transactions other than those listed by the ROLLBACK verb are affected.

A rollback of an inactive transaction is ignored.

RUN COMMAND

Executes an operating system command.

Syntax

RUN COMMAND string|item [option]...

string|item

Specifies the command to be executed. The command can be a string or the name of a character item.

Options

RUN COMMAND options	
CLEAR ALL|SCREEN|[LINES]	INPUT B|C|SAME
NOCONSOLE	NOWARN
ON ERROR CONTINUE|TERMINATE	REFRESH ALL|SCREEN|[LINES]
RESPONSE	WAIT|NOWAIT

CLEAR ALL|SCREEN|[LINES] n [TO m]

Clears an area of the terminal memory before the command is invoked. Any output to the terminal from the command appears starting on the first line of the cleared area. Lines that are cleared are refreshed automatically when the screen is reactivated and QUICK is ready to prompt the user.

ALL

Clears the entire terminal memory.

SCREEN

Clears the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears the area between and including lines n to m, numbering from the first line of terminal memory. LINE n by itself clears line n only.

INPUT B|C|SAME (MPE/iX)

Puts the terminal in the specified input mode prior to executing the command. QUICK returns the terminal to the original mode after the command is completed.

Default: SAME. However, if a CLEAR, REFRESH, or RESPONSE option is specified, the default is C.

B

Puts the terminal in Block mode. B should only be used for commands that must be run in Block mode.

C

Puts the terminal in Character mode.

SAME

Leaves the terminal in the current input mode. If the screen can be run in Block mode, SAME should only be used for commands that do not write to the terminal.

Default: The mode it was in before QUICK was invoked.

NOCONSOLE (Windows)

Suppresses opening a Command Console window. Normally QUICK opens a second Command Console window to run the command. If the command runs in the background, does not require user input, or does not display useful output, the second command console window may not be necessary.

NOWARN

Specifies that if a command returns a non-zero status (and the ON ERROR CONTINUE option was specified), QUICK does not issue a warning message after executing the command. However, any message issued by the command itself is displayed.

ON ERROR CONTINUE|TERMINATE

Specifies the action to be taken if a system error occurs during the execution of a command. If TERMINATE is in effect, a system error causes QUICK to process the error as it would for an ERROR verb (see (p. 420). TERMINATE is the default value of the option. If CONTINUE is specified, a system error is ignored and processing continues as if the error had not occurred.

REFRESH ALL|SCREEN|[LINES] n [TO m]

Clears and rewrites an area of the terminal memory when the screen is reactivated and QUICK is ready to prompt the user. REFRESH options are performed before, and in addition to, an automatic refresh from any CLEAR option.

ALL

Clears and rewrites the entire terminal memory.

SCREEN

Clears and rewrites the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears and rewrites the area between and including lines n to m, numbering from the first line of terminal memory. LINE n by itself refreshes line n only.

RESPONSE

Prompts the QUICK screen user for a response after the command finishes executing. This delay allows the screen user to view the results of the command before QUICK refreshes the screen.

WAIT|NOWAIT (Windows)

The WAIT option instructs QUICK to suspend current screen processing until the command has executed, at which time control returns to the screen. The NOWAIT option specifies that screen processing continues immediately and the command executes concurrently.

Default: NOWAIT

Discussion

UNIX, Windows: Although the command runs in a subprocess from the main QUICK process, it starts a separate shell (UNIX) or command (Windows). The results of a setenv (UNIX) or set (Windows) command are not accessible from QUICK or any later commands. For this use the SETSYSTEMVAL function.

Limit: The combined maximum number of SUBSCREEN statements, RUN SCREEN and RUN COMMAND verbs is 256 per screen; if you exceed this limit, QDESIGN issues an error message.

Example

In the following example, the RUN COMMAND verb invokes QUIZ.

```
> SCREEN PERSDATA
>
> FILE EMPLOYEES
>
> FIELD EMPLOYEENUMBER OF EMPLOYEES &
>       REQUIRED NOCHANGE LOOKUP NOTON EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> FIELD FIRSTNAME OF EMPLOYEES
> FIELD DIVISION OF EMPLOYEES
>
> PROCEDURE DESIGNER QZ NODATA
>   RUN COMMAND  &
>     "QUIZ AUTO=PERSRPRT"
>
> BUILD
```

RUN REPORT

Specifies a report program to execute.

Syntax

RUN REPORT filespec [option]...

filespec

The file specification of an executable report program.

Options

RUN COMMAND options	
CLEAR ALL|SCREEN|[LINES]	INPUT B|C|SAME
NOCONSOLE	NOWARN
ON ERROR CONTINUE|TERMINATE	REFRESH ALL|SCREEN|[LINES]
RESPONSE	WAIT|NOWAIT

CLEAR ALL|SCREEN|[LINES] n [TO m]

Clears an area of the terminal memory before the command is invoked. Any output to the terminal from the command appears starting on the first line of the cleared area. Lines that are cleared are refreshed automatically when the screen is reactivated and QUICK is ready to prompt the user.

ALL

Clears the entire terminal memory.

SCREEN

Clears the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears the area between and including lines n to m, numbering from the first line of terminal memory. LINE n by itself clears line n only.

INPUT B|C|SAME (MPE/iX)

Puts the terminal in the specified input mode prior to executing the command. QUICK returns the terminal to the original mode after the command is completed.

Default: SAME. However, if a CLEAR, REFRESH, or RESPONSE option is specified, the default is C.

B

Puts the terminal in Block mode. B should be used only for commands that must be run in Block mode.

C

Puts the terminal in Character mode.

SAME

Leaves the terminal in the current input mode. If the screen can be run in Block mode, SAME should be used only for commands that do not write to the terminal.

Default: The mode that the terminal was in before QUICK was invoked.

NOCONSOLE (Windows)

Suppresses opening a Command Console window. Normally QUICK opens a second Command Console window to run QUIZ. If QUIZ runs in the background, does not require user input, or does not display useful output, the second command console window may not be necessary.

NOWARN

Specifies that if a command returns a non-zero status (and the ON ERROR CONTINUE option was specified), QUICK does not issue a warning message after executing the command. However, any message issued by the command itself is displayed.

ON ERROR CONTINUE|TERMINATE

Specifies the action to be taken if a system error occurs during the execution of a command. If TERMINATE is in effect, a system error causes QUICK to process the error as it would for an ERROR verb (p. 420). TERMINATE is the default value of the option. If CONTINUE is specified, a system error is ignored and processing continues as if the error did not occur.

REFRESH ALL|SCREEN|[LINES] n [TO m]

Clears and rewrites an area of the terminal memory when the screen is reactivated and QUICK is ready to prompt the user. REFRESH options are performed before, and in addition to, an automatic refresh from any CLEAR option.

ALL

Clears and rewrites the entire terminal memory.

SCREEN

Clears and rewrites the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears and rewrites the area between and including lines n to m, numbering from the first line of terminal memory. LINE n by itself refreshes only line n.

RESPONSE

Prompts the QUICK screen user for a response after the command finishes executing. This delay allows the screen user to view the results of the command before QUICK refreshes the screen.

WAIT|NOWAIT (Windows)

The WAIT option instructs QUICK to suspend current screen processing until after the report executes, at which time control returns to the screen. The NOWAIT option specifies that screen processing continues immediately and the report executes concurrently.

Default: NOWAIT

RUN RUN

Specifies a QTP program to execute.

Syntax

RUN RUN filespec [options...]

filespec

The file specification of an executable QTP program.

Options

RUN COMMAND options	
CLEAR ALL|SCREEN|[LINES]	INPUT B|C|SAME
NOCONSOLE	NOWARN
ON ERROR CONTINUE|TERMINATE	REFRESH ALL|SCREEN|[LINES]
RESPONSE	WAIT|NOWAIT

CLEAR ALL|SCREEN|[LINES] n [TO m]

Clears an area of the terminal memory before the command is invoked. Any output to the terminal from the command appears starting on the first line of the cleared area. Lines that are cleared are refreshed automatically when the screen is reactivated and QUICK is ready to prompt the user.

ALL

Clears the entire terminal memory.

SCREEN

Clears the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears the area between and including lines n to m, numbering from the first line of terminal memory. LINE n by itself clears only line n.

INPUT B|C|SAME (MPE/iX)

Puts the terminal in the specified input mode prior to executing the command. QUICK returns the terminal to the original mode after the command is completed.

Default: SAME. However, if a CLEAR, REFRESH, or RESPONSE option is specified, the default is C.

B

Puts the terminal in Block mode. B should be used only for commands that must be run in Block mode.

C

Puts the terminal in Character mode.

SAME

Leaves the terminal in the current input mode. If the screen can be run in Block mode, SAME should be used only for commands that do not write to the terminal.

Default: The mode that the terminal was in before QUICK was invoked.

NOCONSOLE (Windows)

Suppresses opening a Command Console window. Normally QUICK opens a second Command Console window to run QTP. If QTP runs in the background, does not require user input, or does not display useful output, the second command console window may not be necessary.

NOWARN

Specifies that if a command returns a non-zero status (and the ON ERROR CONTINUE option was specified), QUICK does not issue a warning message after executing the command. However, any message issued by the command itself is displayed.

ON ERROR CONTINUE|TERMINATE

Specifies the action to be taken if a system error occurs during the execution of a command. If TERMINATE is in effect, a system error causes QUICK to process the error as it would for an ERROR verb (p. 420). TERMINATE is the default value of the option. If CONTINUE is specified, a system error is ignored and processing continues as if the error did not occur.

REFRESH ALL|SCREEN|[LINES] n [TO m]

Clears and rewrites an area of the terminal memory when the screen is reactivated and QUICK is ready to prompt the user. REFRESH options are performed before, and in addition to, an automatic refresh from any CLEAR option.

ALL

Clears and rewrites the entire terminal memory.

SCREEN

Clears and rewrites the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears and rewrites the area between and including lines n to m, numbering from the first line of terminal memory. LINE n by itself refreshes only line n.

RESPONSE

Prompts the QUICK screen user for a response after the command finishes executing. This delay allows the screen user to view the results of the command before QUICK refreshes the screen.

WAIT|NOWAIT (Windows)

The WAIT option instructs QUICK to suspend current screen processing until after the run executes, at which time control returns to the screen. The NOWAIT option specifies that screen processing continues immediately and the run executes concurrently.

Default: NOWAIT

RUN SCREEN

Invokes a lower-level screen.

Syntax

RUN [SCREEN] filespec|{ITEM item} [option]...

filespec

Names the file containing the compiled QUICK screen that you want to invoke.

Limit: You cannot call subscreens named ITEM unless you precede the file specification with a percent sign (%).

ITEM item

Indicates that the subscreen's file specification is defined in an item.

item

Names the item in which the subscreen's file specification is defined. The item can be either a record item, a temporary item, or a defined item.

Limit: The item type must be CHARACTER or VARCHAR.

Options

RUN SCREEN options	
CLEAR ALL|SCREEN|[LINES]	INPUT B|C|SAME
KEEP ROLLBACK	MODE
ON ERROR CONTINUE|TERMINATE	PASSING
REFRESH ALL|SCREEN|[LINES]	RESPONSE
WINDOW	

CLEAR ALL|SCREEN|[LINES] n [TO m]

Clears an area of terminal memory before the screen is called. Any terminal writes from the screen appear, starting on the first line of the cleared area. Lines cleared are refreshed automatically when the screen is reactivated and QUICK is ready to prompt the user.

The CLEAR option doesn't require the **restore=lines** program parameter to be used.

ALL

Clears the entire terminal memory.

SCREEN

Clears the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears the area between and including lines n to m, numbering from the first line of terminal memory. LINE n by itself clears line n only.

INPUT B|C|SAME (MPE/iX)

Specifies the input mode the subscreen is to be in when it appears. The terminal is not put back to the original mode on return from the called screen.

Default: SAME

B

Starts the screen in Block mode if the subscreen can be run in Block mode.

C

Starts the subscreen in Character mode.

SAME

Starts the subscreen in the current mode of the calling screen.

KEEP ROLLBACK [INFORMATION]

Overrides the default behavior of clearing the non-relational rollback buffers when the subscreen is loaded.

Limit: Affects non-relational records only.

Limit: KEEP ROLLBACK INFORMATION is allowed only on a RUN SCREEN statement in a recoverable procedure or in an INTERNAL procedure invoked from a recoverable procedure.

INFORMATION

For documentation purposes only.

MODE E|F|S|NULL|SAME|GHOST

Specifies the mode of the subscreen.

Default: E for subscreen invoked during the standard Entry sequence; otherwise NULL.

E|F|S

Indicates that the subscreen is in one of Entry (E), Find (F), or Select (S) mode when it first appears.

NULL

Indicates no mode. QUICK prompts for a mode at the Action field when the screen appears.

SAME

Indicates that the subscreen is in the same mode as the current screen when the screen named in the verb is invoked.

GHOST

Indicates that the subscreen being called is a "ghost" screen. This causes QUICK to skip the refreshing of the calling screen when returning from the subscreen call.

When QUICK runs a subscreen with the GHOST option, the default mode is used.

The GHOST option should only be used to call a ghost screen; that is, a screen that does all its work in the INITIALIZE procedure with no terminal output. If the GHOST option is used on a subscreen that is not a ghost screen, results will be unpredictable and screen corruption may occur.

ON ERROR CONTINUE|TERMINATE

Determines whether processing on the calling screen continues or terminates if an error which the user had no opportunity to correct occurs on the subscreen. This option only has an effect when an error on a subscreen is not displayed to the user on that subscreen.

CONTINUE

The execution of the calling screen continues, regardless of the fact that an error which the user had no opportunity to correct, occurred on the subscreen.

TERMINATE

When an error which the user had no opportunity to correct, occurs on the subscreen, processing on the calling screen terminates as if the SUBSCREEN statement failed.

Default: TERMINATE

Prior to 7.33C (UNIX), 7.10E1 (OpenVMS), and 8.09 (MPE/IX), the default behavior was equivalent to CONTINUE.

PASSING record-structure|item [,record-structure|item]...

Specifies which of the current screen's existing record-structures, defined items, and temporary items are passed to the named screen.

Entity names in the list must be separated by commas. Items must match, on the basis of identical item attributes, with the items named in the RECEIVING option of the lower-level SCREEN statement. The names themselves may differ.

Passing a defined item allows you to use a higher-level screen expression on the lower-level screen without having to redeclare it. No value is passed. Defined items on higher-level screens can only be passed to defined items on lower-level screens; temporary items on higher-level screens can only be passed to temporary items on lower-level screens.

Limit: A combined maximum of 16 files and items.

REFRESH ALL|SCREEN|[LINES] n [TO m]

Clears and rewrites an area of the terminal memory when the screen is reactivated and QUICK is ready to prompt the user. REFRESH options are performed before, and in addition to, an automatic refresh from any CLEAR option.

ALL

Clears and rewrites the entire terminal memory.

SCREEN

Clears and rewrites the area taken by the current QUICK screen.

[LINES] n [TO m]

Clears and rewrites the area between and including lines n to m, numbering from the first line of terminal memory. LINE n by itself refreshes line n only.

RESPONSE

Prompts the QUICK screen user for a response after the subscreen finishes executing. This delay allows the screen user to view the results of the subscreen before QUICK refreshes the screen.

WINDOW WIDTH CONSTANT|DEFAULT WHEN CALLING| WHEN RETURNING

Overrides the default behavior and sets the terminal to the correct screen width (80 or 132 columns).

WHEN CALLING

Keeps the current screen width when calling a subscreen.

