

Developer Self-Paced Training Guide
HP ALLBASE/4GL



HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Information Management Series



HP ALLBASE/4GL

For MPE XL Systems

Developer Self-Paced Training Guide



Australian Software Operation

Part Numbers

30601-90013 E0989 (Text only)

30601-64003 (Manual kit)

Printed in Australia

September 1989

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD PROVIDES THIS MATERIAL "AS IS" AND MAKES NO WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HEWLETT-PACKARD SHALL NOT BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS) IN CONNECTION WITH THE FURNISHING, PERFORMANCE OR USE OF THIS MATERIAL WHETHER BASED ON WARRANTY, CONTRACT, OR OTHER LEGAL THEORY.

Some states do not allow the exclusion of implied warranties or the limitation or exclusion of liability for incidental or consequential damages, so the above limitation and exclusions may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Australia Ltd.

Copyright © 1986, 1987, 1988, 1989 by HEWLETT-PACKARD AUSTRALIA LIMITED

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages for you to merge into the manual. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page.

The software product part numbers printed alongside the date indicate the versions and update levels of the software products at the time the manual edition or update was issued.

First Edition	October 1988	30601A
Second Edition	September 1989	30601A

List of Effective Pages

This list of effective pages gives the date of the most recent version of each page in this manual.

All pages September 1989

Preface

This manual is a self-paced training guide for application developers who are learning how to use HP ALLBASE/4GL. It uses a number of tutorial style lessons to provide an introduction to the features and facilities available in HP ALLBASE/4GL. While following these lessons, you will build a small working application with HP ALLBASE/4GL.

The lessons in this manual are designed to allow you to complete this training course at your own pace, without additional classroom instruction or assistance from an instructor. We don't suggest any specific time or duration for this course. Instead, we suggest that you proceed at a pace that makes you feel comfortable. However, since later lessons do depend on the results of the earlier lessons, we recommend that you proceed through these lessons in the sequence presented in this manual.

This manual contains a total of 12 chapters. Chapters 1 and 2 provide an introduction to HP ALLBASE/4GL, and the use of Hewlett-Packard terminals with HP ALLBASE/4GL. Chapters 3 to 10 describe the procedures to develop a working HP ALLBASE/4GL application using the inbuilt HP ALLBASE/4GL KSAM data file manager. Chapters 11 and 12 describe the procedures to develop a similar application using the interface from HP ALLBASE/4GL to HP ALLBASE/SQL.

This manual also contains one appendix, a glossary, and two indexes. The appendix outlines the procedure for defining a new application in the HP ALLBASE/4GL administrator. (If you are also the Administrator for your HP ALLBASE/4GL system, you will need to follow the procedures outlined in this appendix before you can start working on the training lessons in this manual.)

The glossary explains the meanings of some HP ALLBASE/4GL terminology. The first index is an index of names of the application components that you will create during this training course. The second index is an index to the topics and procedures described in this manual

A chart at the rear of the manual shows the structure of the HP ALLBASE/4GL developer menus.

Intentionally Blank

Table of Contents

- Chapter 1 Overview**
- Chapter 2 Getting Started**
- Chapter 3 The Application**
- Chapter 4 Fields, Records and Files**
- Chapter 5 Screens**
- Chapter 6 Logic**
- Chapter 7 Running Your Application**
- Chapter 8 Reports**
- Chapter 9 More Features**
- Chapter 10 Further HP ALLBASE/4GL Facilities**
- Chapter 11 Using HP ALLBASE/SQL with HP ALLBASE/4GL**
- Chapter 12 Further HP ALLBASE/SQL Techniques**

- Appendix A Starting a New Application**

- Glossary**

- Index of Names**
- Subject Index**

- HP ALLBASE/4GL Developer Menu Chart**

Intentionally Blank

Contents

Chapter 1: Overview

You and HP ALLBASE/4GL	1 - 1
What is HP ALLBASE/4GL?	1 - 1
The HP ALLBASE/4GL System	1 - 2
The Operating System Interface	1 - 2
Data Manager	1 - 3
The Administrator	1 - 3
The Developer	1 - 3
The HP ALLBASE/4GL Administrator Environment	1 - 3
System Administration Responsibilities	1 - 4
System Specifications	1 - 4
System Security	1 - 4
End-User Environment	1 - 4
Run-Time Environment	1 - 4
How HP ALLBASE/4GL Developer Works	1 - 5
Working with HP ALLBASE/4GL	1 - 6
Data Definition	1 - 6
Field Specifications	1 - 7
Validation Items	1 - 8
Storage Items	1 - 8
Record Layouts	1 - 9
Database Items	1 - 9
Messages	1 - 9
Help Screens	1 - 10
Screen Development	1 - 10
Screen Types	1 - 10
Menus	1 - 10
Data Screens	1 - 10
Windows	1 - 11
Function Keys	1 - 11
Creating HP ALLBASE/4GL Screens	1 - 11
The Screen Painter	1 - 12
Prototyping with the Screen Painter	1 - 12
Logic Definition	1 - 13
Processes	1 - 13

Functions	1 - 14
SQL Logic Blocks	1 - 14
Decision Tables	1 - 14
Report Definition	1 - 14
Line Groups	1 - 15
Record Sorting and Selection	1 - 15
Control Breaks	1 - 15
Report Linkages	1 - 15
The Report Painter	1 - 16
From Here On	1 - 16
How This Guide Works	1 - 16
Terminology Used in This Manual	1 - 17

1 Overview



Welcome to the HP ALLBASE/4GL Self-Paced Training Guide. This guide will help you become familiar with using HP ALLBASE/4GL to develop applications.

You and HP ALLBASE/4GL

This guide assumes that you are already familiar with a conventional programming language. However, it does not assume that you have any experience with a programming productivity tool like HP ALLBASE/4GL.

HP ALLBASE/4GL is not an end-user tool. It's an application development system designed for professional programmers. It allows you to develop and maintain applications quickly and efficiently. HP ALLBASE/4GL is a powerful tool that will significantly improve your productivity.

HP ALLBASE/4GL uses a number of concepts that may well differ from the concepts you are familiar with when using a conventional programming language. This guide introduces you to these concepts and explains how to use them when developing HP ALLBASE/4GL applications.

What is HP ALLBASE/4GL?

HP ALLBASE/4GL is an advanced fourth generation programming language. It enables you to design and implement application software by defining the required results, rather than defining the procedures necessary to achieve those results.

HP ALLBASE/4GL allows you to:

- Develop new applications.
- Modify existing applications.
- Produce multiple versions of existing applications.

During this training course, you will create, and then modify some HP ALLBASE/4GL applications.

The HP ALLBASE/4GL System

HP ALLBASE/4GL is available in two versions: the full developer system, and the run-time environment. The developer system contains all the facilities necessary to develop applications, and to run them. The run-time environment provides the necessary facilities for application end users to run developed applications.

The developer system consists of the following major functional components:

- The operating system interface.
- The data manager.
- The administrator.
- The developer.

The run-time environment contains the first three components only. It does not contain the developer facilities.

This guide concentrates on describing the HP ALLBASE/4GL developer facilities. To help you place this description in context, the following paragraphs provide an outline description of the other components of HP ALLBASE/4GL.

The Operating System Interface

HP ALLBASE/4GL operates on Hewlett-Packard HP 3000 series 900 computer systems using the MPE XL operating system.

The HP ALLBASE/4GL operating system interface handles all the communication with the host operating system. For the most part, you will never need to use MPE XL commands while developing applications. However, if the need arises you can use the MPE XL system to run external programs in a language other than HP ALLBASE/4GL. You can call such external programs from within HP ALLBASE/4GL.

