IBM Cognos Application Development Tools
IBM Cognos PowerHouse 4GL

Version 8.4G

Primer

Information Management software

Product Information

This document applies to IBM Cognos PowerHouse 4GL 8.4G and may also apply to subsequent releases. To check for newer versions of this document, visit the IBM Cognos Information Centers (http://publib.boulder.ibm.com/infocenter/cogic/v1r0m0/index.jsp).

Copyright

Table of Contents

Chapter 1: IBM Cognos PowerHouse 4GL Means Productivity

IBM® Cognos® PowerHouse® 4GL is a server-based, fourth-generation application development environment. PowerHouse 4GL gives you the power to design, create, and manage complex yet easy-to-use business applications that are integrated with your existing environment, data, and applications. It makes you a more productive developer and gives you reliable, flexible, distributed, and scalable deployment options.

Industrial Strength Power

You can develop an application with PowerHouse 4GL considerably faster than with a third-generation language such as Cor COBOL. PowerHouse 4GL simplifies operations that might take dozens of lines of 3GL code into a single instruction.

With PowerHouse 4GL, the power is ready when you need it. The smooth, incremental learning curve of PowerHouse 4GL means you get an immediate return on your investment. When you first start to use PowerHouse 4GL, many of your basic requirements are handled by its default features. As you become more familiar with PowerHouse 4GL, you can add your own specifications to the default application. Finally, you can take full control and make use of PowerHouse 4GL's powerful procedural constructs.

Commercial applications contain common elements that 3GL programmers find themselves coding repeatedly. Basic validation functions such as data input, entire screen-handling routines, file I/O, and concurrency controls are all problems that 3GL programmers encounter. PowerHouse 4GL, however, provides such functions automatically and intelligently, while still allowing programmers to customize standard operations, without having to resort to 3GL coding.

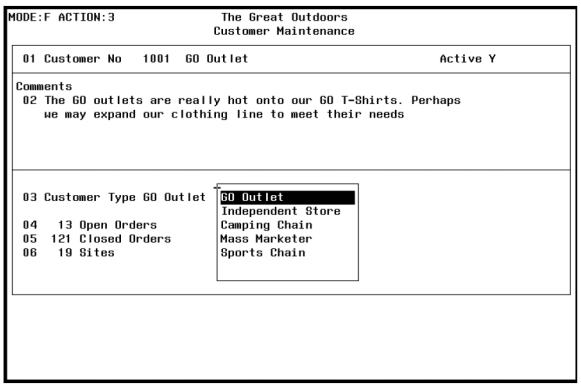
With PowerHouse 4GL, you have the power to develop systems with features like these:

- sophisticated menus that tie processes together into user-friendly applications
- production reporting applications that combine data from multiple sources
- complex, multi-step volume processing tasks such as rollups, resets, and bulk file edits

With its English-like commands, PowerHouse 4GL can easily be used by data processing professionals and novice users alike.

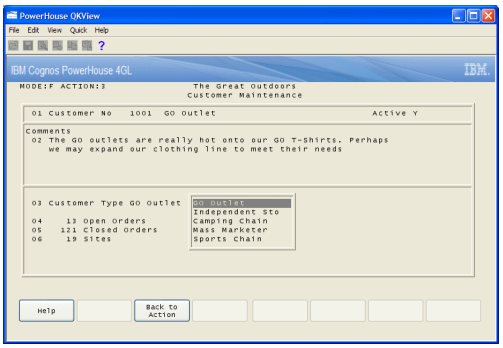
Multiple End-User Interfaces

PowerHouse 4GL helps you develop applications that are easy to learn and intuitive to operate. QUICK, the interactive end-user component of PowerHouse 4GL, allows you to deploy data entry and inquiry applications to terminals. This is the traditional terminal interface available on OpenVMS and the UNIX® and Linux® operating systems.



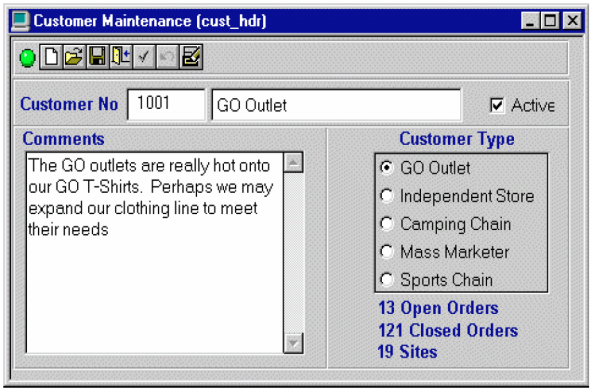
On the Microsoft® Windows® operating system, you can use the traditional terminal interface in a Command Prompt window.

On UNIX, Linux, and Windows you can use the QKView interface.

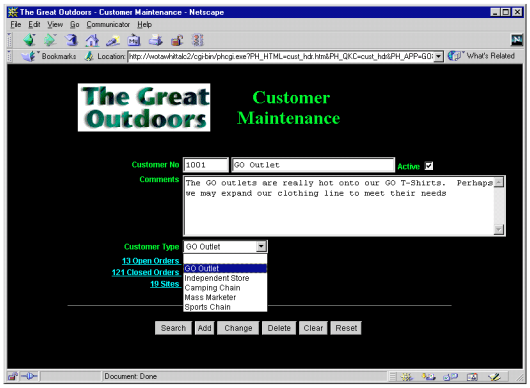


The QKView interface is a Java™ application. It can be used as a shell client, a network applicaiton, or as an applet in a Web browser.

But there's more. IBM Cognos Axiant® 4GL is a visual Windows-based development environment for creating PowerHouse 4GL 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 4GL a Windows-like user interface.



IBM Cognos PowerHouse Web enables Web deployment of applications built with PowerHouse 4GL. You can use a standard Web browser to search, add, change, and delete data using screens and pages developed using the same high-productivity tool - PowerHouse 4GL.



For more information about Axiant 4GL and PowerHouse Web, please visit our Web site at http://www.ibm.com/software/data/cognos/products/powerhouse.

Power to Integrate

PowerHouse 4GL uses your existing data files and database management systems without requiring costly data conversion. You can enhance and integrate existing applications with new applications you build in PowerHouse 4GL.

Building Applications with PowerHouse 4GL

Building an application with PowerHouse 4GL is easy. Once you determine the information you need, simply

- Build your data dictionary
- Define your data and build your files
- Generate, test, and refine your screens

Once you have a working system, you can

- create reports
- add power with high-volume processing
- integrate the system with menus and screens
- fine-tune your applications
- build in security

With the help of this primer, you can start developing your total solution right now.

About this Primer

This primer has two objectives: it gives you an overview of the PowerHouse 4GL language and it gives you a hands-on demonstration of just how easy it is to use PowerHouse 4GL.

Using this primer, even a novice computer user can create a simple but complete working application in a short time. For some users, this primer may be all that is needed to make productive use of PowerHouse 4GL for some time to come.

This primer provides only a glimpse of PowerHouse 4GL's potential. However, it will give you a feel for what PowerHouse 4GL can do.

What's Ahead?

The following chapters introduce you to PowerHouse 4GL. In about half a day, you'll create a practical PowerHouse 4GL business application: a working purchase order tracking system.

The sample application uses an indexed file system to make it as universal as possible. However, it could also be built with a relational database system since PowerHouse 4GL effectively integrates with the most popular relational databases. For the benefit of relational users, notes are provided throughout the primer to indicate where the use of a relational database would make a difference.

In Chapter 2, you'll learn how to create a practical framework for your application by defining your data using PDL (PowerHouse Definition Language) to create a data dictionary. PDL is the foundation on which PowerHouse 4GL applications are built. Once you have built your dictionary, the other PowerHouse 4GL components automatically use the definitions you have created.

In Chapter 3, you'll learn how to design and build screens using QDESIGN, the QUICK screen builder. You'll discover the simplicity of using PowerHouse 4GL QUICK screens for online interactive transaction processing. You'll also see how easy it is to create and use a Web page with PowerHouse Web.

In Chapter 4, you'll learn to use QUIZ, the PowerHouse 4GL report writer. You'll find out how quickly you can put your data to work. With QUIZ, you can turn data into organized information, producing simple or complex reports in just minutes.

In Chapter 5, you'll learn to use QTP, the PowerHouse 4GL transaction processor. QTP gives you the ability to update large batches of data automatically.

In Chapter 6, you'll put the finishing touches on your application and add a user-friendly menu.

In Chapter 7, you'll find a springboard to new ideas for application development.

That's PowerHouse 4GL, a remarkable computer language that combines simplicity and sophistication in a way that will both satisfy the most demanding professional requirements, and allow novice users to benefit from their computers as never before. And that means increased productivity in any language.

Conventions Used in this Document

When this document tells uses the term "enter", type the entry and then press [Return] or [Enter]. An entry is not recognized until you press the [Return] or [Enter] key.

In this document, PowerHouse 4GL code is shown in uppercase type (for example, SCREEN). When you enter code, however, you may use uppercase, lowercase, or mixed case type with the exception of case-sensitive file names on UNIX and Linux.

For all operating systems, we assume that PowerHouse 4GL commands have been set up during product install. If commands such as "pdl" or "quiz", do not work correctly, check with your system manager.

This document describes PowerHouse 4GL on four platforms – OpenVMS, UNIX, Linux, and Windows. The only differences in the PowerHouse 4GL code used are physical file names that are operating system dependent. This gives you a vivid example of how portable PowerHouse 4GL can be.

On OpenVMS, UNIX, Linux, and Windows, PowerHouse 4GL uses file extensions to identify the type of file. For example, a compiled screen file would have an extension of QKC, as in PURCHASE.QKC, while the source file would have an extension of QKS, as in PURCHASE.QKS. PowerHouse 4GL automatically adds the extensions. This means that you can create a screen and its source using the same root name.

The UNIX and Linux operating systems are case-sensitive. For example, PDL is not the same as pdl. Make sure you use the appropriate case when you enter UNIX or Linux commands and file names that are passed to the operating system; for most UNIX and Linux commands, this is lowercase. Once you're inside PowerHouse 4GL, you can use either case for PowerHouse 4GL keywords and names, but physical file names will retain the case you use. Relational database table or column names may be case- sensitive.

For convenience, this document shows operating system and shell commands in lowercase, and PowerHouse 4GL code in uppercase.

IBM Cognos PowerHouse 4GL Documentation Set

PowerHouse 4GL documentation, available on the IBM Cognos PowerHouse 4GL Books CD, includes planning and configuration advice, detailed information about statements and procedures, installation instructions, and last minute product information.