WHEN RETURNING

Keeps the screen width of the subscreen when returning from the subscreen.

Discussion

The RUN SCREEN verb initiates the activities necessary to invoke a screen. QDESIGN generates one RUN SCREEN verb in the ENTRY procedure for each SUBSCREEN statement with an AUTO or IF option. SCREEN is used for documentation only.

Limit: The combined maximum number of SUBSCREEN statements, RUN SCREEN, and RUN COMMAND verbs is 256 per screen; if you exceed this limit, QDESIGN issues an error message.

Note: For information about verb and procedure compatibility, see (p. 239).

Dynamic Screen Calls in QDESIGN and QUICK

You can call subscreens by using the SUBSCREEN statement, the THREAD statement, the RUN SCREEN verb, or the RUN THREAD verb. Each of these statements/verbs works in either of two ways:

- You provide the subscreen's file specification directly in the statement or verb syntax. Thus, the same subscreen is called every time the statement or verb is executed.
- You use the ITEM keyword to point the statement or verb to an item that contains the subscreen's file specification. Thus, a different subscreen might be called each time depending on what file specification the item contains at the moment of execution.

The latter method is known as dynamic screen calling because you control which subscreens are called based on run-time variables. This allows you to build context-sensitive applications in which different users may see different screens based on these variables.

For more information about the SUBSCREEN statement or the THREAD statement, see (p. 210) and (p. 221), respectively. For more information about the RUN THREAD verb, see (p. 478).

Rollback and Subscreens

By default, when a subscreen is called, all rollback information for non-relational records on the calling screen is lost.

You can call screens in the Update phase from the recoverable procedures without losing rollback information for the non-relational records PUT on the calling screen. To do this, you specify the KEEP ROLLBACK [INFORMATION] option on the RUN SCREEN verb.

If you use this option, then PUTs performed before the RUN SCREEN verb is executed can be rolled back in the event of an error occurring on the subscreen, or after return to the calling screen.

Example

The following example illustrates how the RUN SCREEN verb can invoke a subscreen to perform a lookup on a field. In this example:

- Entering an exclamation mark (!) in the BRANCH field of the EMPBRNCH screen causes the display of a lower-level screen that lists valid BRANCH codes.
- The MODE F option determines that the subscreen appears in Find mode.
- The SCREEN statement of BRNCHCHK must have a receiving specification that accepts the item TEMPBRANCH.

```
> SCREEN EMPBRNCH
> TEMPORARY TEMPBRANCH CHARACTER*2
> FILE EMPLOYEES PRIMARY
> FIELD EMPLOYEENUMBER OF EMPLOYEES REQUIRED &
>   NOCHANGE LOOKUP NOTON EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> FIELD BRANCH OF EMPLOYEES
> PROCEDURE INPUT BRANCH
>   BEGIN
>     IF FIELDTEXT = "!"
>       THEN BEGIN
>         RUN SCREEN BRNCHCHK &
>            PASSING TEMPBRANCH MODE F
>         LET FIELDTEXT = TEMPBRANCH
>         END
>     END
> BUILD
>
.
.
```

```
.
> SCREEN BRNCHCHK RECEIVING TEMPBRANCH &
>  ACTIVITIES FIND
>
> TEMPORARY TEMPBRANCH CHARACTER *2
> FILE BRANCHES PRIMARY &
>   OCCURS 15
>   ACCESS SEQUENTIAL
>
> SKIP TO 3
> TITLE "Enter the ID-Number for the desired Branch." &
>   CENTERED
> SKIP 1
> ALIGN (1,,4) (,,10) (,,35)
>
> CLUSTER OCCURS WITH BRANCHES
>   FIELD BRANCH OF BRANCHES &
>     REQUIRED &
>     NOCHANGE &
>     LOOKUP NOTON BRANCHES
>   FIELD BRANCHNAME OF BRANCHES
>   FIELD BRANCHMGR OF BRANCHES
> CLUSTER
>
> PROCEDURE DESIGNER 01
> BEGIN
>   LET TEMPBRANCH = BRANCH OF BRANCHES
>   RETURN
> END
>
> BUILD
```

In the following example, the conditional-expression in item SUB1 tests the user's application security class, resulting in a file specification for either a restricted-access screen or a full-access screen, as appropriate:

```
> SCREEN STAFF
>
> DEFINE SUB1 CHAR*31 =    &
>    "Employee_All" IF MATCHUSER ("MANAGER")   &
>    ELSE "Employee_Restricted"
.
.
.
>    RUN SCREEN ITEM SUB1
```

RUN THREAD

Specifies a screen thread.

Syntax

RUN [THREAD] filespec|{ITEM item} [option]...

filespec

Names the root screen of a screen thread.

Limit: You cannot call root screens named ITEM unless you precede the file specification with a percent sign (%).

ITEM item

Indicates that the root screen's file specification is defined in an item.

item

Names the item in which the root screen's specification is defined. The item can be either a record item, a temporary item, or a defined item.

Limit: The item type must be CHARACTER or VARCHAR.

Options

RUN THREAD options	
INPUT B|C	MODE E|F|S|SAME
RESPONSE	SHARED
WINDOW	

INPUT B|C (MPE/iX)

Specifies the input mode the thread is to be in when it appears. (Although the SAME sub-option is accepted syntactically, it has no meaning in the context of threads.)

Default: Unless overridden by other means, such as in QKGO or the SCREEN statement, when a thread is first opened, it will be opened in Character mode. When toggling back to a thread, it will be in the same input mode that it was in when it was left.

B

Starts the thread in Block mode if the thread can be run in Block mode.

C

Starts the thread in Character mode.

MODE E|F|S|NULL|SAME

Specifies the mode of the thread.

Default: E for thread invoked during the standard Entry sequence; otherwise NULL.

E|F|S

Indicates that the thread is in one of Entry (E), Find (F), or Select (S) mode when it first appears.

NULL

Indicates no mode. QUICK prompts for a mode at the Action field when the screen appears.

SAME

Indicates that the thread is in the same mode as the current screen when the screen named in the verb is invoked.

RESPONSE

Prompts the QUICK screen user for a response after the thread finishes executing. This delay allows the screen user to view the results of the thread before QUICK refreshes the screen.

SHARED

If the SHARED option is specified and the thread already exists, then the RUN THREAD verb is ignored.

WINDOW WIDTH CONSTANT|DEFAULT WHEN CALLING

Overrides the default behavior and sets the terminal to the correct screen width (80 or 132 columns).

Discussion

The RUN THREAD verb specifies the screen to be loaded as the root of a new screen thread. This is functionally equivalent to the RUN SCREEN verb, except that no passing and receiving lists are possible.

Note: For information about verb and procedure compatibility, see (p. 239).

Dynamic Screen Calls in QDESIGN and QUICK

You can call subscreens by using the SUBSCREEN statement, the THREAD statement, the RUN SCREEN verb, or the RUN THREAD verb. Each of these statements/verbs works in either of two ways:

- You provide the subscreen's file specification directly in the statement or verb syntax. Thus, the same subscreen is called every time the statement or verb is executed.
- You use the ITEM keyword to point the statement or verb to an item that contains the subscreen's file specification. Thus, a different subscreen might be called each time depending on what file specification the item contains at the moment of execution.

The latter method is known as dynamic screen calling because you control which subscreens are called based on run-time variables. This allows you to build context-sensitive applications in which different users may see different screens based on these variables.

For more information about the SUBSCREEN statement or the THREAD statement, see (p. 210) and (p. 221), respectively. For more information about the RUN SCREEN verb, see (p. 473).

Example

The following example illustrates how the RUN THREAD verb can invoke a separate screen hierarchy. In this example:

- Entering two asterisks (**) in the BRANCH field of the EMPBRNCH screen causes the display of a screen (in a separate hierarchy) that lists valid BRANCH codes.
- The MODE F option determines that the screen appears in Find mode.

```
> SCREEN EMPBRNCH
> TEMPORARY TEMPBRANCH CHARACTER*2
> FILE EMPLOYEES PRIMARY
> FIELD EMPLOYEENUMBER OF EMPLOYEES REQUIRED &
>   NOCHANGE LOOKUP NOTON EMPLOYEES
> FIELD LASTNAME OF EMPLOYEES REQUIRED NOCHANGE
> FIELD BRANCH OF EMPLOYEES
>
> PROCEDURE INPUT BRANCH
>   BEGIN
>     IF FIELDTEXT = "**"
```

```
>       THEN BEGIN
>         RUN THREAD BRNCHCHK MODE F
>         END
>     END
> BUILD
>
.
.
.
```

In the following example, the conditional-expression in item SUB1 tests the user's application security class, resulting in a file specification for either a restricted-access screen or a full-access screen, as appropriate:

```
> SCREEN STAFF
>
> DEFINE SUB1 CHAR*31 =    &
>    "Employee_All" IF MATCHUSER ("MANAGER")   &
>    ELSE "Employee_Restricted"
.
.
.
>    RUN THREAD ITEM SUB1
```

SELECT

Prompts for a selection value in Select Mode.

Syntax

SELECT [ITEM] field

field

Specifies the field to be used in the selection of data records on a QUICK screen.

Discussion

In the default SELECT procedure, SELECT verbs determine the fields in which a QUICK screen user can enter selection criteria for the retrieval of data records. When QUICK processes a SELECT verb, the user is prompted for a value in the field referenced by the SELECT verb. QUICK then uses this value as a selection criterion when selecting data records.

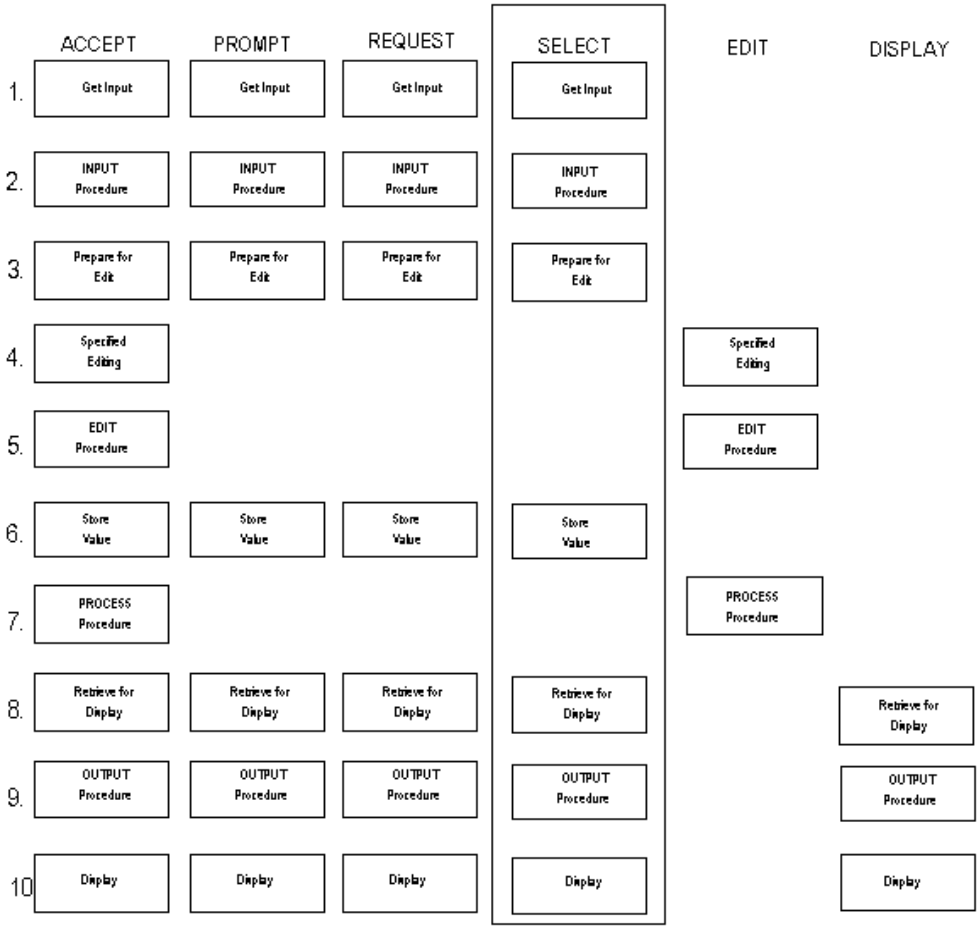
Where the SELECT Verb is Used

The SELECT verb is used primarily in the default SELECT procedure. Each QUICK screen field causes QDESIGN to generate a SELECT verb in the SELECT procedure, except fields for which the NOSELECT option is specified. No prompting occurs for a field with the NOSELECT option, even if the SELECT procedure contains a SELECT verb for that field.

No SELECT procedure is generated for screens that are created with the NOPANEL option specified for either the SCREEN or the SET statement.

Processes Initiated by the SELECT Verb

The following figure illustrates the processes that are performed when the SELECT verb is executed, and contrasts these steps with the steps that are performed by other field processing verbs:



The SELECT verb follows the same steps as the ACCEPT verb except that Steps 4, 5, and 7 (editing and processing) are bypassed. For details about the steps listed in the preceding figure, see (p. 364).

Differences Between the REQUEST, SELECT, and PROMPT Verbs

The REQUEST, SELECT, and PROMPT verbs prompt the QUICK screen user for data in a field but store the entered values in different internal buffers.

The REQUEST verb places the response entered in the request buffer for the current screen. The values in the request buffer are used to retrieve data based on the PATH and FIND procedures. The values in the request buffer are used by the underlying file system to establish a retrieval criteria.

The SELECT verb places the response entered in the select buffer for the current screen. QUICK uses the values in this buffer after it has retrieved values determined by the retrieval criteria.

The PROMPT verb prompts for a value and stores the entered response into the field that's referenced by the PROMPT verb.

Always use the REQUEST verb rather than the PROMPT verb when you prompt for segment values to be used in record retrieval.

Using the SELECT Verb in Procedures Other Than the SELECT Procedure

You can use the SELECT verb in procedures other than the SELECT procedure. For example, the use of SELECT verbs in the POSTFIND procedure allows you to specify additional selection criteria after QUICK has retrieved data in the FIND procedure.

Example

The following example illustrates the use of the SELECT verb to restrict prompting in Select mode. In this example, the SELECT procedure restricts the specification of selection values to the fields CITY, PROVSTATE, and POSTALCODE.

```
> PROCEDURE SELECT
>   BEGIN
>     SELECT CITY OF EMPLOYEES
>     SELECT PROVSTATE OF EMPLOYEES
>     SELECT POSTALCODE OF EMPLOYEES
>   END
```

SEVERE

Aborts processing and issues a severe-level message.

Syntax

SEVERE [MESSAGE] string|=string-expression|n

string

Defines the contents of the message using a string.

=string-expression

Defines the contents of the message using a string expression.

n

Defines a message number that corresponds to a message in a designer message file. The designated file is QKMSGDES.

For more information about message files, see Chapter 4, "Messages in PowerHouse", in the *PowerHouse Rules* book.

Discussion

Other than in Compatible Block mode (**MPE/iX**), the SEVERE verb instructs QUICK to display the stated message on the message line, perform the BACKOUT procedure (if specified), reinitialize the buffers, and reprompt at the Action field.

A SEVERE message has the highest priority in the four-level message hierarchy and is displayed if the message line is empty or if it contains an ERROR, WARNING, or INFORMATION message.

MPE/iX: In Compatible Block mode, the SEVERE verb instructs QUICK to highlight fields in error, display the stated message on the message line, reinitialize the buffers, perform the BACKOUT procedure (if specified), and position the cursor at the field associated with the error message.