Data Manager

The data manager handles all access to application databases. The data manager allows multiple users and multiple applications to access the same database simultaneously.

The data manager allows programmatic access to HP ALLBASE/SQL databases. Programmatic access to HP ALLBASE/SQL is similar to the access provided by embedded HP ALLBASE/SQL preprocessors for third generation language systems.

HP ALLBASE/4GL also has an inbuilt KSAM (Keyed Sequential Access Method) data file manager that provides the facilities to create and delete KSAM data files.

The Administrator

The administrator is an HP ALLBASE/4GL application. The system administrator (the person) uses the administrator application to define system user names, applications and versions, and various system wide defaults.

The Developer

The HP ALLBASE/4GL developer is also an HP ALLBASE/4GL application. This is the application that you use to create end user applications. As you work through the lessons in this guide, you will be using the developer facilities to develop some working applications.

The HP ALLBASE/4GL Administrator Environment

The administrator application is an integral part of all HP ALLBASE/4GL systems. It allows the administrator (the person) to control the overall operation and configuration of the HP ALLBASE/4GL system.

If you are the administrator for your HP ALLBASE/4GL system as well as a developer user, you may need to establish a suitable HP ALLBASE/4GL system configuration for your site before you can complete this training course. Refer to the *HP ALLBASE/4GL Developer Administration Manual* for more information about using the administrator application.

System Administration Responsibilities

The system administrator's responsibilities come under the following broad headings:

- System specifications.
- System security.

System Specifications

HP ALLBASE/4GL allows centralized definition of a number of system-wide items such as formats for dates and decimal numbers, and the currency symbol. Centralizing these definitions ensures consistency across all applications on the system.

System Security

The system administrator controls all aspects of system security. This includes:

- Assigning names to developer users and end users.
- Assigning user passwords.
- Defining application names, version names, and passwords.
- Defining development security codes.
- Securing (limiting access to) items on application menus.

The HP ALLBASE/4GL system security facilities are additional to the MPE XL security facilities.

End-User Environment

End users can run HP ALLBASE/4GL applications in either the developer system or the run-time environment.

In either environment, an end user cannot change the application in any way. For end users, applications operate in exactly the same way in both the development and run-time environments.

The source code for an application does not need to be present to allow an end user to use the application in the developer environment, or the run-time environment.

Run-Time Environment

The run-time environment is a subset of the full developer system. It contains all the facilities needed to run developed applications and administer an end user site.

It does not contain the developer application, and it does not contain some of the programs that the developer application uses.

Applications developed with the developer system can be transported to any run-time system. The HP ALLBASE/4GL developer administrator contains facilities for unloading and loading applications. The HP ALLBASE/4GL run-time administrator contains facilities for loading applications.

How HP ALLBASE/4GL Developer Works

The HP ALLBASE/4GL developer provides you with application development facilities that can be grouped under the following headings:

- Dictionary.
- Screen development.
- Logic.
- Reports.

The developer application also includes a number of utilities for copying or deleting items within applications, and for printing documentation about application items.

The developer application is primarily menu and screen driven. You specify most application items by filling in formatted screens to define the details of the item.

As you develop an application, your definitions are stored in a set of internal files. The application definitions exist in two forms: application “source” and the “generated” application. Conceptually these are similar to source code and object code in a conventional programming system.

The generated form of the application is an executable form of the application. HP ALLBASE/4GL uses the instructions in the generated application to present the application items to the user. The application source code does not need to be present at run-time, although its presence makes no difference.

The developer environment contains all the logic necessary to convert your application source code to the generated form. You don’t need any external compilers or utilities.

Both the developer system and the run-time environment contain all the necessary facilities to run generated applications.

Working with HP ALLBASE/4GL

Typically, developing an application with HP ALLBASE/4GL involves five phases:

- Phase 1 – Application Planning.
- Phase 2 – Data definition.
- Phase 3 – Screen development.
- Phase 4 – Logic definition.
- Phase 5 – Report definition.

HP ALLBASE/4GL does not place many restrictions on the way you can develop applications, apart from the need to define dictionary items before they can be used in other parts of the application.

The phases of application development shown here, and used in this guide represent only one of many possible approaches to application development with HP ALLBASE/4GL.

The following descriptions introduce the tasks involved in each phase of application development, and describe the facilities that you will use to carry out these tasks.

Data Definition

The first task in application development is application planning. When you have identified the data storage requirements of the application, you can use the HP ALLBASE/4GL dictionary facilities to define these items and create the application data files.

The dictionary is a central storage facility for the definitions of many application items. You can refer to a item defined in the dictionary from other parts of HP ALLBASE/4GL.

The dictionary contains the following types of definitions:

- Field specifications.
- Validation items.
- Storage items.
- Record layouts.
- Database items.

- Messages.
- Help screens.

During the process of developing an application, you will use most of these facilities. In the early part of developing an application you will need to define some field specifications, record layouts, and database items.

Field Specifications

A dictionary field specification defines the attributes, such as name, length, and type, of a “field” used throughout the application. A field is a data item that can exist on a file, a screen, or a report. A field on any file, screen, or report can refer to a field specification in the dictionary.

This is the screen you use to define a field specification. It’s typical of the screens you will use in HP ALLBASE/4GL. You will define most of the items in your application by simply completing a formatted screen, and in many cases you can accept the defaults that HP ALLBASE/4GL provides.

Developer		Field Specifications		field.specs	
Field Spec. Name	<input type="text"/>	Secured	<input type="checkbox"/>	(Y/N)	
Field Length	<input type="text"/>	Repeated	<input type="checkbox"/>	Times	
Minimum Entry Length	<input type="text"/>	Edit Code	(X/A/U/K/N/S/Q/D/T)		
Justification	<input type="text"/>	(L/R/C/N)	Pad Character		
Decimal Places	<input type="text"/>	Blank When Zero		(Y/N)	
Validation: Range	<input type="text"/>	Table	<input type="text"/>		
Help Name	<input type="text"/>				
Description	<input type="text"/>				
Last Modification:	Date	<input type="text"/>	Time	<input type="text"/>	
Records Menu	Ranges	Tables	Database Items	4	32
				System Keys	Commit Data
				Help	Previous Menu

The dictionary field specifications do not contain application data in themselves. A field specification is a template that defines the following characteristics of a field:

- The field specification name.
- The length of the field and the number of times it is repeated.
- The edit code for the field. This is similar to defining a data item's type in a conventional language. It determines the type of data (for example, numeric, alphanumeric, or alphabetic only) the field can hold.
- The justification code for the field. This determines how a field's contents is presented on a screen or report.
- The pad character that fills unused spaces at the beginning or end of the field.
- The names of any validation table or range used to validate data for this field. Tables and ranges are defined in the next section.

Validation Items

The dictionary validation items are validation tables and validation ranges. A validation table is a table of discrete values used to test input data for a field. For example, you could use a validation table to ensure that the user enters a correct value in a field requiring entry of a department code or a state code abbreviation. A validation range specifies the lower and upper limits for a range of acceptable values for a field.

You can associate a validation range or a validation table with a field specification. When you use such a field to define an input field on a screen, HP ALLBASE/4GL automatically displays an error message if the user enters invalid data.

Storage Items

The dictionary storage items are somewhat different from the other dictionary entries. In contrast to dictionary field specifications, storage items do hold application data. The dictionary storage items are:

- Variables.
- Calculated items.
- Numeric constants.

- Alphanumeric constants.
- Scratch-pad field names.
- Application titles.
- Work areas.

HP ALLBASE/4GL variables are similar to variables in other programming systems. You can specify the number of characters, the edit code, and number of decimal places for a variable.