Objective	Document
Install PowerHouse 4GL	The *IBM Cognos PowerHouse 4GL Getting Started* document provides step-by-step instructions on installing PowerHouse 4GL.
Review changes and new features	The *IBM Cognos PowerHouse 4GL Release Notes* document provides information on supported environments, changes, and new features for the current version.
Get an introduction to PowerHouse 4GL	The *IBM Cognos PowerHouse 4GL Primer* document provides an overview of the PowerHouse language and a hands-on demonstration of how to use PowerHouse.
Get detailed reference information for PowerHouse 4GL	The IBM Cognos PowerHouse 4GL Reference documents provide detailed information about the PowerHouse language and each PowerHouse component. The documents are • *IBM Cognos PowerHouse 4GL PowerHouse Rules* • *IBM Cognos PowerHouse 4GL PDL and Utilities Reference* • *IBM Cognos PowerHouse 4GL PHD Reference (OpenVMS)* • *IBM Cognos PowerHouse 4GL PowerHouse and Relational Databases* • *IBM Cognos PowerHouse 4GL QDESIGN Reference* • *IBM Cognos PowerHouse 4GL QUIZ Reference* • *IBM Cognos PowerHouse 4GL QTP Reference*

IBM Cognos PowerHouse Web Documentation Set

PowerHouse Web documentation, available from the IBM Cognos PowerHouse Web Administrator CD, includes planning and configuration advice, detailed information about statements and procedures, installation instructions, and last minute product information.

Objective	Document
Start using PowerHouse Web	The *IBM Cognos PowerHouse Web Planning and Configuration* 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.
Install PowerHouse Web	The *IBM Cognos PowerHouse Web Getting Started* document provides step-by-step instructions on installing PowerHouse Web.
Review changes and new features	The *IBM Cognos PowerHouse Web Release Notes* document provides information on supported environments, changes, and new features for the current version.
Get detailed information for developing PowerHouse Web applications	The *IBM Cognos PowerHouse Web Developer's Guide* document provides detailed reference material for application developers.

Objective	Document
Administer PowerHouse Web	The *PowerHouse Web Administrator Online Help*, available from within the PowerHouse Web Administrator, provides detailed reference material to help you during PowerHouse Web configuration.

IBM Cognos Axiant 4GL Documentation Set

Axiant 4GL documentation, available from the IBM Cognos Axiant 4GL CD, includes planning and configuration advice, detailed information about statements and procedures, installation instructions, and last minute product information.

Objective	Document
Install Axiant 4GL	The *IBM Cognos Axiant 4GL Web Getting Started* document provides step-by-step instructions on installing Axiant 4GL.
Review changes and new features	The *IBM Cognos Axiant 4GL Release Notes* document provides information on supported environments, changes, and new features for the current version.
Get an introduction to Axiant 4GL	The *A Guided Tour of Axiant 4GL* document contains hands-on tutorials that introduce the Axiant 4GL migration process and screen customization.
Get detailed reference information on Axiant 4GL	The *Axiant 4GL Online Help*, available from within Axiant 4GL, provides a detailed reference guide to Axiant 4GL.

Getting Help

For more information about using this product or for technical assistance, go to http://www.ibm.com/support. Under **Choose support type**, select **Information Management,** then under **Choose a product**, select **Cognos Application Development Tools**. Under **Cognos Application Development Tools support**, click **Documentation**.

Chapter 2: A Different Kind of Dictionary

In the world of electronic information handling, a data dictionary is quite different from the dictionary you find on a bookshelf. A data dictionary is like a rule book in which you write the rules.

In your data dictionary, you describe your records by assigning names to the categories of information you will store in your files. You specify how much space each category needs, and you specify formats for things like dates, cash amounts, and so on. What you are doing is building a framework to organize and control your data.

In this chapter, you'll use PDL (the PowerHouse Definition Language) to create a data dictionary that is the base on which you'll build your purchase order tracking application.

Unlike a conventional dictionary, your PDL data dictionary isn't just a passive storehouse of words and data definitions. PDL also helps QUICK, QUIZ, and QTP to interpret your instructions and present information the way you want it— automatically. And once you've created your data dictionary, you can use it again and again as the basis for other IBM Cognos PowerHouse 4GL applications.

Relational?

In this chapter we describe how to build a PowerHouse dictionary for use with an indexed file system. Much of the information also applies to relational systems but there are significant differences.

In an indexed system, you create elements that must then be associated with either a record-structure or an index, and stored in files, all of which are explained in this chapter.

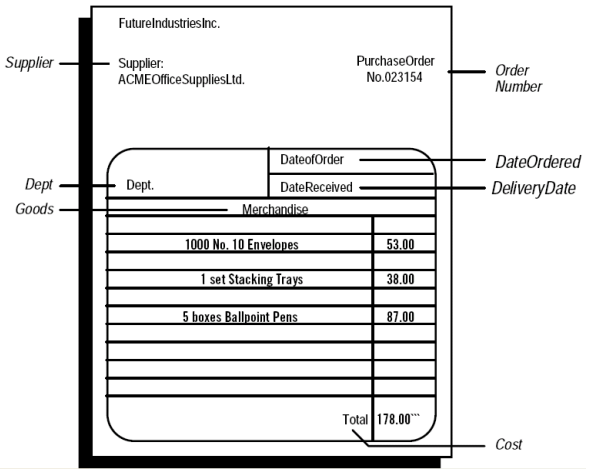
A relational database is structured quite differently, consisting of tables that contain rows and columns of data. So when you create a dictionary to access a relational database, you are describing an existing set of data. We'll point out specific differences as we proceed.

Planning Your System

To introduce you to PowerHouse 4GL, we have invented a company called Future Industries. The entire purchasing operation at Future Industries uses purchase orders based on standard business forms. For each purchase made, one copy of the form goes to the supplier, one goes into a filing cabinet, one goes to Accounts Payable, and yet another goes to the purchasing department. When Future Industries receives goods, someone retrieves the filed copies and marks the order as filled.

To maintain paper records and extract statistics from them is a time- consuming and tedious task—and one that is particularly vulnerable to human error. With PowerHouse 4GL, you can develop an application enabling Future Industries to establish an efficient purchase-order tracking system that reduces the paperwork and speeds up operations.

Consider the Future Industries paper order form, which contains the elements of a new tracking system:



The first step is to take a good look at the information contained in the company's paper-based system. You can break it down into its basic elements and use them to design the structure of your records. With the help of PDL, you can use these simple record-structures to create a computerized purchase-order tracking system.

The backbone of the old system is the purchase order form. Each purchase order is a record of a transaction. It will be the basic record in our system, too. Begin by making a list of the elements that the purchase order form contains.

The first element is the order number. Every record needs a different number, so the ORDERNUMBER element is an essential one in your system. The computer can find any data record by its unique order number.

The next element is the name of the department placing the order. We'll abbreviate it to DEPT. This, too, is key information, as you want to be able to trace the purchases made by a particular department.

Next, you need to add an element identifying suppliers; let's call it SUPPLIER. Following this are two date elements: the date the order is placed and the date the goods are delivered. We'll name these DATEORDERED and DATEDELIVERED. The original purchase order form also has spaces for the actual goods ordered and the cost — add GOODS and COST to complete your list of elements. You have now covered all of the basic elements on the form.

Your list now looks like this:

```
ORDERNUMBER
DEPT
SUPPLIER
DATEORDERED
DATEDELIVERED
GOODS
COST
```

By creating a record-structure using these elements you will produce an electronic version of the order form. We'll add one more element to make the electronic version more efficient.

For data entry purposes, you could save a lot of time by abbreviating the names of your regular suppliers, giving each a brief code name. That gives you one more element — call it SUPPCODE. You can use it in this record-structure instead of SUPPLIER.

The purchasing department also needs to keep track of its suppliers. You need a second record-structure, separate from the one just outlined, containing just SUPPCODE and SUPPLIER.

Files for Your Records

Once you start adding records of actual transactions to your system, you will need files to contain them. You'll need two files: one for data records based on the ORDERS record-structure and one for data records based on the SUPPLIERS record-structure. Let's give each file the same name as its record-structure.

This is how the outline of your system looks now:

ORDERS file:	SUPPLIERS file:
contains one record structure, ORDERS	contains one record structure, SUPPLIERS
ORDERNUMBER	SUPPCODE
DEPT	SUPPLIER
SUPPCODE	
DATEORDERED	
DATEDELIVERED	
GOODS	
COST	

In an actual system, you would probably need more files, and some of your files might contain multiple record-structures, but this simplified model will provide a working application.

Online Editing

As you work through the next few pages you'll be entering the statements that create your data dictionary. If you make a typing mistake you can type CANCEL CLEAR and reenter your statements from the beginning.

If you're familiar with your system's text editor, you can use it to make corrections much more efficiently. You call up the editor by entering

```
> REVISE
```

The REVISE statement also works the same way in QDESIGN, QUIZ, and QTP.

When you finish your editing session, you're automatically returned to the PowerHouse 4GL component you were in when you began the session. A "save" file is created to contain all of your edited statements. To safeguard against losing your edits in a revised file, PowerHouse 4GL checks for unsaved changes in your save file when you try to exit PDL. If unsaved changes exist, PowerHouse 4GL prompts you to save the changes.

To edit an existing file, type that file's name after the REVISE statement, as in

```
> REVISE ORDERS
```

PowerHouse 4GL supplies the named file to the editor automatically. When you exit from the editor, you re-enter PDL and your edited statements from the revised file are processed automatically.

Getting Started

So far, you've just outlined what you want to do; now, let's do it. First, log on to your computer system. If you're using Windows, open a Command prompt window and navigate to a working directory.

When you see the operating system or Shell prompt, activate the data dictionary program by entering

```
pdl
```

When you see the prompt (>), PDL is ready to accept your dictionary statements.

The first step is to create your data dictionary. Since this is a dictionary for a purchase order tracking system, enter

```
> CREATE DICTIONARY PODICT
```

Now, enter a name for a title that prints on all your QUIZ and QSHOW reports. It should be something descriptive, such as "P.O. Tracking System". You'll also want to set the default century to 20. Enter

```
> SYSTEM OPTIONS TITLE "P.O. TRACKING SYSTEM" DEFAULT CENTURY 20
```

Defining Your Elements

Once you've named your data dictionary, you must describe each of the elements on your list in separate ELEMENT statements. Each element in your dictionary needs a unique name, a type (numeric, character, or date), and a size.

Relational?

Defining elements for PowerHouse 4GL is exactly the same whether you're using an indexed file system or a relational database. The difference is in how the element definitions are used.

When you add an element to a dictionary working with a non- relational file system, you're actually designing a new unit of data. But with a relational system what you're doing is customizing the attributes of existing columns in a database table. The database definition for the column is unchanged. In a relational system, the ELEMENT statement determines how the data will be used in PowerHouse 4GL applications.

You have already identified all the basic elements on the Future Industries' purchase order form. Use these names as the names for the elements in your data dictionary. The order in which you list the elements isn't important. We'll start with the supplier name. Enter the following statement:

```
> ELEMENT SUPPLIER CHARACTER SIZE 25
```

This tells PDL that SUPPLIER is a character element of size 25. That means the name can consist of up to 25 letters, digits, or other symbols.