Note: For information about verb and procedure compatibility, see (p. 239).

START

Starts a transaction.

Syntax

START [TRANSACTION] transaction_name [,transaction_name]

TRANSACTION

An optional keyword for documentation purposes only.

transaction_name

Names the transaction with which the START verb is associated.

Discussion

In most cases, the START verb is not required as transactions are started implicitly by PowerHouse. The START verb lets you start a transaction without doing any explicit I/O to the database. It also forces all physical transactions associated with QUICK's logical transaction to start together, ensuring a consistent start time across databases.

If a transaction definition contains a RESERVING list, the tables are reserved at the start of the transaction and are therefore available for the duration of the transaction. For a full description of the RESERVING list impact, see the appropriate database documentation.

A START issued on an already active transaction will cause a run-time error to occur.

STARTLOG (MPE/iX)

Marks the start of a transaction set in an IMAGE log file.

Syntax

STARTLOG filespec [string-expression]

filespec

Names a file that is a dataset in an IMAGE database for which updates are to be logged.

string-expression

Defines a string expression that specifies a message written to the standard IMAGE log file.

Default: The screen name.

Discussion

STARTLOG is used to indicate the start of a transaction set (a set of records that are to be treated as a single transaction for backup and recovery purposes).

For IMAGE files, STARTLOG issues a DBBEGIN call. For other types of files, the verb is ignored. A STARTLOG can be issued for each database defined on a screen. Use the name of any IMAGE dataset in the desired database.

If no STARTLOG or STOPLOG verb is included in your UPDATE procedure, but MEMOLOG is, then the message is written to the log file using DBMEMO. STARTLOG only issues an IMAGE DBBEGIN the first time it is invoked for a given IMAGE database. If you issue STARTLOG again with any file that belongs to the same database before invoking STOPLOG, then the message is written to the log file using DBMEMO.

Note: For information about verb and procedure compatibility, see (p. 239).

STOPLOG (MPE/iX)

Marks the end of a transaction set in an IMAGE log file.

Syntax

STOPLOG [string-expression]

string-expression

Defines a string expression that specifies a message written to the standard IMAGE log file.

Default: The screen name.

Discussion

STOPLOG is used to perform end-of-transaction processing for any STARTLOG verbs that have been executed since the last STOPLOG. For IMAGE files, STOPLOG issues a DBEND call for each DBBEGIN call issued by STARTLOG. For other types of files, the verb is ignored.

Note: For information about verb and procedure compatibility, see (p. 239).

UNLOCK

Unlocks a file.

Syntax

UNLOCK [ALL|filespec [BASE|FILE|RECORD]] (MPE/iX)

UNLOCK [ALL|filespec [FILE|RECORD]] (OpenVMS, UNIX, Windows)

ALL

Unlocks all files at all lock levels. If an unlock is executed on any file, all files sharing the open are unlocked.

filespec

Names the file to be unlocked.

BASE (MPE/iX)

Specifies that only base-level and file-level locks are to be released.

FILE

OpenVMS, UNIX, Windows: Specifies that only file-level locks are to be released.

MPE/iX: Specifies that only base-level and file-level locks are to be released.

RECORD

Specifies that only record-level locks are to be released.

Discussion

The UNLOCK verb releases locks applied using the LOCK verb, including LOCK verbs generated by the LOCK option of the SCREEN statement. If no lock has been established, the UNLOCK verb has no effect. If neither the FILE nor RECORD option is specified, then both file and record-level locks are released.

The UNLOCK verb is ignored for all relational database products. Locks are released when the transaction ends.

MPE/iX: It is not possible to unlock individual files or records in an IMAGE database. If an UNLOCK is done, then all locks that are held through the same physical open of the database are released at the same time. This can have unexpected results if file-level, record-level, or a combination of file and record-level locks exist on more than one dataset in the database when the UNLOCK is done.

For IMAGE datasets, the BASE and FILE options are equivalent. These options can be added to the UNLOCK verbs to document the original lock level. Both options release any base-level and file-level locks that are held. If file-level locks are being used and more than one dataset is locked through the same physical open of the database, the locks on all of these datasets are released. If only record level locks exist through this open of the database, these locks are not released by the BASE or FILE options. If, however, both file-level and record-level locks are held through the same physical open of the database, the record-level locks are released.

The RECORD option can only be used with IMAGE datasets. It releases any record-level locks that are held. If record-level locks are held on more than one dataset through the same physical open of the database, the locks on all of these datasets are released. If only base-level or file-level locks exist, these locks are not released by the RECORD option. If, however, both record-level and file-level locks are held through the same physical open of the database, the file-level locks are released.

[SQL] UPDATE

Updates rows in a table.

Syntax

```
[SQL[IN database]
[TRANSACTION transaction_name
    [FOR {CONSISTENCY|{[CONCURRENCY]
        phase-option [,phase-option]...}}]]]
UPDATE tablespec SET column-name = {sql-expression|NULL}
    [,column-name = {sql-expression|NULL}]...
    [WHERE sql-condition|{DBKEY=:expression}]
```

IN database

Specifies the name PowerHouse uses to attach to the database. This is the name used to declare the database in PDL.

TRANSACTION transaction_name [FOR {CONSISTENCY| {[CONCURRENCY] phase-option[,phase-option]...}]...

Defines transactions used for relational data structures.

transaction_name

Any valid PowerHouse name.

FOR CONSISTENCY

Determines that the SQL statement is associated with a particular transaction in Consistency model.

Limit: Only one transaction association can be specified.

FOR [CONCURRENCY] phase-option [,phase-option]...

Determines that the SQL statement is associated with a particular transaction or transactions in Concurrency model.

Limit: Up to three transaction associations can be specified.

phase-option

Specifies the screen phase with which the transaction is associated.

Phase option	Description
PROCESS	The phase in which you are entering, correcting, or changing data records on the screen.
QUERY	The phase in which data is retrieved from the database.
UPDATE	The phase in which data is updated.

UPDATE tablespec

Identifies the table where rows are to be updated. The syntax for tablespec is:

MPE/iX:	**[[database.]owner.]table-name**
OpenVMS, UNIX, Windows:	**server-name.[database-name.] [owner-name.]table-name**

If server-name is included in a Sybase tablespec, double quotation marks are required for the server-name and database-name. For example,

```
"dbsvr01.accnt".manager.billings_tbl
```

For Oracle, the syntax is

[owner-name.]table-name[@database-linkname]

If the database-linkname is included, it is treated as part of the table-name, and double quotation marks are required. Here is an example:

```
manager."billings_tbl@dblnk01"
```

Oracle synonyms may be used for table-names. For more information about how PowerHouse uses Oracle synonyms, see "PowerHouse Language Rules" in the *PowerHouse Rules* book.

SET column-name = sql-expression|NULL [,column-name = sql-expression|NULL]...

Identifies the columns to be updated and their new values.

WHERE sql-condition|DBKEY=:expression

Sql-condition identifies the rows of the table to be updated. If you don't specify a WHERE clause, all the rows in the relational table are changed. DBKEY is available only if the underlying database supports it.

The sql-condition is a condition which is limited for use within Cognos SQL syntax. For more information about SQL conditions, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book, or refer to an SQL reference manual.

Limit: DBKEY cannot be used with Sybase.

Discussion

The SQL UPDATE statement acts directly on a table or view in the database and is never shown in PowerHouse procedural code.

WARNING

Issues a warning message.

Syntax

WARNING [MESSAGE] string|=string-expression|n
[NOW [RESPONSE]]

Defines the contents of the message. MESSAGE is used only for documentation.

string

Defines the contents of the message using a string.

=string-expression

Defines the contents of the message using a string expression.

n

Defines a message number that corresponds to a message in a designer message file. The designated file is QKMSGDES.

For more information about message files, see Chapter 4, "Messages in PowerHouse", in the *PowerHouse Rules* book.

NOW [RESPONSE]

Forces the display of the specified message on the screen when the WARNING verb is executed in a procedure. Otherwise, the message isn't displayed until the next time QUICK prompts for user input. When you use the NOW option, QUICK writes the contents of the display buffer to the screen. This action refreshes the screen and any function key labels.

RESPONSE temporarily erases the top line of the screen and prompts the user to press [Return].

Discussion

The WARNING verb instructs QUICK to display the stated message on the message line, but has no effect on processing unless the RESPONSE option is included. (There is one exception: if the string is greater than the screen width, you are prompted to press Return even if the RESPONSE option is not used.) If a warning is issued during an UPDATE or PREUPDATE procedure, QUICK treats any update command as a US (Update Stay) command so that the current data and the warning remain visible after the update.

A warning message can be displayed only when the message line is empty or when it contains an INFORMATION message, unless the NOW option is included.

Once the message has been displayed, the message line in the display buffer is cleared.

Note: For information about verb and procedure compatibility, see (p. 239).

WHILE

Executes the next procedural statement as long as the condition is true.

Syntax

WHILE condition

condition

States a condition to be evaluated.

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

Discussion

The WHILE control structure is used for executing the next procedural statement as long as a condition is true. The statement can be a compound statement. The WHILE condition creates a looping effect that ends when the condition fails or when a BREAK verb is encountered.

Nesting FOR, WHILE, and WHILE RETRIEVING Control Structures

You can nest control structures within other control structures as follows:

The control structure ...	Can contain the control structure(s) ...
FOR	WHILE
WHILE	FOR, WHILE, or WHILE RETRIEVING
WHILE RETRIEVING	FOR or WHILE

Although there are limitations on which of these control structures you can nest explicitly, you can often get around these by calling from within the loop another procedure that contains the control structure you want to nest.

Example

In the following example, a WHILE control structure is used to retrieve a specified number of SALESHIST (sales history) records. These records are then used to calculate the Total Sales, the sales representative's highest and lowest year of sales, and the average yearly sales. The user is prompted to enter a value for "How many years of history?". This value determines the maximum number of times the WHILE loop is processed. In this example:

- CURR-YR is initialized to the current year.
- LOW-SALES is initialized to the highest possible value.
- FILE SALESHIST DESIGNER counts the number of history records.
- WHILE YRS-REQUESTED NE ACTUAL-YRS indicates the start of WHILE loop.
- The format for the SHIST-IDX looks like this:

```
INDEX SHIST-IDX PRIMARY UNIQUE
SEGMENT EMPLOYNO
SEGMENT SALES-YEAR
```

- IF YRLY-SALES LT LOW-SALES determines LOW SALES and year of sales.
- IF YRLY-SALES GT HIGH-SALES determines HIGH SALES and year of sales.
- LET CURR-YR = CURR-YR + 1 indicates that the value of CURR-YR is increased by 1 each time the WHILE loop is processed and is used in place of the segment SALES-YEAR when reading a SALESHIST record.
- BREAK indicates the end of ACCESSOK

```
> SCREEN HIST ACTIVITIES FIND
```

```
> TEMP CURR-DATE DATE INIT SYSDATE
> TEMP CURR-YR NUM*4 INITIAL &
>      DATEEXTRACT (CURR-DATE, YEAR)
> TEMP YRS-REQUESTED NUM*2
> TEMP TOTAL-SALES NUM*9
> TEMP HIGH-SALES NUM*8
> TEMP HIGH-YEAR NUM*4
> TEMP LOW-SALES NUM*8 INITIAL 99999999
> TEMP LOW-YEAR NUM*4
> TEMP ACTUAL-YRS NUM*2
> TEMP NUM-YEAR NUM*2
> FILE EMPLOYEES PRIMARY
> FILE SALESHIST DESIGNER
> DEFINE SALESREP CHAR*35 =   &
>      PACK(FIRSTNAME + " " + LASTNAME)
> DEFINE SYSYEAR = DATEEXTRACT (CURR-DATE, YEAR)
> DEFINE AVG-SALES NUM*8 = TOTAL-SALES / ACTUAL-YRS
> TITLE "Sales History Screen"
> SKIP 2
> ALIGN (10,14,25) (,38,44)
> FIELD EMPLOYNO OF EMPLOYEES REQUIRED NOCHANGE &
>      LABEL "Salesrep:"
> FIELD SALESREP LABEL "Name:"
> SKIP 1
> ALIGN (,24,53)
> FIELD YRS-REQUESTED   &
> LABEL "How many years of history?" &
>      FIXED BWZ VALUES 1 TO 99
> SKIP 2
> ALIGN (,14,48)
> FIELD TOTAL-SALES PIC " ^,^^^,^^^.^^" FLOAT "$" &
>      LABEL "Total Sales for previous years:"
> SKIP 1
> ALIGN (,14,28) (,36,50)
> FIELD HIGH-YEAR LABEL "Best Year:"
> FIELD HIGH-SALES LABEL "Sales were:" &
>      PIC " ^^^,^^^.^^" FLOAT "$"
> FIELD LOW-YEAR LABEL "Lowest Year:"
> FIELD LOW-SALES LABEL "Sales were:" &
>      PIC " ^^^,^^^.^^" FLOAT "$"
> SKIP 1
> ALIGN (,14,50)
> FIELD AVG-SALES LABEL "Average Yearly Sales:" &
>      PIC " ^^^,^^^.^^" FLOAT "$"
> PROCEDURE POSTFIND
> BEGIN
>      LET NUM-YEAR = SYSYEAR - JOINEDYEAR
>      LET ACTUAL-YRS = 0
>      REQUEST YRS-REQUESTED
>      EDIT YRS-REQUESTED ; to force the values
>                         ; to be evaluated.
>      LET CURR-YR = CURR-YR - YRS-REQUESTED
>      WHILE YRS-REQUESTED NE ACTUAL-YRS
>      BEGIN
>         GET SALESHIST VIAINDEX SHIST-IDX  &
>            USING EMPLOYNO, CURR-YR OPTIONAL
>         IF ACCESSOK ; Sales history record on file
>            THEN BEGIN ; Start of ACCESSOK
>            LET ACTUAL-YRS = ACTUAL-YRS + 1
>            LET TOTAL-SALES = TOTAL-SALES + YRLY-SALES
>            IF YRLY-SALES LT LOW-SALES
>               THEN BEGIN
>               LET LOW-SALES = YRLY-SALES
>               LET LOW-YEAR = SALES-YEAR
>               END
>            IF YRLY-SALES GT HIGH-SALES
>               THEN BEGIN
>               LET HIGH-SALES = YRLY-SALES
```

```
>                LET HIGH-YEAR = SALES-YEAR
>                END
>          LET CURR-YR = CURR-YR + 1
>          END
>       ELSE ; SALESHIST record not on file (not ACCESSOK)
>       BEGIN
>          INFO = "This is " + ASCII(ACTUAL-YRS) + &
>                 "years of " + ASCII(YRS-REQUESTED) + &
>                 " year(s) sales history requested." &
>                 NOW RESPONSE
>          BREAK
>          END
>      END ; End of WHILE loop
> END ; End of POSTFIND
```

WHILE RETRIEVING

Retrieves and processes data records in a loop.

Syntax

WHILE RETRIEVING record-structure [option]...

record-structure

Names a record-structure in a DESIGNER file. The record-structure may be one of:

- a cursor defined in a DECLARE CURSOR statement
- a record-structure named in the dictionary
- a table or view defined in a relational database

Limit: The WHILE RETRIEVING control structure is valid for DESIGNER files only.

Options

BACKWARDS

Reverses the sequence in which the data records are usually read.

Limit: Valid only for C-ISAM, DISAM, RMS ISAM, and IMAGE datasets with keyed access.

Limit: The BACKWARDS and SEQUENTIAL options cannot be used together for RMS ISAM files.