Calculated items are variables that are evaluated each time they are called by an application. The value of a calculated item is the result of an arithmetic expression or a logic function that is evaluated when the item is called.

The scratch-pad contains up to 99 scratch-pad fields. Each scratch-pad field is a temporary storage area that you can use for storing data within an application.

Each scratch-pad field takes on the attributes of the data written into it, and the combined scratch-pad only uses the amount of system memory required to hold the data contained within it at any given time.

Record Layouts

The record layout definitions in the dictionary define the names and sequence of fields within file records. The record layout definitions also define which fields are used as keys for reading records from the application data files. The data manager uses the key fields to build the indexes when you create database tables (or KSAM data files.)

Database Items

The dictionary database items include the definitions for HP ALLBASE/SQL tables, and select lists. The dictionary database items also include the definitions of KSAM data files and serial data files.

Messages

HP ALLBASE/4GL allows you to use a number of different types of messages to communicate with application end users. Messages may be simple literal strings, or combinations of items including literals, variables, constants, and screen field data.

All messages have a type code. You can define message codes ranging from information and prompts to queries, errors, warnings, and aborts.

Help Screens

HP ALLBASE/4GL allows you to associate a help screen with each application screen, and with each field on a data screen. The dictionary help screen definition contains the help screen name, and the text of the screen.

You can link any help screen to further help screens if you need to present more information than one screen can contain.

Screen Development

Screens are central items for most applications. HP ALLBASE/4GL allows you to create screens quickly and easily, yet still maintain complete control over the operation of the user interface. And at the same time, your application screens can provide a very high level of interaction with the user.

Screen Types

HP ALLBASE/4GL uses three types of screens:

- Menus.
- Data screens.
- Windows.

The screens and menus you will use in the developer application are in fact HP ALLBASE/4GL screens themselves. They show the type of user interface that you can include in your own applications.

Menus

Menus contain a number of action items that the user can select. Each item has a specific action assigned to it. HP ALLBASE/4GL executes this action when the user selects the item.

Typical menu actions are branching to another menu, executing a process, or printing a report. After the user selects an item, HP ALLBASE/4GL performs the assigned action when the user presses the **Return** key.

Data Screens

A data screen can be used to display information to the user, to accept input data from the user, and to process data. Data screens can use input and output fields,

literals, and a number of system defined items such as the current date and time. Each field on a data screen can have a logic function associated with it.

Data screens can provide a substantial amount of the data validation, data formatting, and data movement requirements of an application automatically. By giving some thought to the design of an application you can take advantage of the automatic logic associated with HP ALLBASE/4GL screens, and make a substantial reduction in the amount of logic you need to write.

You can also define a scroll area on a data screen. A scroll area can be used to display lists of information to the user. Each line of data scrolled to the screen can consist of a number of items interspersed with literal strings of text.

Windows

Data screens can also use windows to display additional data or provide additional input fields. When you display a window on a data screen, the window overlays part of the data screen and becomes an integral part of the data screen.

Function Keys

All screens can have a set of function keys. For any function key, you can define the text of the key label, and the action performed when the user presses the key.

HP ALLBASE/4GL provides a default set of function keys that satisfy the basic needs of all screens.

Creating HP ALLBASE/4GL Screens

Creating a menu requires two steps. Data screens and windows require one additional step. The steps are:

1. **Define the screen header.** The screen header defines the name of the screen, its type, and some of its characteristics. You can also enter a brief description of the screen when you define the screen header.
2. **Paint the screen image.** In this step, you use the screen painter program to “paint” the image of the screen, exactly as it will appear to the user. While painting the screen, you also define the actions for items on menus, and most of the details for fields on data screens.
3. **Define the screen field details.** This step is only required for data screens and windows. The screen field details specify the automatic data validation and data movement performed for the fields on a screen, and also specify some other characteristics of the fields.

Data screens and windows must be generated to convert the screen definitions into a run-time form.

The Screen Painter

The screen painter is a program that allows you to create the image of an application screen using your terminal facilities directly. When you work with the screen painter, the image you see on your terminal is how the application screen will appear to the user when the application runs.

You can create literals on a screen by simply moving the cursor to where you want the literal to start, and entering the text. Function keys in the screen painter allow you to create fields and system defined items on the screen.

The screen painter also allows you to move, modify, copy, or delete existing items on a screen. You can also use an existing screen as a template to create new screens.

As you paint the image of a screen with the screen painter, you are also defining the actions associated with items on menus, or the details for fields on data screens. You can use your previously defined dictionary field specifications to create fields on data screens. When you do this, HP ALLBASE/4GL automatically builds the field details for the field, based on the dictionary field specification. In most cases, you will be able to use these defaults without making any changes.

Prototyping with the Screen Painter

The screen painter is an effective prototyping tool. The screen painter allows you to work with your end users, and define the requirements of an application by creating screens on the spot.

You can show the user exactly how the screen will appear and operate when the application runs. Your user can suggest changes or additions to the screen layout while you are creating it. Adding new fields or changing existing fields is simple, and you can do it immediately. By working with the user, you can be sure that the application does what the user requires. And, when you create a prototype screen image, you are actually creating most, if not all, of the logic required to implement the screen.

Logic Definition

The next step in developing an application is to define the logic required for the application.

HP ALLBASE/4GL supports four types of logic structures: processes, functions, SQL logic blocks, and decision tables.

Processes

A process is conceptually similar to a program in a conventional language.

When a process starts, HP ALLBASE/4GL initializes the entire application environment to a known state.

When a process starts, the following occurs:

- All application data files are closed.
- All file record and screen buffers are cleared of data.
- Any current non-background activity in the application is terminated. This means that any unfinished process, screen, function, decision table, or report is stopped immediately.
- Any incomplete HP ALLBASE/SQL transactions are reversed.

Typically, applications use a menu to allow the user to select an option. The action item on the menu then calls a process that performs any appropriate initialization required for the selected activity. For example, the process can declare that one or more data files are to be opened in write mode to allow updating. The process can then call a data screen to allow the user to enter data, or examine information from a file. After the user terminates the screen, the process can perform any necessary file updates.

Processes consist of a number of instructions that make up the logic block of the process. Each logic block can contain up to 30 lines of instructions where each instruction contains one or more of the HP ALLBASE/4GL logic commands. HP ALLBASE/4GL has a set of about 40 logic commands.

At the end of each process, HP ALLBASE/4GL control returns to the last menu displayed.

Functions

A function is conceptually similar to a subroutine in a conventional language.

When a function is completed, control returns to the item that called the function. Functions may be nested. That is, one function may initiate another function.

A common use of functions in applications is screen field functions. You can associate a function with each input and display field on a data screen. A screen field function can perform additional data validation or data formatting, or can perform actions such as retrieving data from a file. You will create a number of screen field functions during this training course.

Functions have the same format and use the same commands as processes.

SQL Logic Blocks

SQL logic blocks are the means of communicating with HP ALLBASE/SQL. An SQL logic block can contain up to eight SQL commands. At run-time, the commands in an SQL logic block are invoked by a command in an HP ALLBASE/4GL logic block.

Decision Tables

A decision table is a tool for performing one or more conditional tests, and then implementing one or more actions as a result of the outcomes of the conditional tests. Decision tables may be used to implement more complex conditional testing than is available using the IF and IFLOOP logic commands.

A decision table can test up to eight questions or conditions, and initiate up to eight actions. Each decision table can test up to 31 different combinations of outcomes for the eight decision table questions, and for each combination of outcomes, execute some or all of the eight decision table actions in a specified order.

The decision table actions can include processes, functions, screens, and reports.

Report Definition

The next stage in developing an application is to define the reports.