Now we'll define the COST element. If you had to describe the type, size, and other attributes of the COST element in full, as you did with SUPPLIER, it could be a complex element to define. But you can use a usage to set these characteristics automatically. PowerHouse 4GL includes a set of usages in the dictionary when you enter the CREATE DICTIONARY statement.

Some of these usages are:

ID identifies character elements, such as the supplier name, that contain indexes in your system

- NUMERIC-ID identifies character items, such as ORDERNUMBER, that are key items in your system
- MONEY provides a 9-digit format, including two decimal places, for currency elements
- NAME provides a 20-character format suitable for a name
- PHONE provides a format for 7 or 10-digit phone numbers

Cost is always monetary, therefore use the usage MONEY as in

```
> ELEMENT COST USAGE MONEY
```

The next two elements are dates. PDL automatically assigns the format for you when the type is DATE. You don't have to include a usage. However, you must tell PDL what size you want the dates to be. Dates in PowerHouse 4GL can be either six or eight digits.

The six-digit date format puts the year (yy) first, the month (mm) second, and the day (dd) last. The eight- digit format adds the century (in this case, 20) onto the year portion, so that the date is displayed as yyyymmdd.

In order for your application to handle multiple centuries without any extra work on your part, you should always use eight-digit dates that incorporate the century.

Enter these two statements:

```
> ELEMENT DATEORDERED DATE SIZE 8
> ELEMENT DATEDELIVERED DATE SIZE 8
```

The next element is DEPT, a character element. The usage ID is appropriate for this element. The type is automatically set to character, but you must specify the size. Abbreviating the department names to four letters makes entering the names easier, so specify a size of 4, by entering

```
> ELEMENT DEPT USAGE ID SIZE 4
```

Since PowerHouse 4GL accepts both uppercase and lowercase entries, an entry of "acct" and an entry of "ACCT" are different. This could make entering department names confusing, but the usage ID takes care of that with the UPSHIFT option, which stores values in uppercase. So, for example, "acct" will always appear as "ACCT". And that makes looking up and entering values much easier.

The GOODS element is also a character element. Use the usage NAME to define it by entering

```
> ELEMENT GOODS USAGE NAME
```

The next element to define is ORDERNUMBER. The NUMERIC-ID usage works well here. You'll need to specify the element size—six digits allows for as many orders as you'll ever need, so enter

```
> ELEMENT ORDERNUMBER USAGE NUMERIC-ID SIZE 6
```

For the abbreviated supplier names, use four characters and the usage ID, just as you did for the department names. Enter

```
> ELEMENT SUPPCODE USAGE ID SIZE 4
```

Because the application you're building is a prototype, you've given each element only the minimum definition needed to get the application running. There are many more attributes you could add to improve both the appearance and the operation of your system.

For example, element definitions can include patterns and values (or ranges of values) that other PowerHouse 4GL components check to ensure that all entries match your specifications. You can customize your screens and reports by including your own labels and headings to override PDL's defaults. You can control how an element is displayed by specifying details such as currency symbols and leading or trailing signs. You can also use different "fill" characters such as asterisks (*) to fill the blanks in elements such as monetary amounts.

For more information on elements and their attributes, see the section on the ELEMENT statement in the *PDL and Utilities Reference*.

Defining Your Files

Now that you've described all the elements in your purchase order tracking system, you need to describe the files. A file is a collection of records that use the same record-structure. You need one FILE statement for each of the two files in your application: ORDERS and SUPPLIERS. Each file must be given a name and an organization.

Relational?

If you're working with a relational database, you don't have to be concerned with the steps described on the next few pages as they relate specifically to an indexed file system. The FILE, INDEX, ITEM, RECORD and SEGMENT statements are not used with relational databases.

To create a PowerHouse dictionary for a relational system, you only need one statement: DATABASE. You use the DATABASE statement to attach a database to the dictionary. The DATABASE statement names the type of relational database you're using.

```
> DATABASE PODB TYPE ORACLE
```

You would also provide the open name and password. For more information, see the *PDL and Utilities Reference*.

You can also use the ELEMENT statement to customize the formatting and editing of database columns.

The order in which you enter PDL statements is important. The FILE, RECORD, ITEM, INDEX, and SEGMENT statements for each record-structure must be kept together. You must define all the characteristics of one file first, then all the characteristics of the other.

You've already decided on the file names SUPPLIERS and ORDERS. As for file organizations, there are several available to you. For this application we'll be using indexed files.

Let's start with the ORDERS file. Enter

```
> FILE ORDERS ORGANIZATION INDEXED
```

Creating a Record-Structure

Next you're going to describe the record-structure for the ORDERS file before you move on to the SUPPLIERS file. To create a record- structure, simply state which elements you want to include in it, and in what order.

PDL uses the element descriptions you already entered to create your record-structures. The same element can be used in many different record-structures; each repetition of an element is called an item. The ITEM statement creates an item in its particular record- structure that is based on the element that you defined with an ELEMENT statement. For example, the item ORDERNUMBER describes an actual order number in an ORDERS record.

To create the items you need, enter the following statements:

```
> RECORD ORDERS
>   ITEM ORDERNUMBER
>   ITEM DEPT
>   ITEM SUPPCODE
>   ITEM GOODS
>   ITEM COST
>   ITEM DATEORDERED
>   ITEM DATEDELIVERED
```

You can begin a line in any column after the prompt, but indentation makes your ITEM statements easier to read.

You have now defined a record-structure for the ORDERS file. Before you define the SUPPLIERS file, you need to declare the index items for the ORDERS record-structure.

Working with Indexes

An index in a record-structure works much like the index in a book. If a book has no index, you have to leaf through each page looking for the information that you want. If a record-structure has no index, PowerHouse 4GL has to look at each record until it finds the one that you want. With an index, it can search on just one item (for example, ORDERNUMBER) until it finds the value (for example, ORDERNUMBER 1404660) that matches.

Indexes also allow you to link two different record-structures by a particular item, such as SUPPCODE. This index lets you look up record-structures for any particular supplier without having to enter the company's full name. Later you'll also see how this lets you cross- reference record-structures.

Defining the indexes in a record-structure is an important part of creating a data dictionary. Every record-structure that belongs to an indexed file must have at least one index.

You always use two statements to create an index: INDEX and SEGMENT.

- The INDEX statement actually declares the index. PowerHouse 4GL lets you identify two kinds of indexes. The REPEATING and UNIQUE options describe whether or not the index can contain duplicate values. For example, an index of surnames is repeating because there is always the possibility that several people have the same last name. A unique index, however, describes an index where it's essential that no duplicate values exist, for example, ORDERNUMBER.
- The SEGMENT statement names a record item to be used in the index. Complex applications often have more than one SEGMENT statement (creating what's called a multi-segment index), but for your application, a single SEGMENT statement for each INDEX statement is enough.

For ORDERS choose an item that you are likely to use to look up information—ORDERNUMBER, for example.

You'll also need to be able to check information against the department number. Because the same department can have many orders, we'll make DEPT a repeating index. Enter

```
>   INDEX ORDERNUMBER UNIQUE
>     SEGMENT ORDERNUMBER
>   INDEX DEPT REPEATING
>     SEGMENT DEPT
```

You need to repeat this process in order to create the SUPPLIERS record-structure.

Remember, the first statement must be the FILE statement. Enter

```
> FILE SUPPLIERS ORGANIZATION INDEXED
```

There are only two items in this record-structure—SUPPCODE and SUPPLIER. It will be much easier to look up records using the SUPPCODE, so create an index for SUPPCODE. Enter

```
> RECORD SUPPLIERS
>   ITEM SUPPCODE
>   ITEM SUPPLIER
>   INDEX SUPPCODE UNIQUE
>     SEGMENT SUPPCODE
```

You've now entered all the statements to define your data dictionary. All that remains is to load these statements into PDL. Enter

```
> LOAD
```

If you've entered the statements correctly, PDL displays the following message:

```
0 ERRORS 0 WARNINGS
```

If the message indicates that there are errors, you have two choices: either enter CANCEL CLEAR then re-enter your statements from the beginning, or use the REVISE statement to call up the system text editor, and edit online. (See "Online Editing" earlier in this chapter.)

You should save a copy of your correct data dictionary statements in a file, in case you want to make some additions or changes later. On OpenVMS, UNIX, Linux, and Windows, use the same name that you gave the data dictionary in the CREATE DICTIONARY statement.

```
> SAVE PODICT
```

Now, any time you want to use these statements in PDL, enter the USE statement followed by the dictionary name, as in

```
> USE PODICT
```

You've just created a simple data dictionary. As you develop other parts of your application, all the other PowerHouse 4GL components can refer to this data dictionary as a blueprint.

If you try to create another data dictionary file with the same name, PowerHouse 4GL warns you that you are about to delete the original file.

If you want to delete the original file, enter Y at the warning. If you don't, enter N. Then enter the CREATE DICTIONARY statement again, using a different dictionary name.

Creating Your Data Files

There's one more task to perform before you move on: you need to create the data files you've just defined. To do this, you will leave PDL and use QUTIL, the PowerHouse 4GL file creation utility.

To leave PDL, enter

```
> EXIT
```

When the operating system prompt appears, activate QUTIL and tell it what data dictionary files you wish to create by entering

```
qutil dict=podict
```

When the prompt (>) appears, enter

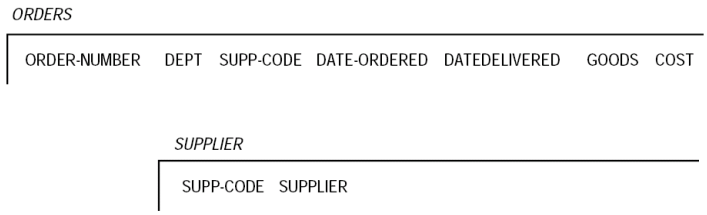
```
> CREATE ALL
```

QUTIL builds your two files. You'll see several messages on your screen while this is happening. Depending on your operating system, the file type may be different, but you will shortly see messages telling you that the files have been created.

To leave QUTIL, enter

```
> EXIT
```

Here's what you've just created:



Refining Your Dictionary

Now that you have built a simple dictionary and created the data files, you can begin using other PowerHouse 4GL components to enter, manipulate, or retrieve information. For simple applications, this may be all you need. However, PDL offers many more features to enhance your application, or to enable the computer professional to build advanced applications. Here are a few of these features:

- centralized data definition
- full commercial data formatting
- a wide range of datatypes
- complex input editing rules
- file and element security
- application-wide standards

All data management functions are controlled through the data dictionary. PDL also includes a comprehensive set of utilities that enable you to

- show the contents of your data dictionary, online or through reports
- investigate the impact of changes before they're implemented
- manage the creation and deletion of files

A Review of PowerHouse 4GL Entities

PDL provides the definitions for all entities the PowerHouse 4GL building blocks. The most basic building blocks are elements. Elements are combined into groups called record-structures. Files store records that have a particular record-structure. As you will see in the following chapters, QUICK and QTP can be used to add data to files.