GENERIC|NOGENERIC

GENERIC allows partial index retrieval. NOGENERIC prevents partial index retrieval.

Limit: Not valid for IMAGE indexes, unless they are B-Tree or OMNIDEX indexes.

Default: GENERIC

ORDERBY item [ASCENDING|DESCENDING] [,item[ASCENDING|DESCENDING]]...

Allows the ordered retrieval of records in a relational table or view by any column (or combination of columns) defined in the table or view.

If the ORDERBY option occurs with the VIAINDEX option, ordering is performed according to the columns of the ORDERBY option and the ordering imposed by the VIAINDEX option is ignored.

Limit: Valid only for relational structures.

Default: ASCENDING

SEQUENTIAL

Accesses the data record sequentially.

Limit: The SEQUENTIAL and USING options can't be used in the same WHILE RETRIEVING control structure.

Limit: The BACKWARDS and SEQUENTIAL options cannot be used together for RMS ISAM files.

USING expression [,expression]...

Accesses the data records of the specified record-structure using the results of the specified expression as

- the value for corresponding linkitems
- the data record number for record-structures in direct files (**MPE/iX, OpenVMS**) or relative files

- the value of a column in a relational table

For direct files (**MPE/iX, OpenVMS**), or relative files there can be only one value which QUICK interprets as a record number.

For indexed files, or IMAGE datasets, there can be more than one value (for segmented indexes), but QUICK interprets the values as a single index value in that file.

Otherwise, a series of expressions may be specified which correspond one-to-one with the segments established by either the VIA or VIAINDEX options. If neither the VIA nor the VIAINDEX option is specified, and the record-structure has only one associated index structure, this index structure will be used as if the VIAINDEX option had been specified except that index retrieval order will not be enforced.

If a record-structure is a relational table, there can be several values in the USING option which QDESIGN interprets as the values of the columns in the table. The VIA or VIAINDEX options must be used to indicate which columns the values belong to if more than one index is in use or if no index is used.

If the VIA option is specified, the number of expressions specified must correspond one-to-one with the number of segments specified on the VIA option.

If the VIAINDEX option is specified and the VIA option isn't specified, the number of expressions specified may be less than or equal to the number of segments contained within the index structure specified. There must always be at least one expression.

Limit: 255 expressions.

Limit (**MPE/iX**): IMAGE does not support retrieval via an initial subset of the segments of a multi-segment index, unless the index is a B-Tree or OMNIDEX index. An expression must be specified for every segment of the index.

VIAINDEX indexname

Names an index of an indexed file, IMAGE dataset or relational table. When VIAINDEX is used with the USING option, there can be as many USING values as there are segments in the index, or fewer values than the index segments. In the latter case, the values are matched to the index segments in order, starting from the first segment; the leftover segments are not used. When using VIAINDEX, the retrieval always follows the order specified by that index.

Use VIA instead of VIAINDEX with relational tables. By explicitly referencing an index with the VIAINDEX option, it becomes harder to change the database definitions. If the index is deleted, then the source code must be modified. If VIA is used instead, the index can be deleted and the screen continues to work properly.

VIA linkitem [,linkitem]...[ORDERED[ASCENDING|DESCENDING]

Accesses the record-structure via the specified linkitems. A linkitem is a segment of an index for an indexed file or a column in a relational database.

When a VIA list is used in combination with the USING option, there must be a one-to-one match between the USING expressions and VIA linkitems. This option is valid for indexed files, IMAGE datasets, and relational tables only.

For indexed files, and IMAGE datasets, the series of linkitems declared must define a series of segments contained within the index structure associated with the record-structure. In this case, the first linkitem is the first segment within the index structure, the second linkitem is the second segment, and so on.

For relational tables, a series of linkitems may represent any series of columns in a table as long as the VIAINDEX option is not specified. If VIAINDEX is specified, a series of linkitems must be a series of segments contained within a specific index structure: match the first linkitem to the first segment, the second linkitem to the second segment, and so on.

THE ORDERED option allows ordered retrieval of records in a relational table or view by any column or combination of columns defined in a table of view.

The ORDERED option is a convenient method of specifying ORDERBY items when the specified items are the same as those in the VIA list.

If the ORDERED option occurs with the VIAINDEX option, which also imposes an ordering, the ordering is done by the columns of the VIA option. The implicit ordering imposed by the VIAINDEX option is ignored.

Limit: 255 segments.

Limit (**MPE/iX**): IMAGE does not support retrieval via an initial subset of the segments of a multi-segment index, unless the index is a B-Tree or OMNIDEX index. The series of linkitems must include all of the segments in the index.

Discussion

The WHILE RETRIEVING control structure is a repetitive statement used for specialized processing of a related set of data records. The WHILE RETRIEVING control structure allows the designer to specify a control structure that is executed for all data records in a chained retrieval. It creates a looping effect that ends when the end of the chain is reached or when a BREAK verb is encountered. The control structure retrieves data records as specified and executes the code in the procedural statement following it.

Unlike the FOR control structure, which executes the procedural statement a specified number of times (based on the number of repetitions of the file or item), the WHILE RETRIEVING control structure executes the procedural statement once for each data record retrieved. The file that contains the record-structure doesn't have to repeat, since data records are read into the buffer, overwriting the previous data record.

Breaking Out of a WHILE RETRIEVING Control Structure

The WHILE RETRIEVING control structure is executed once for each data record retrieved. The BREAK verb can be used to stop retrieval and the processing of the control structure.

Nesting WHILE RETRIEVING Control Structures

WHILE RETRIEVING control structures can't be explicitly nested. In addition, a WHILE RETRIEVING control structure can't be nested within a FOR control structure. Similarly, a FOR control structure can't be nested within a WHILE RETRIEVING control structure.

The Effect of PUT Verbs in WHILE RETRIEVING Constructs

When using the WHILE RETRIEVING control structure to change indexes down a chain, the PUT verbs must reference a different record buffer and data record pointer, using the ALIAS and OPEN options of the FILE statement. Executing a PUT verb on the same file may change the data record pointer for the next retrieval.

Use caution if you attempt to commit an update inside a WHILE RETRIEVING construct. Committing the retrieval transaction may cause the retrieval to end prematurely.

Retrieving Data Records by Index Values

The VIA option with the SEQUENTIAL option retrieves data records sequentially using the specified key. The VIA option without the SEQUENTIAL option retrieves data records using the specified index, in the sorted order of that index.

Example

The job assignment screen displays the previous jobs held by each employee.

```
> SCREEN JOBASSIGN
>
> FILE DIVISIONS PRIMARY
> FILE EMPLOYEES DETAIL OCCURS 5
Item DIVISION initialized (fixed) to DIVISION OF
DIVISIONS.
> FILE EMPLOYEEDETAIL DESIGNER
>
> TEMPORARY MESSAGE CHARACTER*50 INITIAL &
>      "Previous positions for this employee are: "
```

```
> FIELD DIVISION OF DIVISIONS &
>       REQIRED NOCHANGE LOOKUP NOTON DIVISIONS
> SKIP 2
> title "Name" AT ,10
> TITLE "Position Assigned" AT ,40
> CLUSTER OCCURS WITH EMPLOYEES
>   ALIGN (1,,10) (,,48)
>   FIELD EMPLOYEENUMBER OF EMPLOYEES &
>     REQIRED NOCHANGE LOOKUP NOTON EMPLOYEES
>   FIELD POSITION OF EMPLOYEES
> CLUSTER
> PROCEDURE APPEND
>   BEGIN
>     ACCEPT EMPLOYEENUMBER OF EMPLOYEES
>     WHILE RETRIEVING EMPLOYEEDETAIL &
>       VIA EMPLOYEENUMBER &
>       USING EMPLOYEENUMBER OF EMPLOYEES
>       LET MESSAGE = TRUNC(MESSAGE) + &
>         " " + POSITION OF EMPLOYEEDETAIL
>     INFORMATION = MESSAGE
>     ACCEPT POSITION OF EMPLOYEES
>     END
>
> PROCEDURE ENTRY
>   BEGIN
>     ACCEPT DIVISION
>     FOR EMPLOYEES
>       BEGIN
>         PERFORM APPEND
>         END
>     END
>
> BUILD
```

This while retrieving control structure specifies a procedure that is executed for each data record retrieved from the designer file employeedetail. Data records in the file employeedetail are retrieved by the employeenumber index if they match the employeenumber value in employees. For each data record retrieved, QUICK displays previous position values following the message:

```
Previous positions for this employee are:
```

When the while retrieving control structure is used to process and update related data records in a chain, be careful not to destroy the record pointers required for the next retrieval. The following example allows the user to change the value of the item employee. In both the employees and skills record-structures, the item employee is a segment.

```
> SCREEN EMPCHG ACTIVITIES FIND, CHANGE
> TEMPORARY NEWEMPLOYEE INTEGER
> FILE EMPLOYEES PRIMARY
>   ACCESS VIA EMPLOYEE REQUEST EMPLOYEE
> FILE SKILLS DESIGNER
> FILE SKILLS DESIGNER ALIAS NEWSKILLS OPEN 1
>   ITEM SKILL FINAL SKILL OF SKILLS
> FIELD EMPLOYEE OF EMPLOYEES REQUIRED NOCHANGE &
>   LOOKUP NOTON EMPLOYEES
> FIELD FIRSTNAME OF EMPLOYEES ID SAME
> FIELD LASTNAME OF EMPLOYEES ID SAME
> FIELD NEWEMPLOYEE LOOKUP NOTON EMPLOYEES &
>       VIA EMPLOYEE VALUES 1 TO 9999
> PROCEDURE PREUPDATE
>   BEGIN
>     IF NEWEMPLOYEE = 0
>       THEN ERROR "No update without a valid employee"
>     END

> PROCEDURE UPDATE
>   BEGIN
>     WHILE RETRIEVING SKILLS VIA EMPLOYEE &
>       USING EMPLOYEE OF EMPLOYEES
>       BEGIN
```

```
>         LET EMPLOYEE OF NEWSKILLS = NEWEMPLOYEE
>         PUT NEWSKILLS RESET
>         DELETE SKILLS
>         PUT SKILLS
>       END
>       LET EMPLOYEE OF EMPLOYEES = NEWEMPLOYEE
>       PUT EMPLOYEES
>     END
> BUILD
```

When the value of the segment in the SKILLS record-structure is changed, the data record with the new index value is written by a PUT verb. The data record with the old index value is deleted by another PUT verb. The two PUT verbs for the record-structure must reference different file buffers. The second file buffer is obtained using the ALIAS option of the FILE statement. Forcing a separate open for the second file provides a second set of file pointers. We recommend that you add the new record before deleting the old one because the reverse order may break the WHILE RETRIEVING chain on some file systems.

The following example shows the use of the WHILE RETRIEVING with a cursor:

```
> CURSOR EMPNOTES DESIGNER
.
.
.
>  PROCEDURE DESIGNER CDES HELP &
>      "Copy description to a file"
>  BEGIN
>  LET RECORDCOUNT = 1
>  SQL OPEN EMPNOTES &
>        WHERE(EMPLOYEE=:EMPLOYEE ORDER BY DESCRIPTIONLINE)
>     WHILE RETRIEVING EMPNOTES
>        BEGIN
>        LET cmdline = "echo" + trunc(description) + "'"
>        IF RECORDCOUNT = 1
>        THEN LET cmdline = trunc(cmdline) + ">/tmp/descrip"
>     ELSE LET cmdline = trunc(cmdline) + &
>  ">>/tmp/descrip"
>  RUN COMMAND cmdline
>  LET RECORDCOUNT = RECORDCOUNT + 1
>  END
>  END
```

For an additional example of the WHILE RETRIEVING control structure, see (p. 319).

Chapter 9: Debugger

Overview

This chapter describes the uses of the QUICK Interactive Debugger. It includes information about

- running Debugger
- compiling and running screens with Debugger
- setting breaks in a screen
- getting Help
- displaying source code and finding text in the source code
- controlling the execution of screens
- exiting Debugger
- producing a transcript of a Debugger session

Debugger Overview

Debugger lets you analyze and control QUICK screens as they run. With debugging enabled, QUICK runs screens as if Debugger were not there, until it encounters a break.

QUICK always gives control to Debugger before it executes the first statement of a screen (as long as the screen was compiled for debugging). This allows you to examine the initial state of the screen or to set breaks in it.

A break causes QUICK to pass control to Debugger. Debugger displays the statements around the break, then waits for your commands to display or change the value of an item, list the source statements of the screens, and set and clear breaks.

The QUICK Debugger chapters (9-10) cover these steps in detail. These chapters assume you have experience with QUICK screens. Only Debugger is explained in detail.

Chapter 9 is an introduction organized by topic for those not familiar with Debugger operation.

Chapter 10 describes Debugger options and syntax for all the Debugger commands.

General Terms

This section defines general terms that the Debugger chapters use frequently.

item	A record item declared in the dictionary, a defined item declared in a DEFINE statement, a temporary item declared in a TEMPORARY statement, or a predefined value.
predefined condition	one of ACCESSOK, COMMANDOK, PROMPTOK, ENTRYMODE, FINDMODE, CORRECTMODE, CHANGEMODE, NEWRECORD, ALTEREDRECORD, or DELETEDRECORD. Return status is TRUE if the condition is set to on.
predefined item	one of FIELDTEXT, FIELDVALUE, or PATH
predefined value	A predefined condition, predefined item, or a system function.

system function	one of AUDITSTATUS, PROCESSLOCATION, COMMANDCODE, SCREENLEVEL, COMMANDMESSAGE, OCCURRENCE, SYSNAME, SYSDATE, or SYSTIME.
user break	Ctrl-Y (**MPE/iX**) Ctrl-C (**OpenVMS, UNIX, Windows**)
value	A string or a number depending on the item type.

Running Debugger

To use QUICK Debugger:

1. Compile screens to run in debugging mode.
2. Run QUICK in the debugging mode.
3. Set breakpoints and watchpoints and run screens.

Compiling Screens for Debugger

Debugger needs information about a screen that normally is not included in a compiled screen. To use Debugger, you must start QDESIGN with the **debug** program parameter and compile all the screen you want to debug. To start QDESIGN with debug, enter

```
QDESIGN DEBUG
```

or

```
RUN QDESIGN.CURRENT.COGNOS;INFO="DEBUG"
```
(**MPE/iX**)

When QDESIGN compiles a screen under the debugging mode, it generates two files in the same location as the compiled screen. The files contain the following extra information that Debugger needs

Without the generated files, you are not able to use Debugger on the screen. When compiling a series of QUICK screens using QDESIGN **debug**, QDESIGN ignores SET SAVE CLEAR statements and NODETAIL options. This ensures that the Debugger list file contains the fully-expanded screen source. After building each screen, QDESIGN automatically clears the save file before the next screen compile.

MPE/iX:

Generated Files	Description
screenD	the symbol file that contains the debugging information required for the screen
screenL	the list file that contains the fully expanded source listing

In the above table, "screen" is the first seven characters of the compiled screen's filename.

Eight-character screen names ending with "D" or "L" do not work in debugging mode for QDESIGN. They are truncated and the letter "D" and the letter "L" are appended to create symbol and list file names, respectively.

For example, if you have a compiled screen called SALARIED, Debugger tries to create

```
symbol file:        SALARIE + D = SALARIED
list file:          SALARIE + l = SALARIEL
compiled screen:    SALARIED
```

OpenVMS, UNIX, Windows

Generated Files	Description
screen.qkd	the symbol file that contains the debugging information required for the screen
screen.qkl	the list file that contains the fully expanded source listing

In the above table, "screen" is the compiled screen's filename. For example, compiling a QUICK screen named "entry" produces the following files:

- entry.qkd, the symbol file
- entry.qkl, the list file
- entry.qkc, the compiled QUICK screen

Running Screens with Debugger (MPE/iX)

For Running Screens with Debugger (**OpenVMS, UNIX, Windows**), see (p. 503).