HP ALLBASE/4GL allows you to define a report using a number of formatted screens, and a report painter.

The report generator reports on the contents of one file, known as the primary file for the report. You can specify selection criteria to limit the records selected

from the file, and you can specify the order of sorting before the report is printed. The report generator allows you to define linkages to other files to read further information for reporting. The report generator also allows you to define the report layout, totalling, and any special logic requirements for the report.

Line Groups

HP ALLBASE/4GL reports consist of a number of different line groups. Line groups are available for page headings, bottom of page lines, column headings, subheadings, total lines, and the detail lines that are printed for each file record selected for reporting.

Record Sorting and Selection

The report generator allows you to define up to eight fields on the primary file to be used as sorting fields for the report. Records can be sorted in ascending or descending order according to the values in the sort fields. For example, if you had a file containing data about customers, you might assign the first sort level to the state code and the second level to the zip code. The report would first group the customers by state, and then by zip code within each state.

You can specify up to 99 sets of selection criteria for selecting records to report. Each set of criteria defines the starting and ending values by which a field is selected. You can use a logic function to display a screen at the start of the report to allow the user to specify values for selection criteria. You can also use logic functions to perform additional or alternative record selection for a report.

Control Breaks

A control break occurs whenever a value in a sort field changes. HP ALLBASE/4GL allows you to specify that subheadings and subtotals are printed on a report when a control break occurs. An example of this is to define that customer account balances are to be subtotaled for each zip code area and then totalled for each state, and that new subheadings are to be printed after the totals.

Report Linkages

You can access files other than the primary file by using report linkages. Any report line can be linked to one or more files which will then be read immediately before printing the line that initiated the link. Data that is read from a link file can be used and printed in the report.

The Report Painter

The report painter is similar to the screen painter. It allows you to use your terminal screen as a “window” to the report, and use the normal terminal control keys to create report line fields and literals exactly as they will appear on the printed report.

The report painter also allows you to move, copy, or delete existing fields or literals on a report, or use an existing report as a template for a new one.

Since most reports will be larger than the report painter “window”, you can scroll the report image horizontally and vertically using report painter function keys.

From Here On

This guide gradually takes you through the various HP ALLBASE/4GL facilities allowing you to become more competent and confident in their use.

Before you start to create an application you will see how to use your terminal keyboard and screen to communicate with HP ALLBASE/4GL. The remaining chapters introduce and expand on the various facilities you can use in HP ALLBASE/4GL. Through some practical examples, you will see how to use HP ALLBASE/4GL to perform standard programming tasks.

As you work through this guide, you will begin to see how HP ALLBASE/4GL can be used to meet your particular requirements.

How This Guide Works

This guide describes the procedures to develop two HP ALLBASE/4GL applications.

The first application introduces you to all the major components of the developer, and demonstrates how to use them. This application is described in chapters 4 to 10. To avoid the need to define an HP ALLBASE/SQL database, this application uses the HP ALLBASE/4GL KSAM data file interface. For the most part, the techniques used to develop an application do not depend on the data manager.

The second application concentrates on the HP ALLBASE/4GL/HP ALLBASE/SQL interface. This application shows you how you can use the HP ALLBASE/SQL interface to build an application that is functionally equivalent to the application described in the earlier chapters. This application is described in chapters 11 and 12. These chapters assume that you have completed the application described in the earlier part of the guide, and you are familiar with the basic techniques for using HP ALLBASE/4GL. They also assume that you are familiar with using

HP ALLBASE/SQL. Don't attempt to work through these chapters unless you have completed chapters 4 to 10.

Most subsequent chapters in this guide are broken down into a number of individual lessons. Each lesson starts with a statement of objectives, and concludes with a summary of the points covered in the lesson.

Terminology Used in This Manual

The following list explains the terms used in this guide:

- **Enter.** Type in the characters following and press **Return**.

Note

HP ALLBASE/4GL does **not** use the **Enter** key on HP terminal keyboards. Pressing the **Enter** key may cause your terminal to behave erratically.

- **f1** to **f8**. These are the eight function keys across the top of your keyboard. Their identifying labels are displayed across the bottom of the screen.
- **The developer application.** This is the application that you use to develop HP ALLBASE/4GL applications.
- **The administrator application.** This application is used by the system administrator to define the HP ALLBASE/4GL environment. Items such as security, user names, system specifications, and other parameters are defined in the HP ALLBASE/4GL administrator application.
- **Menu path.** You will see this heading every time a new developer screen is introduced. It indicates the menu selections you must make to arrive at a particular screen from the main menu.

Intentionally Blank

Contents

Chapter 2: Getting Started

Before You Start	2 - 1
The Sign-On Screen	2 - 1
User Name	2 - 2
Using Your Terminal and Keyboard	2 - 3
Menus	2 - 4
Data Screens	2 - 5
Field Commit	2 - 6
Commit Data	2 - 6
Function Keys	2 - 7
Help	2 - 7
Field Help	2 - 8
System Keys Function Key Set	2 - 8
More Keys Function Key Set	2 - 9
Signing Off	2 - 9



Intentionally Blank

2 Getting Started

This chapter describes the HP ALLBASE/4GL sign-on procedure, and explains the use of HP terminals and keyboards with HP ALLBASE/4GL.

Before You Start

The process you use to start HP ALLBASE/4GL may vary from one installation to another. You may need to talk to your system administrator about the procedure to log in to the MPE XL system, and to start HP ALLBASE/4GL. Your HP ALLBASE/4GL administrator will allocate an HP ALLBASE/4GL user name for you so you can sign on as a developer.

Before you continue any further, make sure that the HP ALLBASE/4GL administrator has followed the instructions in Appendix A. This enables you to develop and run the training applications described in this manual.

If you are the system administrator, refer to the *HP ALLBASE/4GL Installation Guide* for instructions to install HP ALLBASE/4GL. The *HP ALLBASE/4GL Developer Administration Manual* describes the procedures for defining user names and application names in HP ALLBASE/4GL.

The Sign-On Screen

In the standard HP ALLBASE/4GL installation, the sign-on screen is the first screen that every user sees. It enables you to enter your HP ALLBASE/4GL user name and password.

HP ALLBASE/4GL B.00.00 HP30601A Copyright(c) 1986-88 Hewlett-Packard Aust. Ltd.

HP ALLBASE/
4GL

User Name User Password

17 18

User Name

Enter your HP ALLBASE/4GL user name in the first field. HP ALLBASE/4GL is case sensitive, so make sure you type the correct name. If you type an incorrect letter you can use **Back space**, **←** or **→** to move the cursor, and then retype any incorrect characters. When you have entered your user name, press **Return** or **Tab** to indicate that the user name field is complete.

After you enter your user name, you are prompted to enter your password if one is required. Your password won't appear on the screen as you type it in, although the cursor moves to show how many characters you have entered.

As a developer, you are prompted for the name of the application (or version) that you want to work on.

```

HP ALLBASE/4GL B.00.00 HP30601A Copyright(c) 1986-88 Hewlett-Packard Aust. Ltd.

HP ALLBASE/
  4GL

-----

User Name      [ ]      User Password [ ]
Application or Version to be developed [ ]
Development Security Code [ ]

-----

[ ] [ ] [ ] [ Start Again ] 19 50 [ ] [ Commit Data ] [ Help ] [ Exit ]
  
```

Note

This is the sign-on procedure for a developer. End users receive different messages.

Enter the name of the application you wish to develop. For the applications described in this manual, enter the application name **training** followed by **(Return)**. There is no development security code required unless your system administrator indicates otherwise. If you want to assign a development security code, ask your system administrator to help you.