There are essentially two types of entities in PowerHouse 4GL: logical and physical. Logical entities describe such things as how to present data. Physical entities describe how and where the data in your application is stored in the computer. In most cases, once you've described the logical entities, PDL can take care of the physical entities for you. You always have the option of specifying them yourself.

The main logical entities are:

Element

A category of data that represents many individual values. For example, the element ORDERNUMBER represents all order numbers and the element SUPPLIER represents all of Future Industries' suppliers.

Record-Structure

An ordered collection of elements that pertain to a particular object or activity. For example, the record-structure of a file used to store a mailing list of Future Industries' suppliers might consist of several elements, including Lastname, Firstname, Streetaddress, City, and Prov/State.

The main physical entities are:

File

A collection of records using the same record-structure. For example, a mailing list file typically contains one record for each of your customers. Each record in the file has the same set of items, but the item value in each is different.

Record

One complete set of the items in a record-structure, in which each item has a value.

Item

An element as it occurs in a particular record-structure. Each element can occur in any number of record-structures, and in many different applications, but in each one it forms a different item.

Index

One or more items whose values identify individual records or groups of records. The items appearing in an index are called segments.

What's Next

You are now ready to take the next step in building your application. In Chapter 3, you will learn how to use QDESIGN and QUICK to create your own screens and to add information to your files.

Chapter 3: QUICK and Easy

Each QUICK data entry screen is like a page in an electronic record book or a ledger that has been custom-designed to meet your information storage needs. Using a QUICK screen is like filling out a form, but faster and easier because of QUICK's many automatic features. Once you have designed your screen, you can use it over and over again to enter and file your data—what's more, it's a quick and easy job to create as many screens as you need.

Building a working QUICK screen with QDESIGN requires only four simple statements, provided you've specified a dictionary:

SCREEN	names your screen
FILE	indicates which file you want
GENERATE	generates the fields that your screen will include
BUILD	compiles your screen

To start QDESIGN and tell it what dictionary you want to use, at the operating system or Shell prompt enter

```
qdesign dictionary=podict
```

Relational?

If you're using a relational database with IBM Cognos PowerHouse 4GL, you have a choice in how you declare your data. One way is to use the CURSOR or FILE statement to specify the table that contains the data, as in

```
> CURSOR ORDERS IN PODB
> FILE ORDERS IN PODB
```

If you use the subdict=search program parameter when starting QDESIGN, QDESIGN will search the databases in your dictionary for the specified file. You do not need to specify the database name, as in

```
qdesign subdict=search dictionary=podict
> SCREEN PURCHASE
> CURSOR ORDERS
```

You can also use an SQL DECLARE CURSOR statement containing SQL syntax before the CURSOR statement. This creates a cursor containing the tables and columns you want to use. For more information about using cursors, see the *IBM Cognos PowerHouse 4GL QDESIGN Reference* and the *IBM Cognos PowerHouse 4GL PowerHouse and Relational Databases* document.

When you see the prompt (>), QDESIGN is ready to accept your screen design statements. If you make a mistake while entering your statements, QDESIGN will give you an error message. If you're not using a text editor and you make a mistake, enter CANCEL CLEAR and start again from the beginning.

You can use your text editor at any time without leaving QDESIGN simply by entering the REVISE statement. Normally, you would prepare your design statements using your text editor, then have QDESIGN check the statements for errors. For this prototype, however, start work without the editor and enter your statements carefully.

Let's start designing the screen you will use to enter data in the ORDERS file. This screen will contain fields for all of your items in the ORDERS file. (Remember that you are only going to record supplier "code names" on the ORDERS screen; the purchasing department will track the suppliers and provide the correct code names on the SUPPLIERS screen you will create shortly.)

First, name your screen. We'll call it PURCHASE. Enter

```
> SCREEN PURCHASE
```

To tell QDESIGN that you want the ORDERS record structure, enter

```
> FILE ORDERS
```

Usually file and record structure names are identical. However, there may be times when you want two record structures in one file. That's why you must tell QDESIGN the name of the record structure you want.

That's all you really need before you can generate a screen, but let's add two small refinements. First, to allow QUICK to check entries on this screen against the SUPPLIERS file automatically, enter

```
> FILE SUPPLIERS REFERENCE
```

Then give your screen a title and center it at the top by entering

```
> TITLE "PURCHASE MANAGEMENT SCREEN" CENTERED
```

Now tell QDESIGN to go to work. Enter

```
> GENERATE
```

QDESIGN generates

```
> FIELD ORDERNUMBER OF ORDERS REQUIRED NOCHANGE  &
>   LOOKUP NOTON ORDERS
> FIELD DEPT OF ORDERS REQUIRED NOCHANGE
> FIELD SUPPCODE OF ORDERS &
>   LOOKUP ON SUPPLIERS
> FIELD GOODS OF ORDERS
> FIELD COST OF ORDERS
> FIELD DATEORDERED OF ORDERS
> FIELD DATEDELIVERED OF ORDERS
```

QDESIGN consults your dictionary for descriptions of the fields and makes several assumptions that result in an automatically formatted screen.

In this example, fields representing index segments are classified as REQUIRED and NOCHANGE. This means that you cannot skip any of these fields when entering data, nor can you change the data once it has been put on file.

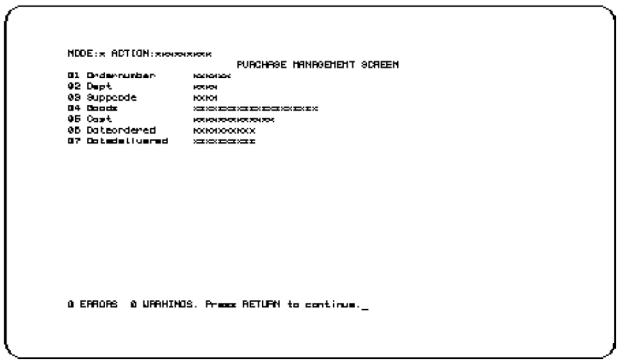
You made ORDERNUMBER a unique index, so QDESIGN has assigned it the LOOKUP NOTON option. Now, each time you add a new record to the file, QUICK will look up existing records in the ORDERS file to make sure the ORDERNUMBER is not on file already.

Because you declared the SUPPLIERS file as a REFERENCE file, QDESIGN added the option LOOKUP ON SUPPLIERS to SUPPCODE, which is in both files named in the screen design. As a result, QUICK checks each SUPPCODE entry made on this screen to see if that code actually exists on the master list of suppliers.

To have QDESIGN build a screen based on these specifications, enter

```
> BUILD
```

You'll see this



Notice that QDESIGN automatically created labels for each field based on its name. For example, the label for the ORDERNUMBER field is Ordernumber.

QDESIGN displays the layout of your screen, complete with ID numbers and labels for each field, and Xs to show the size of each field. If the message on the screen indicates there are errors, you may have entered your statements incorrectly. If there's an error, press Enter or Return, enter CANCEL CLEAR, and enter your statements again from the beginning or use REVISE.

You can use the REVISE statement at any time to correct your statements in a text editor.

Before going any further, you should save a copy of the correct QDESIGN source statements in a text file in case you want to change the statements later. You might want to build a similar screen or modify the design. You can give this text file the same name as you gave the screen. Press Enter or Return to continue, then on OpenVMS, UNIX, Linux, and Windows, enter

```
SAVE PURCHASE
```

Any time you want QDESIGN to use these statements, enter USE, followed by the file name, as in

```
USE PURCHASE
```

The next step is to build a screen for your other file, SUPPLIERS. Before you do that, clear QDESIGN's temporary save file. Enter

```
CANCEL CLEAR
```

QUICK Tricks

Following the steps outlined so far, you could easily build a second screen for your SUPPLIERS file. Since the SUPPLIERS file contains only two items, the screen would have a lot of empty space. It is much more efficient to group or "cluster" several records onto one screen. Clustering speeds up the process of entering and finding data.

First, name your screen. This screen is really just a list of suppliers, but it might be confusing to give it the same name as your SUPPLIERS file. Let's call the screen SUPPLIST.

```
SCREEN SUPPLIST
```

Specify the record structure name, SUPPLIERS, and tell QDESIGN you want to put five copies of this record structure's items on one screen, by entering

```
FILE SUPPLIERS OCCURS 5
```

To be consistent, you should also give this screen a title. You could center it as you did on the last screen. This time, however, position the title manually on the second line from the top of the screen and 25 columns in from the left, by entering

```
TITLE "SUPPLIER LIST" AT 2,25
```

The next statement works with the OCCURS option, which tells QDESIGN to begin a cluster.

```
CLUSTER OCCURS WITH SUPPLIERS
```

A blank line between each group of fields would make the screen easier to read. Tell QDESIGN to skip an extra line before starting the next group.

```
SKIP 1
```

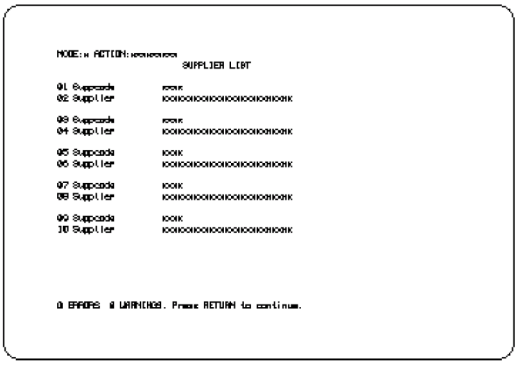
Now, using the FIELD statements from the previous screen as models, enter

```
FIELD SUPPCODE REQUIRED NOCHANGE LOOKUP NOTON SUPPLIERS
FIELD SUPPLIER
```

To complete the process, enter

```
BUILD
```

You'll see this:



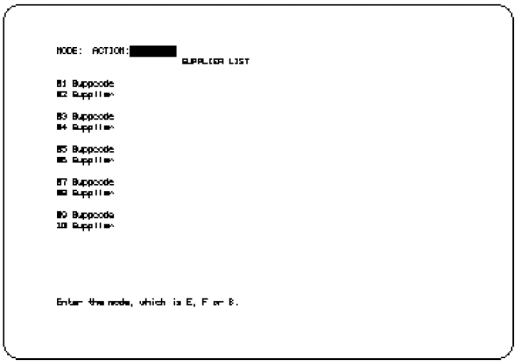
Press Enter or Return to continue. Now you have two screens. To save this screen design the same way that you saved the previous one, on OpenVMS, UNIX, Linux, and Windows, enter

```
SAVE SUPPLIST
```

At this point you can call up QUICK and your current screen, SUPPLIST, simply by entering

```
GO
```

Your SUPPLIST screen appears as follows:



QUICK displays the screen with the cursor positioned in the Action field, which normally appears at the top of every QUICK screen. When you're ready to return to QDESIGN from the QUICK screen, enter a caret (^). Using GO is an easy way to test a screen during development.

Entering Data

Now you're ready to start data entry using QUICK. To leave QDESIGN, enter

```
EXIT
```

To activate QUICK and tell it what dictionary you want to use, enter

```
quick dictionary=podict
```

If QUICK does not recognize the terminal that you are working on, it prompts you to enter the name of your terminal type. If you're unsure of what to enter here, enter a question mark (?) to see a list of acceptable terminal types. If you're still not sure what to enter here, check with your system manager.