To run QUICK with source-level debugging capability turned on:

```
:QUICK "DEBUG=SOURCE"
```

or

```
:RUN QUICK.CURRENT.COGNOS;INFO="DEBUG=SOURCE"
```

When QUICK is running with debugging enabled, it looks for the debug information in the screenD and screenL files generated by QDESIGN. QUICK uses the location specified in the filespec of the screen being run. For example, if you start QUICK as follows,

```
:QUICK INFO="AUTO=SALES.DEV DEBUG=SOURCE"
```

QUICK looks for all three files in the DEV group.

If there are file equations, QUICK uses them to find each of the three files (compiled screen, symbol, and list files). For example,

```
:FILE SALES=SALES.DEV
:FILE SALESD=SALESD.DEV
:FILE SALESL=SALESL.DEV
```

If you use QKGO to point to a screen, you must have three file equations:

```
:FILE QKGO=EMPLOYEE
:FILE QKGOD=EMPLOYED
:FILE QKGOL=EMPLOYEL
```

As each screen is called, QUICK looks for the screenD and screenL files. If it finds them, the screen runs with debugging enabled. If QUICK does not find the debugging files for a screen, it still runs the screen, but you cannot use Debugger on the screen.

To output Debugger information and accept Debugger commands, Debugger restores the terminal to the configuration it had before QUICK was started. This is true whether running in Character mode or Block mode.

Before starting QUICK for debugging, you must configure your system as you do to run QDESIGN interactively. This is because Debugger requires the same configuration to run as QDESIGN.

Running Screens with Debugger (OpenVMS, UNIX, Windows)

For Running Screens with Debugger (**MPE/iX**), see (p. 503).

To run QUICK with source-level debugging capability turned on, use the **debug=source** program parameter. For example,

```
quick debug=source
```

To debug a screen you do not have to compile higher level screens with **debug**. You need only compile those screens that you want to debug.

When QUICK is running with debugging enabled, it looks for the debug information in the screen.qkd and screen.qkl files generated by QDESIGN.

As each screen is called, QUICK looks for the screen.qkd and screen.qkl files. They must be in the same directory as the compiled screen file. If QUICK finds them, the screen runs with debugging enabled. If QUICK does not find the debugging files for a screen, it still runs the screen, but you cannot use Debugger on the screen.

To output Debugger information and accept Debugger commands, Debugger restores the terminal to the configuration it had before QUICK was started. This is true whether running in Field mode or Panel mode.

Before starting QUICK with Debugger enabled, you must configure your system as you do to run QDESIGN interactively. This is because Debugger requires the same configuration to run as QDESIGN.

Setting Breaks in a Screen

A break causes QUICK to pass control to Debugger. Debugger displays the statements around the break, then waits for your commands to display or change the value of an item, list the source statements of the screens, and set and clear breaks.

There are three ways to set breaks in a screen:

breakpoint

You set a breakpoint on a statement in the screen's source. QUICK then gives control to Debugger just before it executes the statement.

watchpoint

You set a watchpoint on an item or items. QUICK gives control to Debugger just after the value of the item changes.

STEP command

Causes QUICK to execute a certain number of statements, then give control back to Debugger.

In addition, you can force QUICK to pass control to Debugger at any time by typing a user break. This is useful if your screen unexpectedly goes into a long loop, or if you forget to set breaks in a screen before you run it.

Getting Help

For a list of available commands, enter a question mark.

```
> ?
Expected: BREAK  BYE  CLEAR  CONTINUE  DISPLAY  EXIT  FIND  LET  LIST  NEXT
PREVIOUS  QSHOW  SAVE  SCREEN  SHOW  STEP  USE  WATCH  <eol>
```

If you are in the middle of typing a command, enter a question mark to see a list of all expected or allowable entries.

```
> CLEAR ?
Expected: ALL  LINE  <line>
```

Exiting Debugger

The BYE and EXIT commands end the QUICK debugging session without returning to the QUICK screen. You can also end the QUICK Debugger session by exiting QUICK.

Windows: If you have Dr. Watson installed and you exit from the QUICK Debugger rather than from QUICK, you may see a trapped exception error. This is normal because the QUICK Debugger raises an exception in order to exit from within QUICK. If you exit from QUICK rather than the Debugger, no exception is raised.

Continuing Execution

To give control back to QUICK for screen execution, enter

```
> CONTINUE
```

at the Debugger prompt.

The STEP command also causes QUICK to execute some of the screen.

Displaying Source Code

When you first start running a screen compiled for debugging, Debugger clears the terminal window and displays the first 20 lines of screen source code. Thus, before QUICK executes the first statement of a screen you can examine the initial state of the screen or set breaks in it. Debugger displays, for example:

```
0001 SCREEN EMPLOYEE
0002 FILE EMPLOYEE
0003 FILE BILLINGS DESIGNER
0004 FIELD EMPLOYEE OF EMPLOYEE REQUIRED &
0005    LOOKUP NOTON EMPLOYEE VIA EMPLOYEE
0006 FIELD LASTNAME OF EMPLOYEE REQUIRED NOCHANGE
0007 FIELD FIRSTNAME OF EMPLOYEE
0008 FIELD STREET OF EMPLOYEE
0009 FIELD CITY OF EMPLOYEE
0010 FIELD PROVSTATE OF EMPLOYEE
0011 FIELD POSTALCODE OF EMPLOYEE
0012 FIELD PHONE OF EMPLOYEE
0013 FIELD BRANCH OF EMPLOYEE
0014 FIELD DIVISION OF EMPLOYEE
0015 FIELD POSITION OF EMPLOYEE
0016 FIELD SEX OF EMPLOYEE
0017 FIELD LANGUAGE OF EMPLOYEE
0018 FIELD BILINGUAL OF EMPLOYEE
0019 FIELD DATEJOINED OF EMPLOYEE
0020 FIELD BILLING OF BILLINGS DISPLAY
>
```

The Debugger prompt (>) follows the source code. At the prompt, QUICK is waiting for you to enter a Debugger command. (To change the Debugger prompt, use the **prompt** program parameter of QUICK.)

The NEXT, PREVIOUS, and LIST commands are used to display additional QDESIGN source code.

To display the next 20 lines of source code, enter

```
> NEXT
0020  FIELD DATELEFT OF EMPLOYEE
0021  FIELD DATEAPPOINTED OF EMPLOYEE
.
.
.
0038      ACCEPT PROVSTATE OF EMPLOYEE
0039      ACCEPT POSTALCODE OF EMPLOYEE
>
```

For the next 10 lines, enter

```
> NEXT 10
0039      ACCEPT POSTALCODE OF EMPLOYEE
0040      ACCEPT PHONE OF EMPLOYEE
.
.
.
0047      ACCEPT MARITALSTATUS OF EMPLOYEE
0048      ACCEPT FILLER OF EMPLOYEE
>
```

For the previous 20 lines of source code, enter

```
> PREVIOUS
0010  FIELD BRANCH OF EMPLOYEE REQUIRED NOCHANGE
0011  FIELD DIVISION OF EMPLOYEE REQUIRED NOCHANGE
.
.
.
0028  PROCEDURE ENTRY
0029    BEGIN
>
```

For the previous 15 lines, enter

```
> PREVIOUS 15
0001  SCREEN EMPLOYEE
0002  FILE EMPLOYEE
.
.
.
0014  FIELD LASTNAME OF EMPLOYEE REQUIRED NOCHANGE
0015  FIELD BRANCH OF EMPLOYEE REQUIRED NOCHANGE
>
```

For 20 lines of source code with the current breakpoint position centered on the screen (if possible), enter

```
> LIST
0001  SCREEN EMPLOYEE
0002  FILE EMPLOYEE
.
.
.
0019  FIELD DATEJOINED OF EMPLOYEE
0020  FIELD DATELEFT OF EMPLOYEE
>
```

For a specified range of lines (20-30 in this case), enter

```
> LIST 20 TO 30
0020  FIELD DATELEFT OF EMPLOYEE
0021  FIELD DATEAPPOINTED OF EMPLOYEE
.
.
.
0029    BEGIN
0030      ACCEPT EMPLOYEE OF EMPLOYEE
>
```

For all lines of the current screen, enter

```
> LIST ALL
```

Finding Text in the Source Code

Use the FIND command to find specified text in the current screen. To find lines that have the string "employee", enter

```
> FIND employee
```

Debugger searches from the beginning of the screen until "employee" is found in any combination of upper and lowercase.

Debugger displays 20 lines of source code centered on the line that contains the text. An "F" beside the line number identifies the first line that contains the text.

```
0001F SCREEN EMPLOYEE
0002  FILE EMPLOYEE
0003
0004  TITLE "Employee Screen" CENTERED
0005  SKIP 5
0006  ALIGN (1,4,21) (41,44,61)
.
.
```

.

Enter FIND with no arguments to look for the same text again, starting from the line that is marked "F".

```
> FIND
0001  SCREEN EMPLOYEE
0002F FILE EMPLOYEE
0003
0004  TITLE "Employee Screen" CENTERED
0005  SKIP 5
0006  ALIGN (1,4,21) (41,44,61)
.
.
.
```

For exact text matches, including the case of the letters, put quotation marks around the text. For example, to find an uppercase "S" followed by lowercase "creen", enter

```
> FIND "Screen"
0001  SCREEN EMPLOYEE
0002  FILE EMPLOYEE
0003
0004F TITLE "Employee Screen" CENTERED
0005  SKIP 5
0006  ALIGN (1,4,21) (41,44,61)
.
.
.
```

You can also search for text that matches a PowerHouse pattern. To find "employee" at the end of a line, use FIND PATTERN with the PowerHouse pattern-matching character at-sign (@) before the string you want to find. For example,

```
> FIND PATTERN "@EMPLOYEE"
0001F SCREEN EMPLOYEE
0002  FILE EMPLOYEE
0003
0004  TITLE "Employee Screen" CENTERED
.
.
.
```

You must put quotation marks around the pattern. Patterns must represent the complete line. For example, to find lines beginning with EMPLOYEE (the string EMPLOYEE followed by any other characters), enter

```
> FIND PATTERN "EMPLOYEE@"
```

To find lines containing EMPLOYEE (the string EMPLOYEE surrounded by any other characters), enter

```
> FIND PATTERN "@EMPLOYEE@"
```

You can also search for text with a PowerHouse SOUNDEX string. For more information on pattern matching and the SOUNDEX function, see Chapter 5, "PowerHouse Language Rules", and Chapter 6, "Functions in PowerHouse", respectively, in the *PowerHouse Rules* book.

For all variations of FIND, you can restrict the search to a range of lines. After the string, pattern, or SOUNDEX string, enter the start and end line numbers.

```
> FIND "Then" 10 TO 70
```

Controlling Execution

QUICK Debugger provides control of screen execution with

- breakpoints
- stepping
- watchpoints
- user Breaks

When QUICK encounters one of these breaks, it passes control over to Debugger so you can examine the screen. Then you can resume execution of the screen until another break is encountered, or the screen completes execution.

Breakpoints

You can set a breakpoint at any line in the screen. When QUICK encounters a breakpoint, it gives control to Debugger before executing the breakpoint statement.

Typically, you would set a breakpoint at the start of code that you suspect may be causing a problem. Then you use a CONTINUE command to run the screen. QUICK gives control back to Debugger at the breakpoint and you examine the state of the screen and the values of items to determine the source of the problems.

For example, to set a breakpoint at line 34, enter

```
> BREAK 34
Set at line: 34
>
```

To see the line numbers of all breakpoints in the current screen, enter

```
> BREAK
Set at line: 30
Set at line: 34
>
```

Once a breakpoint is set, QUICK gives control to Debugger every time it reaches the statement. To remove a breakpoint, enter CLEAR followed by the line number of the breakpoint.

```
> CLEAR 34
Breakpoint cleared at line: 34
>
```

Stepping

With the STEP command, QUICK executes one or more statements and then gives control back to Debugger as if it had encountered a breakpoint. To execute just the next statement, enter

```
> STEP
```

To execute the next 5 statements, enter

```
> STEP 5
```

Typically, you use breakpoints to get close to suspect code, then use STEP to follow every statement until the problem is found.

Watchpoints

A watchpoint causes QUICK to give control to Debugger whenever the value of a specified item changes. This gives you an easy way to find the statement that unexpectedly changes the value of an item.

You can set a watchpoint on a record item, a defined item, or a predefined item. To set a watchpoint on the item LASTNAME, enter

```
> WATCH LASTNAME
Watchpoint is set for LASTNAME OF EMPLOYEE
>
```

You can set a watchpoint on all the items in a file by giving a file name instead of an item name, as in

```
> WATCH FILE EMPLOYEE ON
Watchpoint is set for EMPLOYEE OF EMPLOYEE
Watchpoint is set for LASTNAME OF EMPLOYEE
Watchpoint is set for FIRSTNAME OF EMPLOYEE
Watchpoint is set for ADDRESS OF EMPLOYEE
Watchpoint is set for STREET OF EMPLOYEE
Watchpoint is set for CITY OF EMPLOYEE
Watchpoint is set for PROVSTATE OF EMPLOYEE
```

```
Watchpoint is set for POSTALCODE OF EMPLOYEE
Watchpoint is set for PHONE OF EMPLOYEE
Watchpoint is set for DATEJOINED OF EMPLOYEE
Watchpoint is set for DATELEFT OF EMPLOYEE
Watchpoint is set for BRANCH OF EMPLOYEE
Watchpoint is set for DIVISION OF EMPLOYEE
Watchpoint is set for POSITION OF EMPLOYEE
Watchpoint is set for DATEAPPOINTED OF EMPLOYEE
Watchpoint is set for NOOFAPPTS OF EMPLOYEE
Watchpoint is set for LANGUAGE OF EMPLOYEE
Watchpoint is set for SEX OF EMPLOYEE
Watchpoint is set for MARITALSTATUS OF EMPLOYEE
Watchpoint is set for FILLER OF EMPLOYEE
>
```

QUICK predefined items are items that contain information about the state of records or of QUICK. To set a watchpoint on a predefined item, enter

```
> WATCH PREDEFINED NEWRECORD OF EMPLOYEE ON
Watchpoint is set for NEWRECORD OF EMPLOYEE.
>
```

For the predefined values AUDITSTATUS, ALTEREDRECORD, DELETEDRECORD, and NEWRECORD, you must specify a file with an OF option.

To turn a watchpoint off, enter

```
> WATCH LASTNAME OFF
Watchpoint cleared for LASTNAME OF EMPLOYEE.
>
```

Due to the interaction between QUICK and Debugger, Debugger sometimes does not gain control immediately after a watched item changes. This is because QUICK sometimes changes the value of items internally, such as when it initializes a defined item. In this case, Debugger may display a line for a watchpoint that has no apparent connection with the watched item.

User Break

Any time you enter a user break [Ctrl-Y (**MPE/iX**) or Ctrl-C (**OpenVMS, UNIX, Windows**)] during the execution of a screen compiled for debugging, QUICK gives control to Debugger, even when you do not have breakpoints set.

Again, because of the way QUICK and Debugger interact, Debugger sometimes does not gain control immediately after a user break. Often a user break must be typed with something else (find a record, for example) to make QUICK pass control to Debugger.