To begin developing the application, press the **Commit Data** function key. This confirms that you have entered all of the details for this screen and you want to continue. HP ALLBASE/4GL then displays the developer main menu.

Using Your Terminal and Keyboard

HP ALLBASE/4GL supports a number of major terminal and keyboard features such as function keys, special editing keys and touch-screens. This section explains and demonstrates the use of some of these features.

Menus

When you first access a menu, the first item on the menu is highlighted. Press **Return** or the **Activate Item** function key to execute the highlighted item.

You can use the **Tab** key to select other items on the menu. Each time you press the **Tab** key, the highlight moves to the next menu item.

You can also use the cursor keys **←**, **→**, **▲** and **▼** to select menu items. To use these keys, move the cursor to the item you want to select and press **Return**. HP ALLBASE/4GL responds by highlighting the item at the current cursor position. Press **Return** to execute the item.

On any menu, pressing **▼** followed by **Return** takes you to the first item on the menu. Pressing **Shift** and **▼** followed by **Return** takes you to the last item on the menu.

If you have an HP touch-screen terminal, you can select a menu item by touching it on the screen. As you touch the screen, the selection highlight moves to the item closest to the point you touch.

There is another way for you to make a selection of an item on a menu. You can select an item by entering the first character of the item's name. Most menus in the developer have a unique first character for each menu item. If you type the first letter of an item on the menu and then press **Return** the highlight moves to that item. If a menu contains two items that start with the same letter, pressing the initial letter again and then pressing **Return** again selects the second item starting with the letter you enter. If you type a letter that does not match a valid menu item, the highlight remains where it is.

Once you have selected a menu item, press the **Activate Item** function key, or the **Return** key to activate the item. All of the selections on the main menu lead you to a lower level menu. Try activating one of the selections on the current menu to go to a lower level menu so that you can get a feel for this process.

Now press the **Previous Menu** function key to return to the main menu. This key appears on all screens except the main menu. Pressing it terminates the current screen and returns you to the menu that you used to reach the current screen.

HP ALLBASE/4GL also has a menu bypass feature. To try this feature, select a menu at least one level down from the main menu. You can jump from one menu to any other menu by pressing the **]** key followed by **Return**. Now type **D-** followed by the name of the screen you want to jump to. Notice that you must enter an uppercase **D**. For this exercise press the **]** key, press **Return**, type in **D-main**, and press the **Return** key again. The main menu will be displayed again.

At the top right corner of the screen you will see the word **main**. This is the name of the main menu. The name of each developer menu appears in the same position on every screen. As you become more familiar with the system you will get to know these names. The menu bypass facility then becomes very useful.

Note

In the developer application, the menu bypass facility allows you to jump to screens other than menus although there are some limitations on the screens that you can access this way. The documentation printing screen and the copying screen in the developer utilities menu, and some screens in the database items menu cannot be accessed directly using the menu bypass method. If you want to know more about this facility, refer to the *HP ALLBASE/4GL Developer Reference Manual*.

These menu selection methods are also available in any application you develop. By naming your menu items and menus carefully, you can make your application easier to use.

Data Screens

Data screens in the developer are the screens that allow you to enter data to define the details of application components. In data screens, you can use a number of the terminal keys while entering data in fields or moving about the screen.

You can use any of the **Clear line**, **Insert char**, **Delete char**, **Back space**, **▶**, or **◀** keys to edit the data entered or to move about within a data input field on a data screen.

A number of keys allow you to move from one field to another on a data screen. The **Return** key terminates the current field and moves the cursor to the next field in the screen tabbing sequence. The **Tab** key behaves the same way. Pressing the **Shift** and **Tab** keys together moves the cursor to the start of the current field. If the cursor is at the start of a field, pressing the **Shift** and **Tab** keys together, and then pressing **Return** moves the cursor to the previous field in the screen tabbing sequence.

You can use the **▲**, **▼**, **◀** and **▶** keys to move the cursor to a field on the screen. Press **Return** when you have moved the cursor to the field. *HP ALLBASE/4GL* then highlights the field and you can enter data into the highlighted field.

To move the cursor to the first field on the screen, press **⏮** and then press **Return**. To move the cursor to the last field on the screen, press **Shift** and **⏮** together and then press **Return**.

Moving the cursor from one field to another field and then pressing **Return** initiates a *field level commit* action for the first field.

Field Commit

A field commit action occurs whenever you press **Return** or **Tab** to move the cursor from one field to another. The field commit action is a very significant concept in HP ALLBASE/4GL.

Each field on a data screen may have some logic associated with it. This logic is automatically executed when the field is committed. You will learn about using screen field logic processing as you develop some of the screens in this manual.

The logic associated with a field can restrict the movement you can perform after you complete a field. Some developer screens are structured in such a way that entering a certain value on one field automatically moves the cursor to another field, regardless of the key you used to terminate the field. You may occasionally see the result of this sort of logic when you use a cursor directional key on a data screen, and the cursor does not go where you thought it would. Instead, it moves to the next field that you must complete on the screen.

When you have entered all of the necessary data on a screen, you must press the **Commit Data** function key to actually complete the screen. If you have entered data for the very last field on a screen you can press the **Commit Data** function key to perform both a *field commit* for the field as well as a *screen commit* for the screen.

Some fields on screens are required fields. That is, they cannot be left blank. You cannot initiate a field commit for one of these fields if it is blank. However, you can move to a previous field on the screen by pressing the **Shift** + **Tab** followed by **Return**.

Commit Data

The **Commit Data** action is one of the most important functions in HP ALLBASE/4GL. It is central to the processing logic of all screen-oriented procedures in HP ALLBASE/4GL applications, so a good understanding of the concept is important.

Pressing the **Commit Data** function key indicates that all of the data that you have entered in the screen is correct and you want to terminate the screen to continue

to the next step. The **Commit Data** function key terminates the screen that is currently active, and returns control to whatever initiated the screen. In many cases this will be a process which, upon return from the screen, performs some type of file update to reflect the data you have just entered.

In some instances the **Commit Data** function key can also initiate a field level commit and then perform the logic associated with the field before terminating the screen. If the current field (that is, the field currently occupied by the cursor) has not been committed when you press the **Commit Data** function key, a field level commit is performed before the commit data action is performed for the screen.

A commit data action will not succeed if you have left any required fields blank.

You can use the **Previous Menu** function key to cancel the screen if you don't want to terminate the current screen normally. You will be asked to press the **Previous Menu** key a second time to confirm the loss of data entered so far.

Function Keys

So far you have been using some of the standard function keys available to you as a developer. HP ALLBASE/4GL uses function keys extensively throughout the entire system. Four standard keys are available on all screens. These keys are:

- F5** System Keys
- F6** Commit Data or Activate Item
- F7** Help
- F8** Previous Menu

The **Help** and **System Keys** function keys are described below.

Help

You can use the **Help** function key on all screens to obtain help about the current field or screen. Help in HP ALLBASE/4GL is always in the same format. The current screen is cleared and a help screen is displayed on the screen. When you press the **Exit Help** function key, the help screen is cleared and the original screen is redisplayed with the cursor position unchanged.

Some help screens occupy more than one “page”. In this case you can press the **More Help** and **Previous Help** function keys to move through the different screens of help that are available.

Field Help

When entering data on a data screen you can press the **Help** function key to see the help details for the current screen field. This screen is known as the *field help* screen. When you call up a field help screen there are some further options available to you.

You can press the **Field Desc.** function key to display the field specification details. Pressing the **Display Range** function key displays the acceptable range for the field if a range has been specified. Similarly, pressing the **Display Table** function key displays the table contents if they have been defined for the field.

If you want to see general help about a data screen, press the **Screen Help** function key.