QUICK prompts you for a screen name. Enter

```
SUPPLIST
```

To test your SUPPLIST screen, try entering some data from the list of suppliers.

Future Industries Supplier Codes

Code	Supplier
ACME	Acme Office Supplies Ltd.
BEST	Best Typewriter Repairs
FRST	Firstclass Travel Agency
PERF	Perfect Printing Inc.
WAKE	Wake-up Coffee Caterers

Enter E in the Action field. The cursor moves to the first field, SUPPCODE

Enter the code for the first supplier on the list, ACME. Your entry remains on display and the cursor moves to the next field, SUPPLIER. Enter the supplier's full name, Acme Office Supplies Ltd.

Enter data for the remaining suppliers on this screen in the same way. When you're finished, enter // (two forward slashes) in a blank field to get back to the Action field. If you fill all the fields on the screen, you'll be returned to the Action field automatically. When you're finished, enter UR (Update Return) in the Action field to save your data and exit the SUPPLIST screen.

QUICK prompts you for a new screen name. Now you can call up your other screen. Enter

```
PURCHASE
```

You can start entering some sample data on this screen.

Future Industries Department Codes

Code	Department
ACCT	Accounting
ADMS	Administration
DATA	Data Processing
HRES	Human Resources
MKTG	Marketing

Use the codes for the suppliers and departments from the lists provided, then create some order dates from before and after the end of Future Industries' fiscal year, January 1, 1999. The default format for dates is YYYYMMDD. You must enter a century, or use separators to identify the components of the date, as in 99/01/01.

Since this information reflects the current records of your business, leave a few Received Date fields blank. Use dates that are both before and after January 01, 1999.

Enter costs that will total a few thousand dollars for each department. You should enter at least ten records to give QUIZ and QTP something to work with in the next two chapters. Before you begin, let's review how to move around in QUICK screens.

To leave a field blank, press Enter or Return to skip to the next field. If the field is a required field, QUICK won't let you skip it. You'll get a message telling you that you have to enter something in the field.

QUICK also notifies you if your entry doesn't match the size or type you specified in your dictionary, or if you enter an order number that is already on file. Under these circumstances, you are prompted again for a valid entry.

QUICK won't accept an incorrect supplier code. It checks each entry against the SUPPLIERS file because you specified a LOOKUP ON option for this field. When you get to the SUPPCODE field, try entering a code that isn't on file and see how QUICK handles it.

Once you've entered your sample data and you're back in the Action field, check your entries to ensure they're what you want. If you want to change a field, enter the ID number of the field into the Action field. The ID number is the two-digit number beside the label. QUICK will prompt you in the field. Once you enter the new value, you're immediately returned to the Action field where you can make more changes if needed.

When you're ready to send the information on any screen to your files, you need to enter U (Update) in the Action field. QUICK updates the data on the files, clears the fields, and moves the cursor to the first field, ready for another record. When you are finished adding records and you've done the final update, enter UR (Update Return) in the Action field to save your data and exit the screen. If you entered U and are being prompted in a field, you can get back to the Action field by entering a caret (^) in a blank field.

Quick Can Help

Suppose you don't know or can't remember what to enter in a particular field. Anytime you're not sure what to do, just enter a question mark (?) to get a brief help message or a list of abbreviated commands at the bottom of the screen. If that's not enough information, enter two question marks (??). QUICK clears your screen and displays a more detailed explanation. You can add your own help messages to the data dictionary for each element as a further aid to users.

Finding and Modifying Data

Find Mode (F)	Finding your records again is easy. Call up the screen you want, and enter F (Find) in the Action field. This puts the screen in Find mode. QUICK moves the cursor to the field that corresponds to the index you defined first. On your PURCHASE screen, that's ORDERNUMBER. You can either enter an order number to tell QUICK to display the record on file for that number, or press Return to skip to the next index field, which is DEPT. Enter a department code in the DEPT field, and QUICK displays all of the records containing that code.

If you skip past all the fields that correspond to indexes, without making an entry (or if a screen has no fields that correspond to an index), QUICK displays the first record it finds. Keep pressing the Return key to see all of your remaining records sequentially, one at a time.

Select Mode (S)	Enter S in the Action field to find records using Select mode. In this mode, QUICK again prompts you first for fields that correspond to indexes, and then returns you to the Action field. You can now move to any data field you want by typing the ID-number of that field in the Action field. Enter a value that you want QUICK to select. Press Return in the Action field to start QUICK searching for records with matching item values. Whichever method you use to find the records you need, you can make any changes you want by following the same procedure as you used to correct data. Don't forget to update your file by entering U once you finish making changes to your data. You cannot change an entry that has the NOCHANGE option on its FIELD statement. On the PURCHASE screen, such entries are ORDERNUMBER and DEPT—the two items for which you declared indexes.

Deleting Records

Delete (D)	To delete a record from your screen, use the D (Delete) command. As protection against accidental deletion, the record isn't actually deleted from the file until you enter U or UR.

To leave QUICK and return to the operating system or Shell prompt, enter two carets (^^) in the Action field.

QUICKer and Better

That's all you need to know to design and use basic QUICK screens. But that's not all there is to QDESIGN. Programmers can use QDESIGN to build sophisticated systems of screens (and save months of programming time in the process). There are many QUICK features that you can use to make your screens operate more efficiently.

You can position labels and fields where you want them and draw lines to make it easier to identify groupings of fields. You can also highlight fields in different ways based on their status and value.

When you are creating a new screen, it's a good idea to write your design statements first using the text editor. To test your design, simply tell QDESIGN to use your text file by entering USE followed by the name you gave the file. It's equally easy to first access QDESIGN and then use the REVISE statement to access your text editor without leaving QDESIGN. When you finish editing your statements, QDESIGN compiles them automatically.

You can also make QUICK do some of the entry work for you. For example, if the item DATEORDERED always starts out as today's date on a new record, you could have QUICK enter this for you automatically. You can add options to prevent users from changing data in important fields or from deleting records from files.

Advanced Quick

QUICK is extremely flexible. At a more advanced level, QUICK enables you to modify the procedures that accept, update, find, and delete data. With QUICK you can also

- perform and display calculations before a record is updated.
- perform summing and balancing.
- specify security.
- highlight important fields.
- perform value editing based on values for other fields.
- skip data entry fields if a predetermined condition is not met.
- design screen systems for entry of multiple-file transactions.
- build screens that are called up automatically when the entry sequence reaches a certain field on the active screen.
- design menu screens to access all of your screens, and to connect QUICK to the other PowerHouse 4GL components, QUIZ and QTP.

For more information on these and other features, refer to the *IBM Cognos PowerHouse 4GL QDESIGN Reference*.

The QUICK User Interface

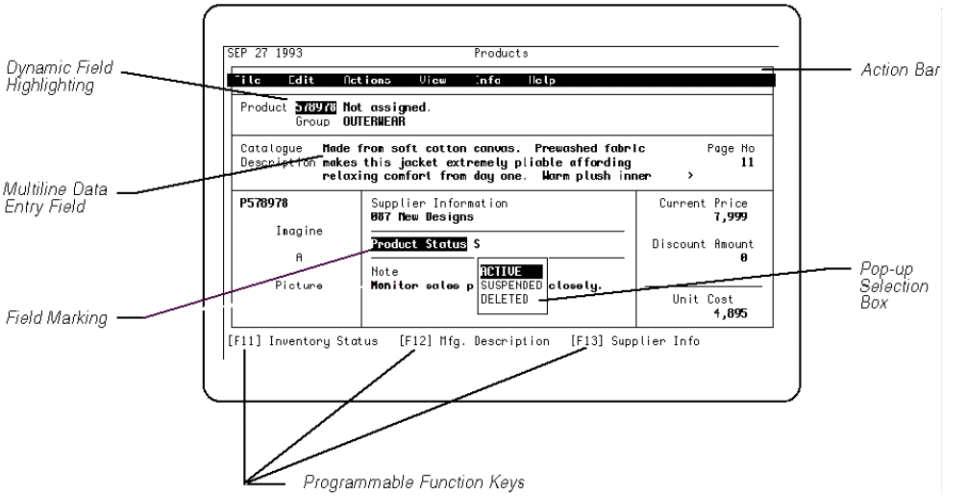
The screens you've built so far are based on the simplest QUICK statements. However, QUICK has many advanced features that allow screen designers to create superior and intuitive user interfaces.

These features help increase data entry accuracy and ease of use through QUICK. For example, you can

- create Action bars with pull-down menus.
- specify pop-up selection boxes.
- change the appearance of a field based on its value (for example, an invalid data entry can be set to blink).
- design pop-up windows that facilitate data entry in long text fields.

For Example

The following page shows how the QUICK User Interface can streamline the data entry process. Explaining how to implement most of the features illustrated is beyond the scope of this introductory manual, but we'll show you the end results that you can achieve with QUICK.



Action Bars are a convenient way to enter QUICK Action commands. Commands can be divided into logical groups, simplifying the end-user interface.	*Dynamic Field Highlighting* is used to draw the user's attention to unusual conditions — for example, a request for more of an item that is not on hand in inventory.	*Programmable Function Keys* are an ideal method of providing commands that are available throughout the entire application — you could, for example, define a function key to provide context-sensitive help.
Pop-up Selection Boxes allow the user to choose from a list of acceptable values — this simplifies the data entry process, and improves accuracy.	*Multiline Data Entry Fields* are ideal for allowing the QUICK screen user to enter detailed information — such as additional comments on a backordered item — without tying up large amounts of screen space.	*Field Marking* allows the user to quickly enter Action commands, including field ID numbers, without having to return to the Action field.

As you can see, the QUICK User Interface provides powerful options that enable screen designers to dramatically affect the QUICK user interface.

QDESIGN's advanced features are covered in the *IBM Cognos PowerHouse 4GL QDESIGN Reference*.

The Web Made Easy

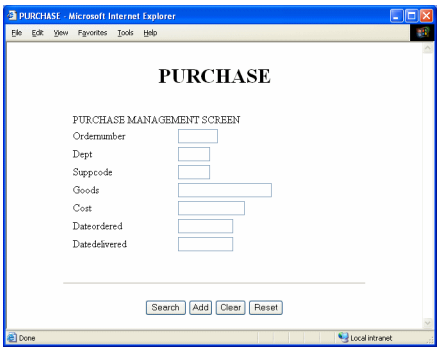
PowerHouse Web lets you deploy screens built using QDESIGN as Web browser pages. All you need to do is add HTML to the SCREEN statement as in

```
> SCREEN PURCHASE HTML
```

Or you can use the more Web-like form

```
> PAGE PURCHASE HTML
```

When you enter BUILD, QDESIGN generates an HTML template as well as building the screen. You can now use this page from a Web browser to add, search, change, and delete data over the Internet. Here's what the default page looks like.