The Screen Environment

QUICK preserves the debugging environment of the screen as you go from screen to screen. When you leave a higher level screen for a subscreen, QUICK maintains all breakpoints for the higher level screen.

During a debugging session, all commands refer to the current debugging environment. Normally this is the current screen. If you want to look at any other active screen, you have to change the current environment.

To display the active screens, enter

```
> SCREEN
```

Debugger displays a message that looks like this:

```
QUICK813c L1 MAINMENU(98/02/23) QDESIGN813c HP2392 (c) COGNOS INCORPORATED.
QUICK813c L2 EMPLOYEE(98/02/23) QDESIGN813c HP2392 (c) COGNOS INCORPORATED.
>
```

"L1" and "L2" are the level numbers of the screen. You can refer to a screen by either its level number or its name.

To change the current environment to the environment of another active screen, enter a screen name or level number. For example, to make the MAINMENU screen the current screen, enter

```
> SCREEN 1
```

or

```
> SCREEN MAINMENU
```

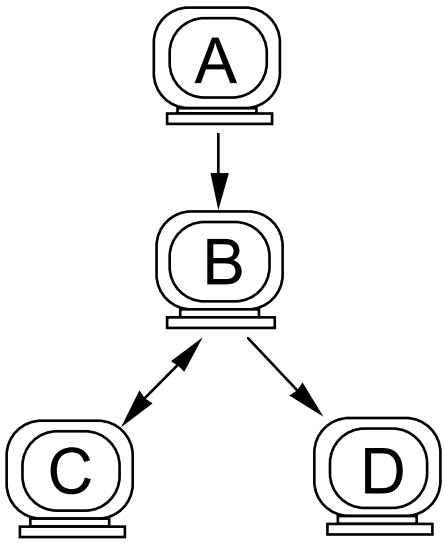
From now until you change environments again, all commands refer to the MAINMENU screen.

When you list the source code of a screen, the next line that QUICK executes is highlighted. If you change environments and list source code, you may still see a highlighted line. If the new environment is the environment of a higher level screen, the highlighted line will be the line that called a subscreen.

When you enter CONTINUE or STEP, QUICK automatically restores the current environment to the environment that was running when the break occurred.

The list of active screens a SCREEN command produces includes the current screen and all the screens above it, plus all its active descendants. A screen's active descendants are

- the last screen that returned to the current screen
- the active descendants of that screen



If screen A calls B which calls C, then C returns to B and B calls D, then the descendant of B is D only, not C and D.

Transcript of the Debugging Session

If the file qkdebug exists when you start QUICK in one of the debugging modes, QUICK writes the screen information and line numbers executed during the session to the file. This enables you to follow the execution of your screen on a line-by-line basis.

OpenVMS: QKDEBUG is a logical name equated to an existing file. For example, before invoking QUICK, create DBGOUT.TXT (an empty file) and equate it to the logical name QKDEBUG, as follows:

```
define QKDEBUG DBGOUT.TXT
```

Used with qkecho (which contains terminal input for the session) and qkout (which contains terminal output for the session), qkdebug is an effective way of tracing the flow of control through an application.

Chapter 10: Debugger Commands

Overview

This chapter describes each of Debugger's commands.

- syntax summaries
- detailed syntax descriptions
- setting breakpoints to stop execution of a QUICK screen
- stepping through source code
- stopping execution of a screen when an item value changes
- displaying the contents of a record, defined, or temporary item
- listing the screen source code

Debugger Command Summary

The following table lists QUICK Debugger commands and briefly describes what they do:

Command	Purpose
BREAK	Sets or displays a breakpoint.
BYE	Exits Debugger and QUICK.
CLEAR	Clears breakpoints.
CONTINUE	Continues execution of a screen after Debugger has gained control for any reason.
DISPLAY	Displays values of items and predefined items.
EXAMINE	Displays values of items and predefined items.
EXIT	Exits Debugger and QUICK.
FIND	Searches for text in the source code.
GO	Continues execution of the screen after Debugger has gained control for any reason.
LET	Sets the value of an item.
LIST	Displays either a range of lines of source code or all lines on the current screen.
NEXT	Displays either the specified lines or the next page of source code.
PREVIOUS	Displays either the specified number of lines or the previous page of source code.
QSHOW	Runs QSHOW to display information from the dictionary.
SAVE	Saves Debugger commands to a file.

Command	Purpose
SCREEN	Lists Debugger environments or changes the current environment.
SCROLL	Displays source code.
SHOW	Shows information about items, record-structures, or screens.
STEP	Executes one or more statements of a screen.
TYPE	Displays either a range of lines of source code or all lines on the current screen.
USE	Reads and executes Debugger commands from a file.
User Break	Breaks to Debugger from QUICK.
WATCH	Sets or displays a watchpoint.

BREAK

Sets or displays a breakpoint.

Syntax

BREAK [[AT] [LINE] n]

AT

For documentation only.

LINE

For documentation only.

n

A line number. There is no warning if you enter a non-existent line number.

Discussion

A BREAK command with a line number sets a breakpoint at that line.

To set a breakpoint at a source statement, provide the line number of the statement.

```
> BREAK 33
Set at line: 33
>
```

The BREAK command without a line number shows all the line numbers in the current screen that have breakpoints.

To display all breakpoints in the current screen, enter BREAK alone.

```
> BREAK
Set at line: 33
Set at line: 52
>
```

When QUICK reaches a breakpoint, it gives control to Debugger. Debugger gains control at a point immediately before execution of the statement marked as a breakpoint.

Debugger replaces the QUICK screen display with a listing of the ten source statements before the breakpoint line and the nine source statements after the breakpoint line. Debugger then waits for the entry of other Debugger commands. Use Debugger commands to display the values of items, set and clear breakpoints, and so on.

To see the context of breakpoints, use the LIST command.

```
> LIST 30 TO 40
0030   ACCEPT FIRSTNAME OF EMPLOYEE
0031   ACCEPT STREET OF EMPLOYEE
0032   ACCEPT CITY OF EMPLOYEE
0033B  ACCEPT PROV-STATE OF EMPLOYEE
0034   ACCEPT POSTALCODE OF EMPLOYEE
0035   ACCEPT PHONE OF EMPLOYEE
0036   ACCEPT BRANCH OF EMPLOYEE
0037   ACCEPT DIVISION OF EMPLOYEE
0038   ACCEPT POSITION OF EMPLOYEE
0039   ACCEPT SEX OF EMPLOYEE
0040   ACCEPT LANGUAGE OF EMPLOYEE
>
```

Enter CONTINUE or STEP to continue executing the QUICK screen after a breakpoint.

Whenever Debugger lists source statements, a "B" appears to the right of the line numbers that are breakpoints.

In most cases, breakpoints should be set on procedure code that gets executed, not constructs such as PROCEDURE, BEGIN or END, which are intended for the QDESIGN parser. If breakpoints are set on non-procedural QDESIGN statements (like FIELD), the breakpoint may not occur where planned.

BYE

Exits Debugger and QUICK.

Syntax

BYE

Discussion

BYE terminates the QUICK Debugger and the QUICK screen session, and returns control to the operating system or the invoking program.

BYE and EXIT are interchangeable.

CLEAR

Clears breakpoints.

Syntax

CLEAR [LINE] n

CLEAR ALL

LINE

For documentation only.

n

A line number.

ALL

Clears all breakpoints in the current screen.

Discussion

The CLEAR command with a line number removes the breakpoint at that line.

```
> CLEAR LINE 421
Breakpoint cleared at line: 421
>
```

Specifying a line that has no breakpoint set results in a message, as in:

```
> CLEAR LINE 5310
No breakpoint set at line: 5310
```

The CLEAR ALL command removes all breakpoints in the current screen.

```
> CLEAR ALL
Breakpoint cleared at line: 113
Breakpoint cleared at line: 297
Breakpoint cleared at line: 421
>
```

CONTINUE

Continues execution of a screen after Debugger has gained control for any reason.

Syntax

CONTINUE

Discussion

Starts or restarts execution of a QUICK screen after Debugger has gained control for any reason. QUICK continues executing screens until

- QUICK reaches a breakpoint
- a variable monitored by the WATCH command changes
- a user break is entered
- the user leaves QUICK

STEP is another command for controlling screen execution.

CONTINUE and GO are interchangeable.

DISPLAY

Displays values of items and predefined items.

Syntax

DISPLAY [ITEM] item [OF [FILE] record-structure]
[(occurrence)]

DISPLAY FILE record-structure [(occurrence)]

DISPLAY PREDEFINED [predefined-value
[OF [FILE] record-structure] [(occurrence)]]

[ITEM] item [OF [FILE] record-structure]

The item name to display and, optionally, the record-structure that the item is from.

FILE record-structure

Displays all items associated with the named record-structure.

PREDEFINED [predefined-value [OF [FILE] record-structure]]

Indicates a specific predefined value to display, and, optionally, the record-structure the predefined value is for. Predefined values include predefined items, system functions, and predefined conditions. To display AUDITSTATUS, NEWRECORD, ALTEREDRECORD, and DELETEDRECORD, display each one by name and specify the record-structure, or use the SHOW FILE command.

(occurrence)

Specifies which occurrence of an item or a record-structure in the cache to display.

Discussion

The result of the display is the current internal representation of the value assigned to the item or items. To see the screen representation, use the SHOW SCREEN IMAGE command.

If an item is specified with no FILE qualifier, Debugger checks to see if the item is the name of a temporary or defined item. If it is, Debugger displays the value. If not, Debugger checks to see if the item is the name of a record item in each record-structure in the order that the record-structures were declared in the FILE statements of the screen. The value of the first such record item is then displayed.

The following example shows how to display the contents of the item EMPLOYEE:

```
> DISPLAY ITEM EMPLOYEE
EMPLOYEE OF EMPLOYEE =  1
>
```

If a record-structure occurs more than once in a screen, and it contains items that occur more than once, then the occurrence specification applies to the record-structure and the DISPLAY command always displays the first occurrence of the item in the cache.

The DISPLAY FILE command displays all items in the record-structure.

The following example shows how to display the fourth occurrence of the SKILLS file:

```
> DISPLAY FILE SKILLS (4)
EMPLOYEE OF SKILLS (4) =  1
SKILL OF SKILLS (4) = "Cobol      "
>
```

All character items are fully displayed between quotation marks, including leading and trailing spaces.

To view the storage type information, use the SHOW command.

QUICK recalculates the value of a defined item any time the value is accessed. The DISPLAY command shows the value of an item as last calculated by QUICK, but does not itself cause the value to be recalculated. To see the current value of a defined item, set a breakpoint immediately after a statement that accesses the item, or set a watchpoint on the item.

The DISPLAY PREDEFINED command displays the value of all the predefined values except AUDITSTATUS, NEWRECORD, ALTEREDRECORD, and DELETEDRECORD. In the following example:

- DISPLAY PREDEFINED displays all the predefined values.
- Include the name of a predefined value to display its value alone.
- To display AUDITSTATUS, NEWRECORD, ALTEREDRECORD, and DELETEDRECORD, display each one by name and specify the record-structure.

```
> DISPLAY PREDEFINED
   ACCESSOK = FALSE
   FIELDTEXT = <This item has no assigned value>
   FIELDVALUE = <This item has no assigned value>
   PROMPTOK = FALSE
   PATH = 0
   OCCURRENCE = 1
   SYSNAME = "Future Industries Staff System          "
   SYSTIME = 8594200
   SYSDATE = 931028
   SCREENLEVEL = 1
   PROCESSLOCATION = <This item has no assigned value>
   COMMANDOK = FALSE
   COMMANDCODE = <This item has no assigned value>
   COMMANDMESSAGE = <This item has no assigned value>
   STATUSTEXT = <This item has no assigned value>
   ENTRYMODE = FALSE
   CORRECTMODE = FALSE
   FINDMODE = FALSE
   SELECTMODE = FALSE
   CHANGEMODE = FALSE
   OSACCESS = FALSE
>
> DISPLAY PREDEFINED ALTEREDRECORD OF EMPLOYEE
ALTEREDRECORD = TRUE
>
```

The SHOW command displays information about items, record-structures, screens, and predefined values.

Notes: If a character item contains unprintable values while in 7-bit display mode, the unprintable values are shown as octal values enclosed between the brackets "<" and ">". For example, an ASCII formfeed character displays as "<14>". The display mode (ASCII7 or ASCII8) is determined by the ASCII7/ASCII8 flag in the dictionary.

DISPLAY and EXAMINE are interchangeable.

EXAMINE

Displays values of items and predefined items.

Syntax

EXAMINE [ITEM] item [OF [FILE] record-structure]
[(occurrence)]

EXAMINE FILE record-structure [(occurrence)]

EXAMINE PREDEFINED [predefined-value
[OF [FILE] record-structure] [(occurrence)]]

[ITEM] item [OF [FILE] record-structure]

The item name to EXAMINE and, optionally, the record-structure that the item is from.

(occurrence)

Specifies which occurrence of an item or a record-structure in the cache to display.

FILE record-structure

Displays all items associated with the named record-structure.

PREDEFINED [predefined-value [OF [FILE] record-structure]]

Indicates that you want to display specific predefined values, and, optionally, the record-structure that the item is from. Predefined values include predefined items, system functions, and predefined conditions. To display AUDITSTATUS, NEWRECORD, ALTEREDRECORD, and DELETEDRECORD, display each one by name and specify the record-structure, or use the SHOW FILE command.

Discussion

The result of the display is the current internal representation of the value assigned to the item or items. To see the screen representation, use the SHOW SCREEN IMAGE command.

If an item is specified with no FILE qualifier, Debugger checks to see if the item is the name of a temporary or defined item. If it is, Debugger displays the value. If not, Debugger checks to see if the item is the name of a record item in each record-structure in the order that the record-structures were declared in the FILE statements of the screen. The value of the first such record item is then displayed.

The following example shows how to examine the contents of the item EMPLOYEE:

```
> EXAMINE ITEM EMPLOYEE
EMPLOYEE OF EMPLOYEE =  1
>
```

If a record-structure occurs more than once in a screen, and it contains items that occur more than once, then the occurrence specification applies to the record-structure and the EXAMINE command always displays the first occurrence of the item in the cache.

The EXAMINE FILE command displays all items in the record-structure.

The following example shows how to examine the fourth occurrence of the SKILLS file:

```
> EXAMINE FILE SKILLS (4)
EMPLOYEE OF SKILLS (4) =  1
SKILL OF SKILLS (4) = "Cobol      "
>
```

All character items are fully displayed between quotation marks, including leading and trailing spaces.

To view the storage type information, use the SHOW command.

QUICK recalculates the value of a defined item any time the value is accessed. The EXAMINE command shows the value of an item as last calculated by QUICK, but does not itself cause the value to be recalculated. To see the current value of a defined item, set a breakpoint immediately after a statement that accesses the item, or set a watchpoint on the item.