System Keys Function Key Set

By pressing the **System Keys** function key on any screen, you can display two sets of function keys. Pressing the **System Keys** function key on a developer screen loads the following set of function keys:

- (f1) **Main Menu** Returns you to the developer main menu.
- (f2) **Screen Printing** Loads the screen printing function key set. These keys allow you to print a screen image on the system printer or on a slave printer attached to an HP terminal.
- (f3) **Clear Screen** Clears all fields on the current screen and repositions the cursor at the first input field on the screen.
- (f4) **Refresh Screen** Repaints the current screen image, including the current contents of all fields.
- (f5) **Previous Keys** Reloads the standard set of function keys for the screen.
- (f6) **More Keys** Loads the second set of system keys. Refer to More Keys Function Key Set for more details.
- (f7) **Help** Displays the help screens associated with the current screen or field.
- (f8) **Previous Menu** Returns you to the last menu displayed by HP ALLBASE/4GL.

More Keys Function Key Set

Pressing the **More Keys** function key from the system keys function key set loads the following function key set:

- (f2) **Op. System** Temporarily suspends execution of HP ALLBASE/4GL and runs the MPE XL command interpreter. When you exit from the command interpreter, (by typing **exit** at the MPE XL prompt) processing resumes at the point where you suspended it.
- (f3) **ISQL** Pressing this key temporarily suspends execution of HP ALLBASE/4GL and runs ISQL.
- (f4) **ALLBASE QUERY** If HP ALLBASE/QUERY is installed on the system, pressing this key temporarily suspends execution of HP ALLBASE/4GL and starts HP ALLBASE/QUERY. HP ALLBASE/QUERY automatically connects to the current application's database. When you exit from HP ALLBASE/QUERY, you are returned to HP ALLBASE/4GL.

If the current application has an open SQL transaction when you press this function key, HP ALLBASE/4GL displays a warning message, allowing you to complete or reverse the transaction before starting HP ALLBASE/QUERY.

- (f5) **Previous Keys** Reloads the previous function key set.
- (f7) **Help** Displays the help screens associated with the current screen or field.
- (f8) **Exit** Terminates the current HP ALLBASE/4GL session and returns you to the MPE XL system.

Signing Off

At the end of your HP ALLBASE/4GL session, return to the developer main menu. Press the **Sign On Screen** function key to return to the sign-on screen.

To exit completely from HP ALLBASE/4GL, you can now press the **Exit** function key. This returns you to the MPE XL system.

You can also exit by pressing the **Exit** function key in the *More Keys* set of system function keys.

Intentionally Blank

Contents

Chapter 3: The Application

Understanding the Task	3 - 1
Application Specification	3 - 1
Application Structure	3 - 3
Structure Diagram	3 - 5



Intentionally Blank

3 The Application

This chapter introduces the application you will develop during this self-paced training course. The name of the application is *training*.

The application performs two functions. It allows the creation and maintenance of a simple customer database, the creation and maintenance of a slightly more complex product/option database, and the creation and maintenance of a simple supplier database.

The application is not a definitive example of system design. Instead, it is designed to demonstrate the use of the major facilities of HP ALLBASE/4GL without becoming too complex.

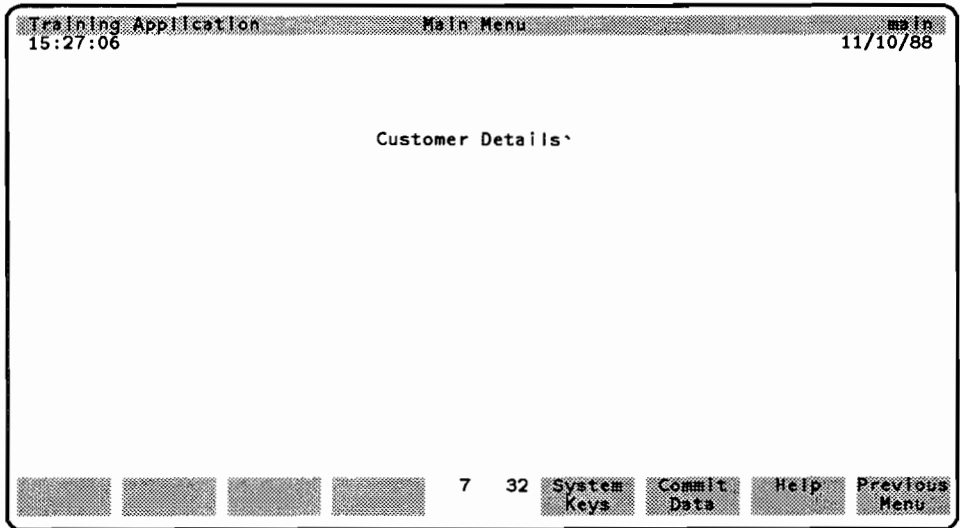
During this training course, you will initially develop a very simple customer file maintenance system. You will then enhance this system to improve the flexibility and usability of the system. In a later part of the course, you will develop the product/option file maintenance system. This system introduces you to a number of more advanced HP ALLBASE/4GL features. Finally you will use the module builder feature of HP ALLBASE/4GL to create the supplier file maintenance system.

Understanding the Task

You are being asked to develop the application after the preliminary design work has been completed. The design work has identified the essential input, output and processing requirements, and you are now ready to begin developing the application.

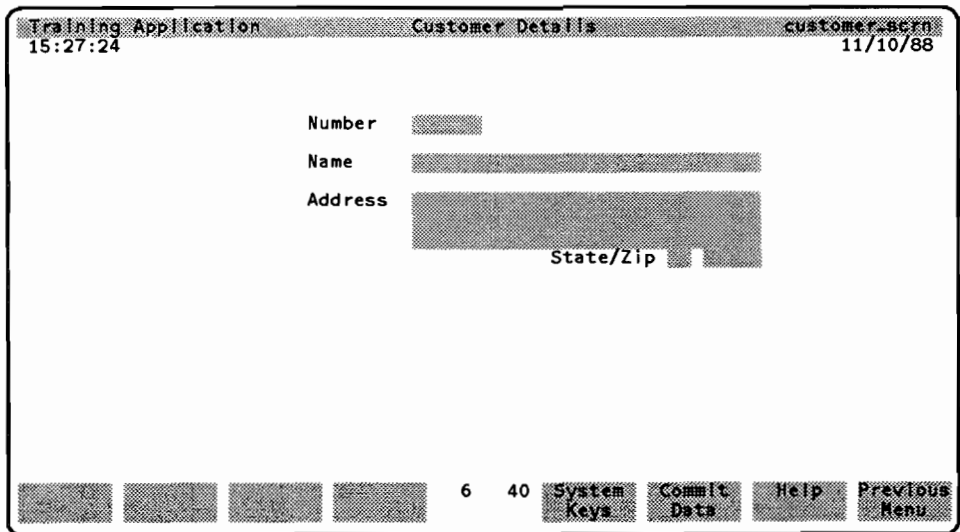
Application Specification

In its simplest form, the customer file maintenance portion of the application consists of one menu, and one data screen. When the user signs on to the application, HP ALLBASE/4GL presents the main menu of the application. In this initial form, this menu only has one option that allows the user to access the data screen for the application.



You will add some further options to the main menu as you proceed with developing the application in later lessons.

When the user selects the *Customer Details* option, HP ALLBASE/4GL displays the *customer.scrn* data screen.



This screen allows the user to add a new customer record, or modify existing customer records.

This screen uses the following fields:

Field	Description
Number	A six digit identifier that uniquely identifies each customer.
Name	The name of the customer. This field is 30 characters long.
Address	A set of three lines, each of which can contain up to 30 characters.
State Code	A two character field that contains the US Postal Service abbreviations for the state.
Zip Code	A 5 character field that contains the US Postal Service ZIP code.