The *IBM Cognos PowerHouse Web Planning and Configuration* document and the *IBM Cognos PowerHouse Web Developer's Guide* describe in detail how to set up the various PowerHouse Web components and how to create PowerHouse Web applications.

For the moment, though, let's get back to PowerHouse 4GL. You have all your information on file. You can put it to use with QUIZ, the PowerHouse 4GL report writer.

Chapter 4: QUIZ Has the Answers

When information is well-organized it is easily accessible. Access is the key to QUIZ, the IBM Cognos PowerHouse 4GL report writer. QUIZ puts all the answers right at your fingertips.

QUIZ has access to all the information you entered using your QUICK screens, as well as access to the rules you wrote in the dictionary. Now you can use QUIZ to analyze, organize, and synthesize this information.

You can produce a simple QUIZ report with only three statements:

ACCESS	names the files to be read
REPORT	lists the items to be included in the report
GO	produces the report

QUIZ automatically formats the report for you, or you can customize it with additional statements. For instance, you can add a title page and specify your own headings, footings, and spacing. You can also include subtotals or running totals in the report.

Here's an example. Assume that the Vice-President of Finance at Future Industries is preparing budgets for the next fiscal year. Many purchase orders have been issued for goods that won't be delivered or paid for until sometime in the new fiscal year. To budget accurately, the Vice-President must know how much money this involves. You've been asked to produce a report on unfilled orders that should include the order number, the department, the date the goods were ordered, a description of the goods, the supplier code, the cost of each order, and a subtotal for each department.

To get started, at the operating system or shell prompt enter

```
quiz dictionary=podict
```

When the prompt (>) appears, QUIZ is ready for you to begin.

Building Your Report

First, use the ACCESS statement to tell QUIZ which file you want to draw your information from. Enter

```
> ACCESS ORDERS
```

If you make a mistake, just re-enter the line, and the previous line is automatically canceled.

If you wanted to see a report of all of the purchase orders, you could simply enter the following statements

```
> REPORT ALL
> GO
```

QUIZ would provide a default layout and list all of the items in the ORDERS record structure horizontally across the page. It would even wrap automatically if there were too many items to fit on one line. Each record would start on a new line.

However, we'll create a more targeted report using the SELECT statement. Instruct QUIZ to select only those purchase orders that haven't yet been received, that is, orders with no delivery date

```
> SELECT IF DATEDELIVERED = 0
```

Your next statement tells QUIZ to sort the information in the ORDERS file by department. This makes DEPT your sort-item for this report. You can specify more than one sort-item if you want. Without a SORT statement, your report would simply list the records in the order they were retrieved from the file, so tell QUIZ to

```
> SORT ON DEPT
```

You don't need all of the information on each record in the file, so tell QUIZ which items you want reported and in what order by entering

```
> REPORT DEPT ORDERNUMBER SUPPCODE DATEORDERED &
>   TAB 43 COST
```

You use an ampersand (&) to continue a statement that is too long for one line. We've also added a TAB setting to position the COST column to the right of the others.

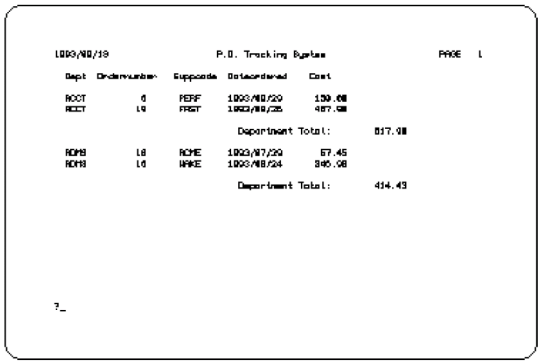
For the next statement, try something a little more sophisticated. Tell QUIZ to skip two lines, write "Department Total:" at the end of each department's listing, then to tab across to the COST column and put in a subtotal for that department. For readability, tell QUIZ to leave a blank line before starting on the next department. Enter

```
> FOOTING AT DEPT SKIP 2 TAB 35 "Department Total:" &
>   COST SUBTOTAL SKIP 2
```

The last statement in building any QUIZ report is the shortest. To see how your report looks, enter

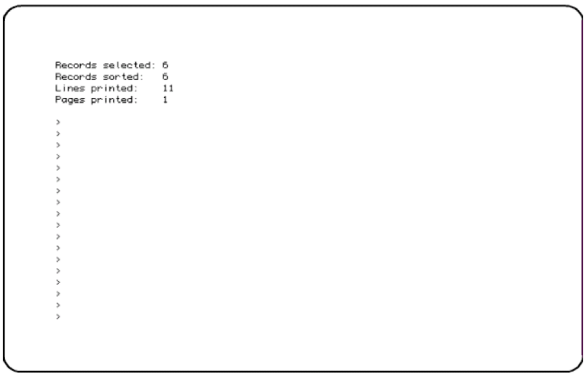
```
> GO
```

Here is what your report might look like:



Your own report will vary depending on the data you entered on your QUICK screens, but the format should be the same. To exit the report, press Return.

QUIZ provides you with a brief report summary:



Because you didn't specify column headings or spacing, QUIZ used information from your dictionary to provide them. The date, title, page number, commas, and decimal points were all added automatically too.

If QUIZ displays a question mark (?) at the bottom of the screen, it indicates that your report fills more than one screen. To see the next screen "page" press Return. To rerun your report, enter GO again.

If your report didn't turn out as you expected, you'll want to review your report statements. Enter

```
> SHOW COMMANDS
```

The SHOW statement displays your statements so you can check them. The statements should look like this:

```
> ACCESS ORDERS
> SELECT IF DATEDELIVERED = 0
> SORT ON DEPT
> REPORT DEPT ORDERNUMBER SUPPCODE DATEORDERED &
>   TAB 35 COST
> FOOTING AT DEPT SKIP 2 TAB 35 "Department Total:" &
>   COST SUBTOTAL SKIP 2
> GO
```

In just a few minutes, you produced a report listing all the requested information using QUIZ. In a few more minutes you can learn how to improve your report's appearance and readability, and how to prepare much more sophisticated reports. And, of course, you probably want that report on paper. We'll get to that, too, in a moment.

First, save this report so that you can use it again. Name the file BACKORD, since this is a report on goods ordered, but not yet received. Enter

```
>SAVE BACKORD
```

Any time you want QUIZ to produce this report, just enter USE and the file name:

```
> USE BACKORD
```

With QUIZ, you can learn very quickly to manipulate the data in your files to get the information needed for decision-making, when and how you need it.

Relational?

Relational database users can enter an SQL DECLARE CURSOR statement containing SQL syntax before the ACCESS statement. This creates a cursor containing the tables and columns that hold the information you want. In effect, it creates a customized view of your database that QUIZ can use to produce your reports. The ACCESS statement that follows names the cursor and any other data sources to be used in your report.

Alternatively, you can name the tables within the ACCESS statement by adding the database containing the table. For example, if ORDERS were in the PODB relational database, you would use

```
> ACCESS ORDERS IN PODB
```

If you use the subdict=search program parameter when starting QUIZ, QUIZ will search the databases in your dictionary for the specified file. You will not need to specify the database name, as in

```
quiz subdict=search dictionary=podict
 >ACCESS ORDERS
```

Some Basic QUIZ Rules

When you start building QUIZ reports on your own, keep these simple rules in mind:

- ACCESS is always the first statement in a QUIZ report. The last statement is usually GO.
- You can save a "compiled" version of your report that runs faster using the BUILD statement.
- Most QUIZ statements can appear only once. With few exceptions, repeating a statement cancels a previous statement of the same type. For example, a second REPORT statement cancels a previous REPORT statement. A second ACCESS statement, however, cancels all previous QUIZ statements. In effect, it tells QUIZ to start over.
- You must leave at least one space between each word in a statement.
- You can begin a line in any column after the prompt character (>), but it's a good idea to use indentation, as we have done in our examples, to make your statements more readable.

QUIZ Tips

If you don't like the way your report looks on the screen, or you make a mistake, you don't have to retype all the statements. Just add what's needed at the end of the statements. Once you've checked over your report on the screen and are satisfied with it, you may want to print a copy. Simply add this statement

```
> SET REPORT DEVICE PRINTER
```

followed by another GO statement to print the report. (You must repeat GO every time you want to re-execute your QUIZ report.)

To reset your report to display on your terminal, enter

```
> SET REPORT DEVICE TERMINAL
```

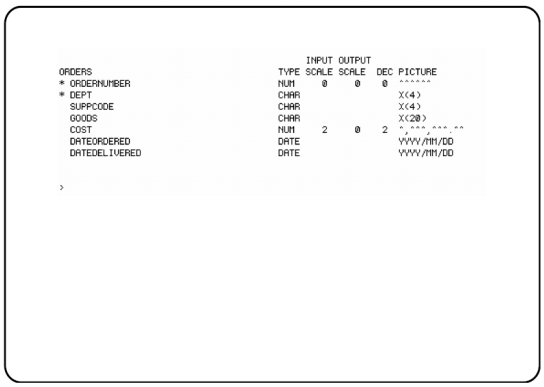
You can also use the SHOW command to list your files if you forget which files in your system are available to you. To see your files, enter

```
> SHOW FILES
```

QUIZ displays a list of file names on the screen. If you want to see the items in the file ORDERS, enter

```
> ACCESS ORDERS
> SHOW ITEMS
```

QUIZ displays them like this:



```
                                    INPUT OUTPUT
ORDERS                              TYPE SCALE SCALE  DEC PICTURE
* ORDERNUMBER                       NUM     0     0    0  ^^^^^^
* DEPT                              CHAR                  X(4)
  SUPPCODE                          CHAR                  X(4)
  GOODS                             CHAR                  X(20)
  COST                              NUM     2     0    2  ^,^^^,^^^.^^
  DATEORDERED                       DATE                  YYYY/MM/DD
  DATEDELIVERED                     DATE                  YYYY/MM/DD

>
```

Asterisks (*) indicate that indexes have been declared for the items.

More About QUIZ Statements

Let's try another report using the same data. Assume that you've been asked to produce a list of suppliers that shows how much each has been paid in the current year. The report should include each supplier's full name, the purchase order numbers, the order and received dates, the amounts, subtotals for each supplier, and an overall total. Using the traditional paper-based system, this task could take days. With QUIZ, it's a matter of minutes.

Before starting, let's clear the slate so that you can preserve the new report when you're finished. Enter

```
> CANCEL CLEAR
```

For simplicity's sake, the preceding QUIZ example dealt with only one file. In reality, you will probably have many files in your system. You can design a QUIZ report that draws information from several files using the LINK TO option of the ACCESS statement, as in

```
> ACCESS ORDERS &
>   LINK TO SUPPLIERS
```

QUIZ establishes linkage to the SUPPLIERS file by name matching. It examines the files previously declared in the ACCESS statement, seeking items that match an index by name in the "linked to" file. In this case, the index to the SUPPLIERS file has a single segment item named SUPPCODE, and the ORDERS file contains an identically- named item. Therefore, QUIZ links the two files via SUPPCODE and allows you to use information from both files in a single QUIZ report.