The EXAMINE PREDEFINED command displays the value of all the predefined values except AUDITSTATUS, NEWRECORD, ALTEREDRECORD, and DELETEDRECORD. In the following example:

- EXAMINE PREDEFINED examines all the predefined values.
- Include the name of a predefined value to display its value alone.
- To display AUDITSTATUS, NEWRECORD, ALTEREDRECORD, and DELETEDRECORD, display each one by name and specify the record-structure.

```
> EXAMINE PREDEFINED
ACCESSOK = FALSE
FIELDTEXT = <This item has no assigned value>
FIELDVALUE = <This item has no assigned value>
PROMPTOK = FALSE
PATH = 0
OCCURRENCE = 1
SYSNAME = "Future Industries Staff System         "
SYSTIME = 8594200
SYSDATE = 931028
SCREENLEVEL = 1
PROCESSLOCATION = <This item has no assigned value>
COMMANDOK = FALSE
COMMANDCODE = <This item has no assigned value>
COMMANDMESSAGE = <This item has no assigned value>
STATUSTEXT = <This item has no assigned value>
ENTRYMODE = FALSE
CORRECTMODE = FALSE
FINDMODE = FALSE
SELECTMODE = FALSE
CHANGEMODE = FALSE
OSACCESS = FALSE
>
> EXAMINE PREDEFINED ACCESSOK
ACCESSOK = TRUE
>
> EXAMINE PREDEFINED ALTEREDRECORD OF EMPLOYEE
ALTEREDRECORD = TRUE
>
```

The SHOW command displays information about items, record-structures, screens, and predefined values.

Note: If a character item contains unprintable values while in 7-bit display mode, the unprintable values are shown as octal values enclosed between the brackets "<" and ">". For example, an ASCII formfeed character displays as "<14>". The display mode (ASCII7 or ASCII8) is determined by the ASCII7/ASCII8 flag in the dictionary.

EXAMINE and DISPLAY are interchangeable.

EXIT

Exits Debugger and QUICK.

Syntax

EXIT

Discussion

EXIT terminates the QUICK Debugger and the QUICK screen session, and then returns control to the operating system or the invoking program.

EXIT and BYE are interchangeable.

FIND

Searches for text in the source code.

Syntax

FIND

FIND string|"string" [n1 [[TO] n2]]

FIND PATTERN "pattern" [n1 [[TO] n2]]

FIND SOUNDEX "string" [n [n1 [[TO] n2]]]

string|"string"

Specifies that you want to find a string. When the string is in quotation marks, the search is made for an exact match (upper and lowercase match).

[n1 [[TO] n2]]

The range of lines you want to search. TO is for documentation only.

A line number range causes the FIND command to search only the inclusive limits n1 to n2. If a range is not specified, then n1 is 1 and n2 is the last line of the file. If n2 is not specified, then n2 is the last line of the file.

PATTERN "pattern"

Specifies that you want to find a pattern. The pattern must be enclosed in quotation marks and must match a complete line.

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

SOUNDEX "string"

Specifies that you want to find a string that sounds similar to the specified string, which must be enclosed in quotation marks.

Note that the syntax for the SOUNDEX function in Debugger does not include brackets ( ). For information about the SOUNDEX function, see Chapter 6, "Functions in PowerHouse", in the *PowerHouse Rules* book.

n

Specifies the size of the SOUNDEX result.

Discussion

When the string has no quotation marks, Debugger ignores the letter case (lowercase a equals uppercase A). The string is called a case- insensitive string (ci-string).

In the following example, when a search is successful, the line number is marked with an "F" and the source is centered in the terminal window.

```
> FIND position
0002  FILE EMPLOYEE
0003
.
.
.
0011  FIELD BRANCH OF EMPLOYEE REQUIRED NOCHANGE
0012  FIELD DIVISION OF EMPLOYEE REQUIRED NOCHANGE
0013F FIELD POSITION OF EMPLOYEE REQUIRED NOCHANGE
0014  FIELD FIRSTNAME OF EMPLOYEE
0015  FIELD STREET OF EMPLOYEE
```

```
.
.
.
>
```

The FIND "string" with quotation marks treats upper and lowercase letters differently: a lowercase "a" is not the same as an uppercase "A".

For example, to search lines 1 to 25 for a match on the string "employee", enter:

```
> FIND "employee" 1 TO 25
```

The FIND command with no options repeats the last search starting at the current line and searches to the last line of the screen.

To search again for "employee" starting from the current line, enter:

```
> FIND
```

GO

Continues execution of a screen after Debugger has gained control for any reason.

Syntax

GO

Discussion

Starts or restarts execution of a QUICK screen after

- QUICK reaches a breakpoint
- a variable monitored by the WATCH command changes
- a user break is entered
- the user leaves QUICK

STEP is another command for controlling screen execution.

GO and CONTINUE are interchangeable.

LET

Sets the value of an item.

Syntax

LET [ITEM] item [OF [FILE] record-structure]
[(occurrence)] = value|TRUE|FALSE

LET PREDEFINED predefined-value [OF [FILE]
record-structure] [(occurrence)] = value|TRUE|FALSE

[ITEM] item [OF [FILE] record-structure]

An item to assign a value to and, optionally, the record-structure that the item is from.

(occurrence)

Specifies which occurrence to assign the value to.

value

The value assigned to the named item. If the item is a character datatype, you must enclose the value in quotation marks.

TRUE|FALSE

Used to assign a TRUE or FALSE status to a predefined item; a Y (Yes) or N (No) to a character item; or a 1 or 0 to a numeric item.

PREDEFINED [predefined-value [OF [FILE] record-structure]

Used to assign a value to a predefined value, and, optionally, the record-structure the predefined value is for. You can use the LET PREDEFINED command to change the value of the following predefined values:

ACCESSOK	ALTEREDRECORD	COMMANDOK
DELETEDRECORD	FIELDTEXT	NEWRECORD
PATH	PROMPTOK	

For the predefined values ALTEREDRECORD, DELETEDRECORD, and NEWRECORD, a record-structure must be specified.

Discussion

The LET command is a convenient way to control the route QUICK takes through the screen code. Item values can be changed to direct the route through the screen.

The following example:

- Assigns a value of Smith to the item LASTNAME.
- Assigns TRUE to the character item BILINGUAL to give it a value of Y.

```
> LET ITEM LASTNAME OF FILE EMPLOYEE = "Smith"
Changing LASTNAME OF EMPLOYEE.
Old Value = "<0><0><0><0><0><0><0><0><0><0><0><0><0>"
New Value = "Smith        "
> LET BILINGUAL = TRUE
Changing BILINGUAL OF POSITIONS.
Old Value = "  "
New Value = "Y "
```

When the value of a record item is changed with the LET command, the status of the record buffer (ALTEREDRECORD) is not automatically changed. The record is only written to the file if you do an update action and ALTEREDRECORD is TRUE. If QUICK changes any values in the record, then ALTEREDRECORD is set to TRUE, and the value assigned to the item with the LET command is written to the file.

If QUICK does not change any values, you can force your change to be written to a file by using the LET command to set ALTEREDRECORD to TRUE. Values of items to be written to a file can be forced by assigning the appropriate value to predefined items.

```
> LET PREDEFINED ALTEREDRECORD OF EMPLOYEE = TRUE
Changing ALTEREDRECORD OF EMPLOYEE.
Old Value = FALSE
New Value = TRUE
>
```

LIST

Displays either a range of lines of source code or all lines on the current screen.

Syntax

LIST [[n [[TO] m]]|ALL]

[n [[TO] m]]

Range of lines to list. TO is for documentation only.

ALL

Lists all source statements of the current screen.

Discussion

If no range is specified, the LIST command displays the source code around the statement that QUICK was executing when Debugger gained control. The display is centered on the terminal window.

The following example displays the statements before and after the statement that QUICK was executing when Debugger gained control.

```
> LIST
0001  ; Positions file screen
0002  SCREEN POSITION
.
.
.
0019      REQUEST POSITION OF POSITIONS
0020      IF PROMPTOK
>
```

If you specify a line range (n TO m) the LIST command lists source code statements only within that range.

If m is missing, Debugger prints 20 lines centered around n.

The LIST command ignores invalid line numbers.

In the following example, LIST 15 lists 20 lines around line 15 and LIST 100 TO 120 displays lines 100 to 120.

```
> LIST 15
0005    LOOKUP NOTON EMPLOYEE VIA EMPLOYEE
0006 FIELD LASTNAME OF EMPLOYEE REQUIRED NOCHANGE
0007 FIELD FIRSTNAME OF EMPLOYEE
0008 FIELD STREET OF EMPLOYEE
0009 FIELD CITY OF EMPLOYEE
0010 FIELD PROV-STATE OF EMPLOYEE
0011 FIELD POSTALCODE OF EMPLOYEE
0012 FIELD PHONE OF EMPLOYEE
0013 FIELD BRANCH OF EMPLOYEE
0014 FIELD DIVISION OF EMPLOYEE
0015 FIELD POSITION OF EMPLOYEE
0016 FIELD SEX OF EMPLOYEE
0017 FIELD LANGUAGE OF EMPLOYEE
0018 FIELD BILINGUAL OF EMPLOYEE
0019 FIELD DATEJOINED OF EMPLOYEE
0020 FIELD BILLING OF BILLINGS DISPLAY
0021 FIELD DATE-APPOINTED OF EMPLOYEE
0022
0023
0024
> LIST 100 TO 120
0100      THEN GET...
0101    IF PATH = 5
```

```
.
.
.
0119    PUT EMPLOYEE
0120    END
>
```

The LIST ALL command lists all the source code statements in the current screen.

```
> LIST ALL
```

LIST and TYPE are interchangeable.

NEXT

Displays either the specified lines or the next page of source code.

Syntax

NEXT [n]

n

The number of statement lines that you want to display.

Default: 20

Discussion

The NEXT command with no number displays the next page of source code. The next page includes the last line of the previous listing.

To display the next page, enter

```
> NEXT
```

The NEXT command with a number displays n lines instead of 20 lines.

To display the next 10 lines, enter

```
> NEXT 10
```

NEXT and SCROLL DOWN are interchangeable.

PREVIOUS

Displays either the specified lines or the previous page of source code.

Syntax

PREVIOUS [n]

n

The number of statement lines that you want to display.

Default: 20

Discussion

The PREVIOUS command with no number displays the previous page of source code.

To display the previous page, enter

```
> PREVIOUS
```

The PREVIOUS command with a number displays n lines instead of 20 lines.

To display the previous 10 lines, enter

```
> PREVIOUS 10
```

PREVIOUS and SCROLL UP are interchangeable.

QSHOW

Runs QSHOW to display information from the dictionary.

Syntax

QSHOW

Discussion

QSHOW lets users make inquiries about elements and files in the dictionary. QSHOW is enabled when the prompt (QSHOW>) appears.

To run QSHOW, enter

```
> QSHOW
```

For detailed information about QSHOW, see Chapter 4, "QSHOW Statements", in the *PDL and Utilities Reference* book.

SAVE

Saves Debugger commands to a file.

Syntax

SAVE filespec [CLEAR]

filespec

The specification for the file where Debugger commands are to be saved.

OpenVMS, UNIX, Windows: The default extension is ".qkl".

CLEAR

The CLEAR option erases the command history after saving the history in the specified file.

Discussion

The SAVE command saves the current command history in the file named by filespec. It contains all Debugger commands entered since Debugger last gained control, or since the last SAVE command.

Use the SAVE command to save commands that are often repeated; reuse saved commands with the USE command.

In the following example the SAVE command is used to save the current history in a file called COMMANDS and clear the command history.

```
> SAVE COMMANDS CLEAR
```

SCREEN

Lists Debugger environments or changes the current environment.

Syntax

SCREEN [ENVIRONMENT] [name|level]

ENVIRONMENT

For documentation only.

name|level

You can specifies which screen to make the current screen by the screen's name or level number.

Discussion

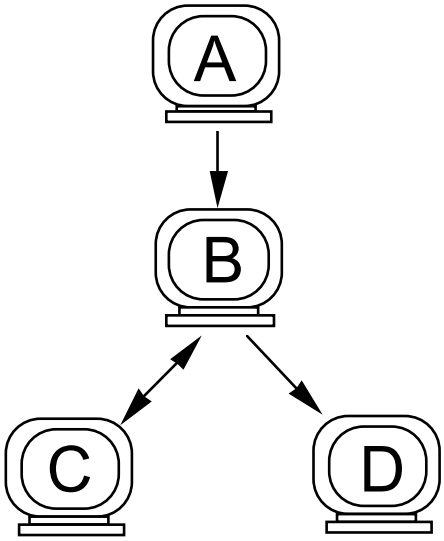
The SCREEN command (without name or level options) lists all Debugger environments in the environment list, starting with the top level and ending with the lowest level. (Use SHOW SCREEN INFO to show which level is the current environment.)

In the following example, the SCREEN command shows a list of all active screen environments.

```
> SCREEN
QUICK840E L1 MAINMENU.QKC(2006/11/30) QDESIGN840E VT100 (c) COGNOS
INCORPORATED
```

The fields ...	represent ...
QUICK840E	QUICK version number
L1	screen level number
MAINMENU.QKC (2006/11/30)	screen name and date compiled
QDESIGN840E	QDESIGN version that compiled the screen
VT100	terminal type
(c) COGNOS INC.	copyright notice

The environment list contains all the screens compiled with the Debugger option that QUICK executed to get to the current screen, and all the screens below the current screen that returned back to the current screen.



The environment list is a chain, not a tree. If A calls B and B calls C, then C returns to B and B calls D, then D returns to B, then the environment list will be A, B, D.

The SCREEN name command and the SCREEN level command change the current environment to the named screen or level. All Debugger commands then refer to the named screen environment. The CONTINUE and STEP commands always resume from the line where Debugger gained control, and also reset the environment to the environment of the current line.

In the following example:

- The SCREEN TEST command is used to set the screen environment to screen TEST.
- The SCREEN 2 command is used to set the screen environment to screen level 2.

```
> SCREEN TEST
>
> SCREEN 2
>
```

The SCREEN command is useful for finding the line that called the current screen. Set the screen environment to the previous level and enter LIST. The line that called the current screen is highlighted.

SCROLL

Displays source code.

Syntax

SCROLL UP|DOWN [n]

UP|DOWN

The direction to scroll.

n

The number of statement lines that you want to display.

Default: 20

Discussion

The SCROLL UP command with no number displays the previous page of source code. To display the previous page, enter

```
> SCROLL UP
```

The SCROLL UP command with a number displays n lines instead of 20 lines. To display the previous 10 lines, enter

```
> SCROLL UP 10
```

The SCROLL DOWN command with no number displays the next page of source code. The next page includes the last line of the previous listing. To display the next page, enter

```
> SCROLL DOWN
```

The SCROLL DOWN command with a number displays n lines instead of 20 lines. To display the next 15 lines, enter

```
> SCROLL DOWN 15
```

Note: The NEXT command is interchangeable with SCROLL DOWN. The PREVIOUS command is interchangeable with SCROLL UP.

SHOW

Shows information about items, record-structures, predefined items or screens.

Syntax

SHOW option

Options

The SHOW options are ITEM, FILE, PREDEFINED, SCREEN, and DB.

ITEM [item [OF [FILE] record-structure] [(occurrence)]]

With no options, SHOW ITEM shows the storage characteristics of all items in all record-structures named in the current screen.

[item [OF [FILE] record-structure]]

Specifying an item with no record-structure designation causes Debugger to display storage characteristics for all items of that name in all record-structures in the current screen. If a record-structure name is specified, Debugger displays only the item from the named record-structure.

(occurrence)

Displays information for the specified occurrence of the named record-structure.

FILE [record-structure|ALL [OPENED] [LOCKED] [(occurrence)]]

With no options, SHOW FILE displays the names of the files named in the current screen.

record-structure|ALL

Specifying a record-structure limits the information displayed to that record-structure; ALL displays information about all record-structures named in the current screen.

OPENED

MPE/iX, UNIX, Windows:	Displays information about files that are open.
OpenVMS:	Displays information about files and mailboxes that are open.

LOCKED

Displays information about files that are locked.

(occurrence)

Displays information for the specified occurrence of the named record-structure.