To add a new record, the user must enter the number for the customer and complete all the fields on the screen. When the user presses the **Commit Data** function key, HP ALLBASE/4GL writes the new record to the customer file.

To modify an existing record, the user must enter the number of an existing customer in the *Number* field. The application retrieves the existing record from the file and displays the information on the screen. The user can then modify the record by typing over the existing details to enter the new data. When the user presses the **Commit Data** function key, the application writes the changed data back to the file.

Application Structure

Internally, the application operates as follows.

When the user signs on to the application, HP ALLBASE/4GL automatically displays the main menu for the application. (The initial action in an application can be either a menu or a process. In this application, the initial action is the main menu. The system administrator defines the name and the type of the initial action.)

When the user selects *Customer Details* on the main menu, HP ALLBASE/4GL executes a process called *customer_proc*. This process displays the *customer_scrn* screen.

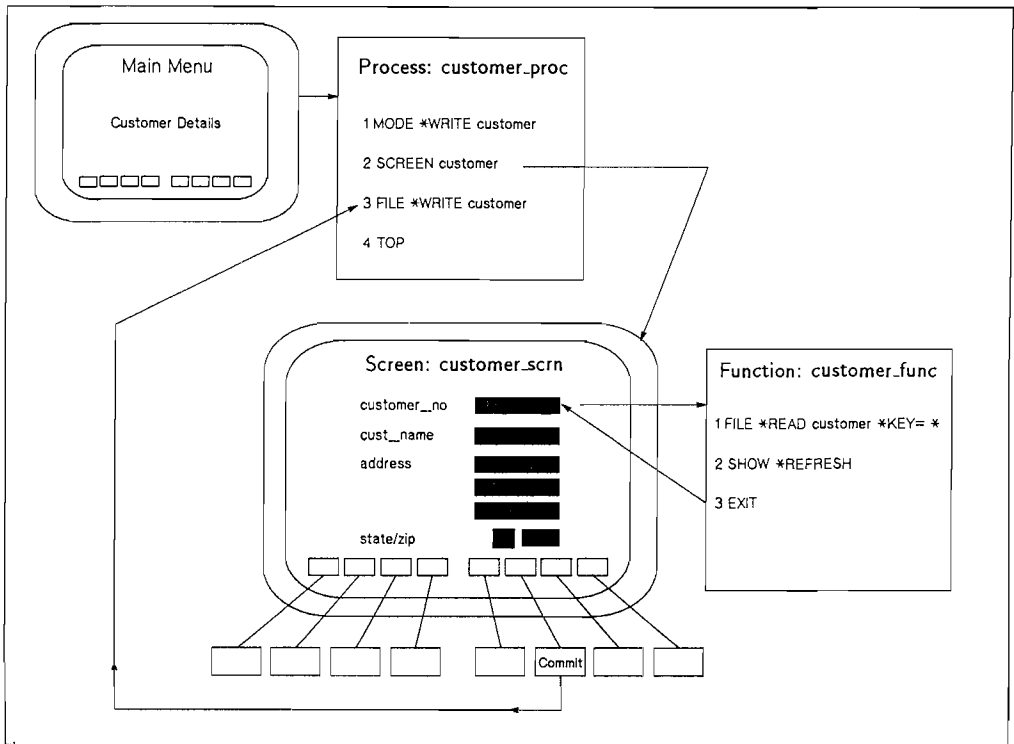
The screen, *customer_scrn*, has a function associated with the first field. After the user has completed data entry in the customer number field, this function reads the customer file for a record matching the customer number just entered.

If a matching record exists, the function displays the details of the record on the screen. The user can now modify the details of the record. If a matching record does not exist, the user can enter the details for a new record.

When the user presses the **Commit Data** function key, control returns to the process. The process updates the customer file to reflect the new or changed data entered by the user, and then displays the screen again.

To exit from the process, the user must press the **Previous Menu** function key.

The structure of the application can be represented as shown in the following diagram.



Application Structure

This structure demonstrates a number of important features of HP ALLBASE/4GL application design.

First, it demonstrates that you can consider an application as being a series of linked elements. In this case, the sequence of elements is a process, a screen, and

then a function. All the elements in a sequence such as this are identified by a name.

Most importantly, it demonstrates one standard format for constructing applications. This is the technique of using a process to call a screen, and, on exit from the screen, returning to the process to update one or more files to reflect data entered while the screen is active. It also demonstrates the technique of using a function associated with a screen field to perform a file lookup while the screen is active. This structure emphasizes the similarity between a process and a program in a conventional language, and the similarity between a function and a conventional language subroutine.

A process initializes the environment to a known state when it starts. Using a process to control the screen and subsequent file update means that the screen display and file update occur within a known and controlled environment. An HP ALLBASE/4GL application only allows one process to operate at a time. In this case, the process remains active and controls the application until the user terminates it by pressing the **Previous Menu** function key.

The function associated with the first field on the screen behaves somewhat like a subroutine. It performs a file lookup, and the application then continues from the point where the function was called after the file lookup is complete. The function does not change the overall environment for the application.

This application also demonstrates the logic and data processing capabilities associated with screens. As you work through this guide you will see how HP ALLBASE/4GL screens operate.

Structure Diagram

The following diagram shows the structure of the application in a simpler form.



This structure diagram shows you where each item in the application is called. You will notice that this diagram does not show any return path for the flow of control from the various items. Since the flow of control is predictable on exit from any item, this information does not need to be shown on this type of diagram.

For all application items, the following rules apply to the flow of control on exit from a item.

Item	Control returns to ...
Process	The last menu displayed by the application. If no menu has been displayed, the application terminates.
Function	The item that calls the function.
Data screen	The next command in the logic block that called the screen, or the previous menu if the screen was called from a menu.

Contents

Chapter 4: Fields, Records and Files



Names	4 - 1
Naming Rules	4 - 1
Referencing Rules	4 - 2
Case Sensitivity	4 - 3
Action Prefixes	4 - 4

Lesson 1 – Fields

Objectives	4 - 5
Identifying the Application's Fields	4 - 5
Creating Field Specifications	4 - 5
Validating Data	4 - 9
User Help Facility	4 - 9
Documentation	4 - 10
Summary	4 - 12

Lesson 2 – Records

Objectives	4 - 14
Record Layouts	4 - 14
Record Layout Header	4 - 14
Record Layout Details	4 - 16
Generating a Record	4 - 19
Summary	4 - 20

Lesson 3 – Files

Objectives	4 - 21
Data Files	4 - 21
Defining a Data File	4 - 21
Creating the Data File	4 - 24
Summary	4 - 26

Intentionally Blank

4 Fields, Records and Files

Typically, the first step in creating an HP ALLBASE/4GL application is to define the fields it uses. This enables you to access these field definitions when creating other HP ALLBASE/4GL items such as: records, screens, and reports.

This chapter contains three lessons that introduce you to fields, records, and files. As you work through the lessons in this chapter you will define and create some fields, records, and files for the *training* application.

This chapter also introduces the rules for naming and referencing items in applications.

Names

HP ALLBASE/4GL uses names to identify the items in an application.

Each item in an application must have a name. You can use the same name for a number of different items, such as a process or function, but a name must be unique for any particular type of item. For example, this training application could use the name *customer* for the file, screen, process, and function (this has not been done in the interest of clarity). However, if an item of a particular type already exists, (for example, a file called *orders*), you cannot create another of the same name and type.