The SELECT statement allows you to specify groups or ranges of information to report. For example,

```
> SELECT IF DATEORDERED > 19990101 &
>   AND DATEDELIVERED > 0
```

This statement tells QUIZ to report only purchases that have been ordered and received after the first day of January 1999.

The greater than (>) and less than (<) symbols, are called "comparison operators". You can also use the abbreviations "GT" and "LT". Other comparison operators include

EQ or =	equal to
GE or >=	greater than or equal to
LE or <=	less than or equal to
NE or <>	not equal to

You can use comparison operators in both QUIZ and QTP statements.

You can sort on any item in any file linked by the ACCESS statement. QUIZ automatically sorts in ascending alphabetical or numeric order. If you want items sorted in descending order instead, just include the letter D after the sort-item, as in

```
> SORT ON SUPPCODE &
>   ON DATEDELIVERED D
```

Your first report repeated the department for each order it printed. That report would look a lot cleaner if each department was printed only when it first appeared. In this report you can use the PRINT AT option, which uses the sort-item to tell QUIZ to print each supplier code only the first time it appears:

```
> REPORT SUPPCODE PRINT AT SUPPCODE ORDERNUMBER &
>   DATEORDERED DATEDELIVERED GOODS COST
```

In your last QUIZ report, you used a FOOTING statement to show the subtotals. In this one, you can use a FINAL FOOTING statement to total all the amounts given in the report.

```
> FOOTING AT SUPPCODE &
>   SKIP 2 &
>   TAB 23 "Total paid to:" SUPPLIER &
>   COST SUBTOTAL &
>   SKIP 3
> FINAL FOOTING &
>   TAB 51 "Final Total:" &
>   COST SUBTOTAL
```

Now your report statements look like this:

```
> ACCESS ORDERS &
>   LINK TO SUPPLIERS
> SELECT IF DATEORDERED > 20060101 &
>   AND DATEDELIVERED > 0
> SORT ON SUPPCODE &
>   ON DATEDELIVERED D
> REPORT SUPPCODE PRINT AT SUPPCODE ORDERNUMBER &
>   DATEORDERED DATEDELIVERED GOODS COST
> FOOTING AT SUPPCODE &
>   SKIP 2 &
>   TAB 23 "Total paid to:" SUPPLIER &
>   COST SUBTOTAL &
>   SKIP 3
> FINAL FOOTING &
>   TAB 51 "Final Total:" &
>   COST SUBTOTAL
> GO
```

Try this QUIZ report for yourself. When you're done, save the report as your payments report. Enter

```
> SAVE PAYMENTS
```

There is one more statement that you should know. EXIT tells QUIZ to end the program. When it does, you return to the operating system.

You now have the knowledge you need to create fairly complex QUIZ reports. With a little practice, you can put this knowledge to work and you will find that QUIZ is a remarkably versatile tool.

Like all the PowerHouse 4GL components, QUIZ has progressively more complex features for more sophisticated applications. For example, with the SET SUBFILE statement, you can create self-describing files that can be used later, but that don't have to be entered in the dictionary. QUIZ also has the math skills to perform a variety of calculations, and include them in your report.

Summary operations, the DEFINE and SET SUBFILE statements, and other advanced features are covered in greater detail in the *IBM Cognos PowerHouse 4GL QUIZ Reference*.

What's Ahead

In the next chapter, you'll get an introduction to subfiles where you'll discover the transaction processing capabilities of QTP.

Chapter 5: QTP — Powerful Processor

QTP is the IBM Cognos PowerHouse 4GL transaction processor. It gives you the power to change the data in your files in one sweep. For example, you can use a single QTP run to increase the prices of your entire inventory list by six percent, or to change all your stock numbers.

QTP is easy to use because it employs much of the same vocabulary as QUIZ. Familiar statements such as ACCESS, SELECT, SORT, USE, GO, and EXIT perform the same functions in QTP as they do in QUIZ. And, of course, QTP works closely with the dictionary.

It is precisely because it is both powerful and easy to use that QTP must be treated with respect. Carelessly used, QTP could wipe out your files and records in the time it takes you to read this sentence.

With that word of warning, let's take a closer look at what QTP does, and then try a few QTP runs.

We've already mentioned how to use REVISE to make corrections in your system editor. You can also use a text editor to prepare your statements and then test them in QTP with the USE statement. Any errors are highlighted automatically by QTP. In fact, it's a good idea to create all your QTP runs this way, and try them out on test files before you use them on the real thing. You won't need to do that here, though. Our example has been well tested already.

Let's assume that it's the start of the fiscal year for Future Industries. You want to archive all old records by transferring them to a subfile. You can keep this subfile as a permanent archive for old records and add to it as you need to. It takes only minutes with QTP. (QUIZ and QTP subfiles work the same way; the subfile you're about to create can be used by QUIZ.)

At the operating system or shell prompt, enter

```
qtp dictionary=podict
```

When the prompt (>) appears, QTP is ready. To begin, give your QTP run a name by entering:

```
> RUN YEAREND
```

A run can consist of many tasks. Each task is called a request and should have a name for easy identification. Enter

```
> REQUEST ARCHIVE
```

Next, tell QTP which file you want to use by entering

```
> ACCESS ORDERS
```

This is your "input" file.

To select all orders received prior to the new fiscal year, enter

```
> SELECT IF DATEDELIVERED < 19990101 &
>   AND DATEDELIVERED NE 0
```

Now tell QTP to create a subfile. Since it's a subfile of ORDERS, let's call it OLDORD:

```
> SUBFILE OLDORD KEEP INCLUDE ORDERS
```

The KEEP option tells QTP to retain the subfile for later use. You've also instructed QTP to include the records from the ORDERS file.

Finally, tell QTP you want to delete the selected records from the ORDERS file:

```
> OUTPUT ORDERS DELETE
```

There's your QTP run. Now enter

```
> GO
```

At this point, QTP displays information about the run, such as the number of records read, the number of transactions processed, and so on. The message "Finished" tells you that the run is complete.

QTP Output

The previous QTP request used only one of QTP's output functions, DELETE. There are three others:

UPDATE	changes the information on existing records
ADD	adds a new record, or records, to the file
ADD UPDATE	adds a new record or changes the record if it already exists

Relational?

As in QUIZ, you can enter an SQL DECLARE CURSOR statement containing SQL syntax before the ACCESS statement. When you enter an SQL DECLARE CURSOR statement, QTP creates a cursor and opens a relational database for read access. QTP can also use three SQL statements to change data in the relational database: SQL INSERT, SQL UPDATE, and SQL DELETE.

Alternatively, you can name the tables within the ACCESS or OUTPUT statement by adding the database containing the table. For example, if ORDERS were in the PODB relational database, you could use

```
> ACCESS ORDERS IN PODB
```

or

```
> OUTPUT ORDERS IN PODB UPDATE
```

The QTP Edit

You can use the QTP EDIT statement to find errors in your data. Suppose you've received a bulk data input from another computer system. You want to make sure that it won't introduce errors into your system.

An EDIT statement tells QTP to check records in any files you access to make certain that the values were entered correctly. QTP does this for you, using either editing criteria in your dictionary or values specified in the EDIT statement itself.

If you regularly receive this kind of input, you should probably include value checking in your dictionary so that this job is done automatically.

In the following request, EDIT ALL tells QTP to edit all the data in the input file according to element definitions in the dictionary. The second EDIT statement tells QTP which department names are correct:

```
> RUN NEWDATA
> REQUEST TWO
> ACCESS ORDERS
> EDIT ALL
> EDIT DEPT &
>   VALUES "ACCT", "ADMS", "DATA", "HRES", "MKTG"
> GO
```

QTP searches the file for any items that don't meet the data dictionary specifications and any department names not matching the five you specified, then lists any errors on your screen. To get a printout of the incorrect records, add the following statement to your request.

```
> SET REPORT PRINTER
```

You can create many such runs using the system text editor and use them periodically to check that no errors have crept into your files.

If you didn't use the editor to create your run, you can still keep it for future use. The SAVE statement puts a copy of your QTP run on file under any name you choose. The BUILD statement saves a "compiled" version of your run that executes faster.

More QTP Power

The subfile is another QTP feature for the advanced user. You've already seen one use for it, but it can also be used to simplify complex QTP and QUIZ tasks by breaking them down into several parts, and to do part of the sorting or summarizing of information.

We've examined only a few of QTP's features in this primer. There's a lot more QTP can do to help you keep your files in order and up to date. Like QUIZ, QTP can subtotal, count records, calculate averages, and find maximum and minimum values.

For more information on these and other powerful transaction processing features, refer to the *IBM Cognos PowerHouse 4GL QTP Reference*.

You leave QTP the same way you leave QUIZ — just enter

```
> EXIT
```

What's Ahead

That completes the components of your purchase order tracking system. It's ready to go to work. But that doesn't mean there's nothing more to do. In the next chapter you'll learn how to put the finishing touches to the system by adding a menu screen that allows the system's users to select the functions they want to use.

Chapter 6: The Finishing Touches

Now that you've finished all the pieces of your purchase order tracking system, this chapter will show you how to put them together. You'll create a menu that automates the start-up of each part of the application to give your application a truly professional look.

This chapter also involves a little experimenting with screen design. At this stage of application development, you'll find the REVISE statement to be an invaluable tool. With REVISE, you can access the text editor and add or edit QDESIGN statements without leaving QDESIGN.

A Simple Menu

A menu offers users a set of choices. To create a menu for your purchase order tracking system, you must determine what choices will appear on the menu. Your purchase order tracking system currently consists of two screens (PURCHASE and SUPPLIST) and two standard reports (BACKORD and PAYMENTS). These screens and reports will be the four options on your menu. Your menu options will give users the ability to call up the PURCHASE or SUPPLIST screens to enter data, and the ability to produce the BACKORD or PAYMENTS reports in QUIZ.

To create the menu, you'll use QDESIGN. At the operating system or shell prompt, enter

```
qdesign dictionary=podict
```

When the prompt (>) appears, QDESIGN is ready. First, name the screen and designate it as a menu screen.

```
> SCREEN POMENU MENU
```

Next, tell QDESIGN to skip two lines and give the menu a title. Enter

```
> SKIP 2
> TITLE "Purchase Order Tracking System" CENTERED
```

The CENTERED option centers the title on the screen. Before you begin identifying the menu options, leave another blank line after the title. Enter

```
> SKIP 1
```

To set up your menu screen's first option, which calls up the Purchase Order screen, enter

```
> SUBSCREEN PURCHASE LABEL "Purchase Order Screen"
```

The SUBSCREEN statement tells QUICK to load a new screen from the current screen. In this case, the current screen is POMENU. When you exit from the subscreen, you'll return to the menu. The LABEL option determines what appears on the screen. The text for the label must be included in single or double quotation marks.