PREDEFINED

Lists all available predefined items.

SCREEN INFO|IMAGE

INFO

Displays information about the current screen. This is the same information that is displayed by entering "I" in the Action field.

IMAGE

Displays the current screen as it would appear on the screen with the values of displayed items.

DB

DB Options	
ALL	ATTACH
LOCALLY ACTIVE TRANSACTION	LOGICAL TRANSACTION
PHYSICAL TRANSACTION	REQUEST
SCREEN TRANSACTION	

ALL [SOURCE]

Displays the same information for relational databases as the following SHOW commands:

```
SHOW DB ATTACH

SHOW DB LOGICAL TRANSACTION

SHOW DB PHYSICAL TRANSACTION

SHOW DB REQUEST
```

SOURCE

Displays the BLR/SQL source structure containing the symbolic BLR values.

ATTACH [db_handle]

Displays information on each attach that is currently active.

db_handle

Displays information for the specified db_handle.

LOCALLY ACTIVE TRANSACTION

Displays information for each logical transaction that has been defined and is locally active.

LOGICAL TRANSACTION [transaction_name]

Displays information for each logical transaction that has been defined.

transaction_name

Displays information only for the specified logical transaction.

PHYSICAL TRANSACTION [transaction_handle]

Displays information about each physical transaction associated with each defined logical transaction.

transaction_handle

Displays information about the specified physical transaction.

REQUEST [request_handle] [SOURCE]

Displays information for each request that has been compiled.

request_handle

Displays information only for the specified request_handle.

SOURCE

Displays the BLR/SQL source structure containing the symbolic BLR values.

SCREEN TRANSACTION

Displays information for each logical transaction that has been defined by the current screen context.

Discussion

The SHOW command displays information about items, record-structures, predefined items, and screens.

In the following example, the SHOW command shows the storage characteristics of the record item EMPLOYEE in the file EMPLOYEE.

```
> SHOW ITEM EMPLOYEE OF FILE EMPLOYEE
EMPLOYEE
  EMPLOYEE Num
>
```

If the item name specified is not unique, all items by that name are listed along with their record name, or marked as a temporary or defined item.

The SHOW FILE command shows the available files with their open name and occurrence.

```
> SHOW FILE
EMPLOYEE(1)
BRANCHES(1)
```

Specifying a file or the ALL option shows more information about the named files, including

- the QUICK file type (for example, PRIMARY, SECONDARY)
- the physical file type
- the open mode
- the values of the predefined items AUDITSTATUS, NEWRECORD, ALTEREDRECORD, and DELETEDRECORD

In the following example, the SHOW command shows the characteristics of the record EMPLOYEE, the status of the record, the type of record declaration, and any open information associated with the record.

```
> SHOW FILE EMPLOYEE
EMPLOYEE
Primary (Indexed) Occurs 1
Open: EMPLOYEES Update Share #0/-1 (Closed)
Audit Status: N  NEWRECORD is TRUE   ALTEREDRECORD is FALSE  DELETEDRECORD is
FALSE
>
```

The line EMPLOYEE identifies the file. If the occurrence option is used, the occurrence is shown. For example, EMPLOYEE(4). The first number in "#0/-1" is the file number given in the OPEN option of the FILE statement. If an open number was not specified, Debugger displays 0. The second number is the physical file number. It is -1 when the file is closed.

In the following example, the SHOW command shows the information for all files.

```
> SHOW FILE ALL
```

The OPENED option causes Debugger to display the information only if the file is open. The LOCKED option causes Debugger to display the information only if the file is locked.

The SHOW PREDEFINED command shows the names of all predefined values, as in:

```
> SHOW PREDEFINED
ACCESSOK
FIELDTEXT
FIELDVALUE
PROMPTOK
PATH
OCCURRENCE
AUDITSTATUS
SYSNAME
SYSTIME
SYSDATE
SCREENLEVEL
PROCESSLOCATION
COMMANDOK
COMMANDCODE
COMMANDMESSAGE
STATUSTEXT
```

```
ENTRYMODE
CORRECTMODE
FINDMODE
SELECTMODE
CHANGEMODE
NEWRECORD
ALTEREDRECORD
DELETEDRECORD
OSACCESS
>
```

The SHOW SCREEN INFO command shows the screen information for the current screen as it would be shown by entering "I" in the Action field of the screen. This information includes the screen name and its level in the environment list. In the following example, the SHOW SCREEN INFO command is useful for showing which screen is the current screen.

```
> SHOW SCREEN INFO
QUICK830C L1 MAINMENU.QKC(2000/11/30) QDESIGN830C VT100 (c) COGNOS
INCORPORATED.
```

The SHOW SCREEN IMAGE command shows the current screen as QUICK would display it. To return to Debugger, press [Return]. The SHOW SCREEN IMAGE command shows the values of displayed items as they appear on the screen.

```
> SHOW SCREEN IMAGE
```

The SHOW DB command provides for the periodic examination of all or some subset of the database operations currently active. This may be used by the application designer in combination with breakpoints placed in the procedure code. When a break is signaled, QUICK gives control to Debugger. You can then display information stored in the QUICK relational data structures.

Information about Database Attaches

The command

```
> SHOW DB ATTACH [db_handle]
```

results in the following output

```
ATTACH <db_handle> TO <db_type> <db_name>
```

for each attach that is currently active, or for the specified db_handle.

Note: Descriptions of terms used in the preceding example are shown on (p. 541).

Information about Logical Transactions

The command

```
> SHOW DB LOGICAL TRANSACTION [transaction_name]
```

produces the following output

```
LOGICAL TRANSACTION <transaction_name> <details>
```

for each logical transaction that has been defined, or for the specified transaction_name. For each relational physical transaction associated with the logical transaction, the following information:

```
PHYSICAL TRANSACTION <transaction_handle> IN <db_handle_list>
```

will be produced.

Information about Locally Active Transactions

The command

```
> SHOW DB LOCALLY ACTIVE TRANSACTION
```

produces output similar to that of SHOW DB LOGICAL TRANSACTION, but only for those logical transactions that are locally active.

Information about Transactions in the Current Screen Context

The command

```
> SHOW DB SCREEN TRANSACTION
```

produces output similar to that of SHOW DB LOGICAL TRANSACTION, but only for those logical transactions that have been defined by the current screen context.

Information about Requests

The command

```
> SHOW DB REQUEST [request_handle] [SOURCE]
```

produces the following output:

```
REQUEST <request_handle> IN TRANSACTION <transaction_handle>
```

for each request that has been compiled, or for the specified request_handle. If SOURCE is specified, the BLR/SQL source structure containing the symbolic BLR values is displayed.

General Database Information

The command

```
> SHOW DB ALL [SOURCE]
```

has the effect of performing the following sequence of commands for relational databases:

For each db_handle:

```
> SHOW DB ATTACH db_handle
```

For each logical transaction associated with db_handle:

```
> SHOW DB LOGICAL TRANSACTION
```

For each physical transaction associated with the logical transaction:

```
> SHOW DB PHYSICAL TRANSACTION
```

For each request_handle associated with transaction_handle:

```
> SHOW DB REQUEST request_handle [SOURCE]
```

Terms Used in Command Output

Term	Description
db_handle	A numeric ID used by PowerHouse.
db_handle_list	A comma-separated list of <db_handle>.
db_name	The database name.
db_type	One of "ALLBASE", "DB2", "ODBC", "ORACLE", "Sybase" or "Oracle Rdb".
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>
request_handle	A numeric ID used by PowerHouse.
reserve_name_ comma_list	A list of the relation names specified on the reserving list for this transaction.
transaction_handle	A numeric code used to identify physical transactions.
transaction_name	A character name used to identify logical transactions.

STEP

Executes one or more statements of a screen.

Syntax

STEP [n]

n

Specifies the number of statements to execute.

Discussion

The STEP command with no number causes QUICK to execute the next line, then return to Debugger.

```
> STEP
```

The STEP command with a number causes QUICK to execute the next n lines, then return to Debugger. In the following example, the next five lines will be executed.

```
> STEP 5
```

If QUICK reaches a breakpoint, if a watched variable changes, or if a user break is entered before QUICK executes all n statements, Debugger still gets control. Debugger does not remember the previous STEP command; if CONTINUE is then entered, QUICK runs until it hits another breakpoint, a watched variable changes, or a user break is entered.

TYPE

Displays either a range of lines of source code or all lines on the current screen.

Syntax

TYPE [[n [[TO] m]]|ALL]

[n [[TO] m]]

Range of lines to list. TO is for documentation only.

ALL

Lists all source statements of the current screen.

Discussion

If no range is specified, the TYPE command displays the source code around the statement that QUICK was executing when Debugger gained control. The display is centered on the terminal window.

In the following example, the TYPE command is used to display the statements before and after the statement that QUICK was executing when Debugger gained control.

```
> TYPE
0001  ; Positions file screen
0002  SCREEN POSITION
.
.
.
0019      REQUEST POSITION OF POSITIONS
0020      IF PROMPTOK
>
```

If you specify a line range (n TO m), the TYPE command lists source code statements only within that range.

If m is missing, Debugger prints 20 lines centered around n.

The TYPE command ignores invalid line numbers.

In the following example:

- TYPE 15 lists 20 lines around line 15.
- TYPE 100 TO 120 displays lines 100 to 120.

```
> TYPE 15
0005    LOOKUP NOTON EMPLOYEE VIA EMPLOYEE
0006 FIELD LASTNAME OF EMPLOYEE REQUIRED NOCHANGE
0007 FIELD FIRSTNAME OF EMPLOYEE
0008 FIELD STREET OF EMPLOYEE
0009 FIELD CITY OF EMPLOYEE
0010 FIELD PROV-STATE OF EMPLOYEE
0011 FIELD POSTALCODE OF EMPLOYEE
0012 FIELD PHONE OF EMPLOYEE
0013 FIELD BRANCH OF EMPLOYEE
0014 FIELD DIVISION OF EMPLOYEE
0015 FIELD POSITION OF EMPLOYEE
0016 FIELD SEX OF EMPLOYEE
0017 FIELD LANGUAGE OF EMPLOYEE
0018 FIELD BILINGUAL OF EMPLOYEE
0019 FIELD DATEJOINED OF EMPLOYEE
0020 FIELD BILLING OF BILLINGS DISPLAY
0021 FIELD DATE-APPOINTED OF EMPLOYEE
0022
0023
0024
> TYPE 100 TO 120
```

```
0100      THEN GET...
0101    IF PATH = 5
.
.
.
0119    PUT EMPLOYEE
0120    END
>
```

The TYPE ALL command lists all the source code statements in the current screen.

```
> TYPE ALL
```

TYPE and LIST are interchangeable.

USE

Reads and executes Debugger commands from a file.

Syntax

USE filespec [DETAIL|NODETAIL] [LIST|NOLIST]

filespec

The specification of the file where Debugger commands are saved.

UNIX, Windows, OpenVMS: The default extension is ".qkl".

DETAIL|NODETAIL

The DETAIL option saves the contents of the file in the command history. The NODETAIL option saves only "USE filename NODETAIL" in the command history.

Default: DETAIL

LIST|NOLIST

The LIST option causes Debugger to display statements as they are read. The NOLIST option causes Debugger to read the statements in the file without showing them.

Default: LIST

Discussion

Debugger reads and interprets each statement as if it were entered from the terminal. The file can contain other USE commands, to a maximum of 20 levels. If the USE file contains a CONTINUE or STEP command, Debugger starts QUICK and stops reading input from the file.

In the following example, previously saved commands from a file called COMMANDS are used. The file contains a SHOW SCREEN INFO command and a BREAK command.

```
> USE COMMANDS
> SHOW SCREEN INFO
QUICK830C L1 MAINMENU.QKC(2000/11/30) QDESIGN830C VT100 (c) COGNOS
INCORPORATED.
> BREAK
Set at line: 33
Set at line: 52
>
```

To save Debugger commands in a source file, use the SAVE command.

User Break

Breaks to Debugger from QUICK.

Syntax

MPE/iX:	Ctrl-Y
OpenVMS, UNIX, Windows:	Ctrl-C

A user break is not a command entered at Debugger prompt, but occurs whenever the user presses Ctrl-Y or Ctrl-C.

Discussion

[Ctrl-Y] and [Ctrl-C] break to Debugger from QUICK. Debugger receives a user break when the interrupt key for the system is used while QUICK is executing a screen.

A user break causes QUICK to give control to Debugger as soon as it can. Because of the way QUICK and Debugger interact, QUICK may not give control to Debugger immediately. Often a user break must be typed with something else (find a record, for example) to make QUICK pass control to Debugger.

WATCH

Sets or displays a watchpoint.

Syntax

WATCH

WATCH [ITEM] item
[OF [FILE] record-structure] [(occurrence)] [ON|OFF]

WATCH PREDEFINED predefined-value
[OF [FILE] record-structure] [(occurrence)] [ON|OFF]

WATCH FILE record-structure [(occurrence)] [ON|OFF]

WATCH ALL [ON|OFF]

Options

[ITEM] item [OF [FILE] record-structure]

Specifies an item name for which to set a watchpoint, and optionally the record-structure that the item is from.

(occurrence)

Specifies which occurrence of an item or record-structure to watch.

FILE record-structure

Sets watchpoints on all items associated with the named record-structure.

PREDEFINED [predefined-value [OF [FILE] record-structure]]

Sets a watchpoint on a specified predefined value and, optionally the record-structure the predefined value is for. Predefined values include predefined items, system functions, and predefined conditions.

ALL

Sets a watchpoint on all items in the current screen except predefined items.

ON|OFF

The ON option sets a watchpoint; the OFF option removes a watchpoint.

Discussion

The WATCH item command sets a watchpoint on the named record-structure, record item or temporary item. Whenever the value of the watched item changes, Debugger gets control. The current line is the line after the line that changed the variable (except in certain circumstances discussed below).

The following example shows how to set a watch on the item LASTNAME:

```
> WATCH LASTNAME OF EMPLOYEE
Watchpoint is set for LASTNAME OF EMPLOYEE.
>
```

The WATCH item OFF command removes the watch on a record-structure or temporary item. The following example shows how to turn off the watch on the item LASTNAME:

```
> WATCH LASTNAME OFF
Watchpoint cleared for LASTNAME OF EMPLOYEE.
>
```

The WATCH PREDEFINED predefined-value command sets a watch on a predefined value. For the predefined values AUDITSTATUS, ALTEREDRECORD, DELETEDRECORD, and NEWRECORD, a record-structure must be specified. The following example shows how to set a watch on the predefined value NEWRECORD:

```
> WATCH PREDEFINED NEWRECORD OF EMPLOYEE ON
Watchpoint is set for NEWRECORD OF EMPLOYEE.
>
```

The WATCH ALL command watches all items. The WATCH ALL OFF command removes and lists all watchpoints cleared from the current screen.

The WATCH command by itself lists all watched items in the current screen. The following example shows how to list all the items being watched:

```
> WATCH
> Watchpoint is set for EMPLOYEE OF EMPLOYEE.
> Watchpoint is set for LASTNAME OF EMPLOYEE.
> Watchpoint is set for CITY OF EMPLOYEE.>
```

When Debugger gets control because of a watchpoint, the display is

```
Watchpoint trap encountered at line: 52, for EMPLOYEE OF EMPLOYEE
Old Value =      0
New Value =      1
>
```

Note: Debugger can only gain control when QUICK is executing procedure code, which is either user generated or from QUICK. If a watched item changes when QUICK is not executing procedure code, Debugger does not notice the change until the next procedure code statement.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z