To set up the suppliers list screen as the second option on the menu, use the SUBSCREEN statement as you did to create the first option.

```
> SUBSCREEN SUPPLIST LABEL "Suppliers Screen"
```

That's all you need to do to create option 2.

The third and fourth menu options identify your QUIZ reports. You want option 3 to call up QUIZ and run the BACKORD report.

To do this, you need to use the COMMAND statement. Enter

```
> COMMAND "quiz dictionary=podict auto=backord" LABEL &
>   "Back Order Report" CLEAR ALL
```

The operating-system command, quiz dictionary=podict auto=backord, tells QUICK to start QUIZ using the podict dictionary and immediately execute the BACKORD report when a user selects option 3 from the menu. You must put operating system commands in quotation marks when you use them in a COMMAND statement.

CLEAR ALL tells QUICK to clear the terminal memory before invoking a command. When you call a command or program from a higher-level screen such as the menu, QUICK does not automatically clear or rewrite terminal memory.

Repeat this procedure to have option 4 call up the PAYMENTS report. Enter

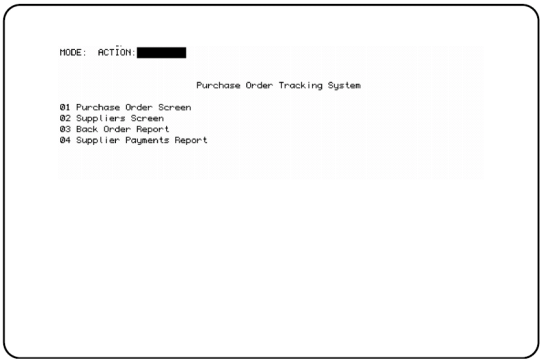
```
> COMMAND "quiz dictionary=podict auto=payments" LABEL &
>   "Supplier Payments Report" &
>   CLEAR ALL
```

Now all that's left is to tell QDESIGN to build the screen:

```
> BUILD
```

Your initial version of the menu is finished. Press Return to continue. To test it, enter

```
> GO
```



Choose option 1 from your new menu and the PURCHASE screen appears. To return to the menu, enter a caret (^) in the Action field.

Now, try the other options. When you choose either of the reports, option 3 or 4, IBM Cognos PowerHouse 4GL loads QUIZ and displays the report on the screen. Enter EXIT to return to the Purchase Order Tracking System menu.

When you're finished with the menu, enter a caret (^) in the Action field to return directly to QDESIGN. At this point, it is a good idea to save your new menu screen. On OpenVMS, UNIX, Linux, and Windows, enter

```
> SAVE POMENU
```

You have now created a functional prototype menu which ties together all the components of the Purchase Order Tracking System. In the rest of this chapter, we'll add some new features to the user interface, making it more convenient to use.

Enhancing the User Interface

The most efficient way to edit your QDESIGN statements is to use the REVISE statement to activate the system text editor. Enter

```
> REVISE
```

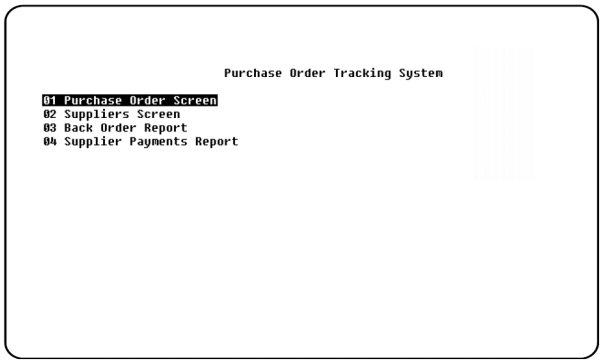
With three simple options to the SCREEN statement, you can completely change the way the menu screen operates. In the original menu screen, to select a menu item, you entered the ID-number in the Action field. With PowerHouse 4GL's powerful user interface features, you can select a menu item by highlighting it with the cursor keys and pressing Return to take action.

Change your screen statement to look like this:

```
SCREEN POMENU MENU NOMODE NOACTION FIELDMARK
```

That's all there is to it. Now, return to QDESIGN. Your edited statements from the revised file are parsed. When you are prompted to create a new POMENU file, enter Y to save your new compiled screen.

To test how your menu works now, activate QUICK by entering GO at the QDESIGN prompt.



Try moving between the menu options with the up and down arrow keys. Each selection is highlighted when you move to it. To select an item, highlight it and press Return.

On UNIX, Linux, and Windows, pressing F8 will return you to QDESIGN. On OpenVMS, press PF4. EXIT returns you to the operating system or shell prompt.

Congratulations! Your purchase order tracking system is finished.

What's Ahead

That's the end of the beginning. In the final chapter, we'll introduce some ideas that will expand and enhance your abilities with PowerHouse 4GL in the future.

Chapter 7: Ideas Unlimited

The application that you developed through the last six chapters isn't a production system. As a working purchase order tracking system it has some shortcomings, but the intention was not to build a perfect purchase order tracking system. Rather, this prototype was designed to demonstrate just how easy it is to create a useful, working application in a short time using IBM Cognos PowerHouse 4GL. In a word, productivity.

With more time and practice, you can further develop this system by adding refinements to improve the operation of the screens and the appearance of the reports. Your purchase order tracking system is, however, a working system that could well have been developed by anyone, with only a limited knowledge of PowerHouse 4GL.

PowerHouse 4GL is not just an application generator. Like any language, it's an idea generator too. Once you're familiar with its many uses, you'll quickly learn to develop your own applications. You'll find new ways to make your work more efficient, perhaps even do things you couldn't do before.

PowerHouse 4GL has the potential to increase your productivity many times over. To write an application using PowerHouse 4GL takes, on average, only one-tenth of the time that the same job would require if written in one of the traditional programming languages such as COBOL or RPG. Because you spend less time on routine assignments, you have more time for the more creative and productive parts of your job.

PowerHouse 4GL is an ideal system for producing applications quickly. For an urgent one-time job, for instance, you don't have to worry about formatting or the physical appearance of the work, because PowerHouse 4GL always produces results that are presentable.

PowerHouse 4GL is designed to work the way an application developer works. It's especially helpful in situations where the people asking for the application really aren't sure what they want in the early stages. With PowerHouse 4GL you can modify the application and have it back to the user in hours, not days or weeks.

The End of the Beginning

PowerHouse 4GL has been used for a remarkably wide variety of applications. It is used by airline companies and advertising agencies, by hospitals and government, by scientific research organizations—even by other software companies to develop their own products.

In fact, we like to think that the only real limitation on PowerHouse 4GL is the user's imagination. So be creative. You have nothing to lose, and a lot to gain.

In this brief introductory guide, you have seen only a glimpse of PowerHouse 4GL. There is a comprehensive documentation covering each PowerHouse 4GL component. The manuals are written and designed to meet the needs of computer users at all levels, and are extensively indexed. For a complete list of the IBM Cognos PowerHouse 4GL documentation, see Chapter 1 of this document.

Glossary

Action bar	An alternative to the traditional Action field entry mechanisms.
Action field	A special field in the top left-hand corner of every PowerHouse 4GL screen created with default user interface options. Commands entered here control what the QUICK-screen user can do on the screen.
allowed syntax	The syntax PowerHouse 4GL allows you to enter.
application	A set of programs designed to solve a specific problem or meet a specific need.
attribute	A characteristic or property (such as size) that you assign to a particular entity.
cluster	A group of fields that repeats on one screen.
cursor	A cursor is the name of a set of data declared on a DECLARE CURSOR statement.
compile	In PowerHouse 4GL, the actions required to transform a source statement file into the compiled file containing the tables that control processing.
compiled file	An executable PowerHouse 4GL file.
component	Refers to one of the PowerHouse 4GL programs: PDL, QSHOW, QUTIL, ETOP, QDESIGN, QUICK, QUIZ, and QTP.
data definition	The characteristics of an application's data, which is stored in the data dictionary. The data definitions determine how PowerHouse 4GL accesses data for the application.
data dictionary	A storehouse of information about the data that you use in your applications.
data file	The characteristics of an application's data as stored in the data dictionary. The data definition determines how the data is accessed and formatted by the application.
data item	See item.
data record	See record.
datatype	The way an item is stored in a record.
default	An automatic response built into a program to ensure that appropriate actions are performed or that acceptable values are provided. Defaults can be overridden by the user if desired.
dictionary	See data dictionary.
dictionary definition	The data definition, security specification, system-wide standards, and other information stored in a data dictionary.

element	The smallest category of data described in the data dictionary. Elements are the basic building blocks of a PowerHouse 4GL application since they represent many individual values. It is physically represented in a record-structure by a record item. It is physically represented in an index structure by a segment. (See also item and segment.)
entity	The building blocks of a PowerHouse 4GL application. Entities include files, elements, items, records, and record-structures. They are described in the data dictionary.
field	An item declared in a FIELD statement in QDESIGN. It becomes a location on a QUICK screen used for entering, finding, changing, and deleting data. (See also required field.)
file	Declares a relation, table, or view on a screen, run, or report.
ID-number	A number identifying a field or group of fields on a PowerHouse 4GL screen.
index	A data structure used by relational systems or indexed file systems to locate records quickly. The index for each record contains an index value and the address for the rest of the information associated with that index value.
indexed file	A type of file organization in which all the information in a record is associated with the value of the index or segment of an index in the record. For an indexed file, the operating system automatically creates and maintains an index.
item	An entity in a record-structure that holds a value.
menu screen	Serves as a table of contents for other screens, programs, or commands.
Mode field	A special field found in the top left-hand corner of every PowerHouse 4GL screen with default user interface options. The Mode field indicates the current mode: E (Enter), F (Find), or S (Select).
option	In syntax, refers to features of a particular statement that a user can select.
primary file	See primary record-structure.
primary record-structure	A record-structure whose information is most important to the screen, report, or request. In QUIZ and QTP, the primary record-structure is the first record-structure named in the ACCESS statement. In QDESIGN, it is either the first record-structure named in a screen design (that is not received from a higher-level screen) or the record-structure labeled primary.
record	One set of the items in a record-structure and their values. A record is stored in a file.
record-structure	An ordered collection of items. Each record-structure is associated with exactly one file. For example, the record-structure of a file used to store a mailing list of your customers might consist of several items, including NAME, ADDRESS, POSTALCODE, and PHONENUMBER.
relational database	Data in a relational database is organized into tables which are made up of rows and columns. No physical linkages between tables are specified.

required field	A field on a PowerHouse 4GL screen in which an entry is required.
segment	An item that's part of an index. Each index is composed of one or more segments.
statement	A line of instructions to one of the PowerHouse 4GL components that can either be entered in response to the prompts or entered into a text editor file.
subfile	A self-describing data file; that is, it's a file that contains both data and the information that describes the data. A subfile is not described in a PowerHouse dictionary.
syntax	PowerHouse 4GL language, statements and commands, containing specific rules and conventions (for example, the use of case, brackets, slashes) that enable communication to take place between the user and the computer.

Index

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V