HP ALLBASE/4GL differentiates between the items by using the name in context. This means that the FILE logic command always refers to a file, the PROCEED command always refers to a process and so on. Where an ambiguity may exist HP ALLBASE/4GL forces you to specify a unique type prefix for the item.

The following paragraphs describe the main naming and referencing rules. If you need a complete description of these rules at any time, refer to the *HP ALLBASE/4GL Developer Reference Manual*.

Naming Rules

Any name that you create in an application must conform to the following rules.

- All names must start with an alphabetic character. Names can contain any combination of alphabetic characters, 0 to 9, and - (underscore),

including extended (eight-bit) characters. Chapter 11 introduces some additional rules for naming SQL components in applications.

- The names that you use must be unique for each type of item that you are creating. However, you can use one name for a number of different types of items. For example, you can use the same name for a report, a screen, a field and a process.
- You can use up to eight characters for:
 - file names or SQL tables names, and
 - SQL select list names.
- You can use up to 16 characters to name any of the following items:
 - alphanumeric or numeric constants,
 - application titles,
 - calculated items,
 - decision tables,
 - field specifications,
 - functions,
 - function key sets,
 - help screens,
 - messages,
 - processes,
 - record layouts,
 - reports,
 - scratch-pad fields,
 - screens,
 - validation ranges,
 - validation tables,
 - variables, and
 - work areas.

Referencing Rules

In an application you refer to an item of the application by using a prefix and a hyphen followed by the item name.

For example, an application may contain a numeric constant and a variable that both have the name *discount*. You would reference the numeric constant as *N-discount*, and you would refer to the variable as *V-discount*. The following list shows the item referencing prefixes:

Prefix	Description
A-	application title
B-	work area buffer
C-	alphanumeric constant
F-	file record field
M-	master title
N-	numeric constant
P-	scratch-pad fields (referenced by name)
R-	file record buffer
S-	screen field
U-	calculated item
V-	variable item
W-	work area field
*	the asterisk * prefixes special system item names including: <ul style="list-style-type: none">— communication area fields,— switches,— scratch-pad fields (referenced by number),— screen fields, and— command arguments.

Case Sensitivity

HP ALLBASE/4GL names are case sensitive. You must always type the name of an item the same way each time that you use it. The variable *V-item-cost* is not the same as the variable *V-ITEM-COST*.

Action Prefixes

HP ALLBASE/4GL also uses a set of prefixes to identify the action types that can be called from menus and function keys. The following table lists the action prefixes used in HP ALLBASE/4GL.

Code	Action Type
B-	Background process
D-	Data screen or menu
F-	Function logic block
H-	Help screen
I-	Internal action
P-	Process logic block
R-	Report
X-	External program

Some of these action prefixes are identical to normal prefixes. For instance, the prefix P- indicates scratch-pad fields. The action prefix P- indicates a process. HP ALLBASE/4GL differentiates between the two types of prefix according to their usage. Action prefixes are used in menu painting, the menu bypass facility and in trace mode. Normal prefixes have no meanings in these situations.

Lesson 1 – Fields

Objectives

In this lesson you will use the dictionary field specifications screen to define a number of field specifications.

When you have completed this lesson:

- You will be familiar with the main characteristics of HP ALLBASE/4GL fields.
- You will know the meaning of the terms *edit code* and *justification code*.
- You will know how to enter a description for an HP ALLBASE/4GL item.

Identifying the Application's Fields

The first task in developing any application is to identify the fields that the application requires.

The analysis performed for this application shows that you need the following data fields:

Data Field	Contents
Customer Number	6 digit identifier
Customer Name	30 character free form name
Address	3 lines of 30 characters each
State Code	2 character state abbreviation
Zip Code	5 digit zip code

Creating Field Specifications

You can now create the specifications for these fields using the dictionary field specifications screen.

Menu Path

You should now be signed on to HP ALLBASE/4GL as a developer of the application *training*. From the main menu, select and activate the item *Dictionary*. The dictionary menu will be displayed. From the dictionary menu, select and activate the item *Field Specifications*.

Through the remainder of this guide this process of reaching the appropriate screen from the main menu will be shown simply as:

Dictionary, **Field Specifications**

Description

Developer		Field Specifications		field-specs	
Field Spec. Name	customer.no	Secured	<input checked="" type="checkbox"/>	(Y/N)	
Field Length	6	Repeated	<input type="checkbox"/>	Times	
Minimum Entry Length	1				
Edit Code	N	(X/A/U/K/N/S/Q/D/T)			
Justification	R	(L/R/C/N)	Pad Character		
Decimal Places	0		Blank When Zero	<input checked="" type="checkbox"/>	(Y/N)
Validation: Range		Table			
Help Name					
Description	Customer Number AUTHOR: rmjones This field forms a unique six digit customer identifier				
Last Modification:	Date		Time		
Records Menu	Ranges	Tables	Database Items	21	66
			System Keys	Commit Data	Help Previous Menu

In this screen you can define a field specification to be used as a template in the records, screens and reports for this application. Defining a field specification does not allocate any storage space for the field. The field specification only defines the essential characteristics of a field used in your application.

Entering the Field Values

You can work through the fields on this screen one at a time and fill in the values required. To complete the entry of data into a screen field, you must press **Return** or **Tab**. HP ALLBASE/4GL then accepts the current contents of the field, and moves the cursor to the next field.

The first field you are going to define is the *customer.no* field. The details for this field are summarized on page 4 – 5.

Field Spec. Name. Enter *customer_no*

This is the name that is used to refer to the field specification within the application.

Secured. Accept the default value of **N** (by pressing **Return**).

The entry in this field determines whether this item is secured against modification by an unauthorized developer. In this particular example, there is no need to secure any items in this application. For further information about source code security, refer to the *HP ALLBASE/4GL Developer Administration Manual*.

Field Length. Enter **6**

This specifies the maximum number of characters that can be entered for this field. In this application you need to allow for a six digit field. This also specifies the space the field occupies when used in a file record, screen, or report.

If you are calculating the length of a numeric item containing a number of decimal places, you must include the single character space taken by the decimal point in the field's length.

Repeated  Times. Accept the default value of **1**

Some fields may be made up of a number of smaller fields, all with the same specifications. To create such a field, which is simply a single dimensional array, specify the number of times this field occurs (or is repeated) within the total field specification. The record you create later in this chapter contains an example of a repeated field. Through using this field you will see how the different facilities handle a repeating field.

Minimum Entry Length. Accept the default value of **1**

This value specifies the minimum number of characters to be entered when the field specification is used on a screen. In this case, the user must enter at least one character.

The user is only required to enter the minimum number of characters if a field on a data screen is defined as a required field. If it is not a required field, the user may leave it blank.

Edit Code. Enter **N**

This entry indicates the type of data that the field can contain. An *N* edit code indicates numeric data is required.

The available edit codes are shown in the following table:

Code	Meaning
X	Any printable character.
A	Alphabetic characters only (A to z, extended characters, or space).
U	Forced uppercase. Same as X edit code, but shifts all lowercase alphabetic characters to uppercase.
K	Alphanumeric characters only (A to z, 0 to 9, extended characters, and underscore).
N	Unsigned number (0 to 9 . or ,).
S	Signed number. Same as N edit code, but may include + or -.
Q	Question. Y, y, N, or n only. Any characters beyond the first character in the field are ignored.
D	Date field. Must be eight characters long. The date may be either MM/DD/YY or DD/MM/YY (depending on the system-wide date format) where MM, DD, and YY represent the month, day, and year respectively.
T	Date field, defaulting to the current date.

For on-line help about edit codes, press the **Help** function key. All input fields in the developer have a help screen that you can access while the cursor is on that field.

Justification. Accept the default value of *R*.

When you committed the previous field, this field defaulted to *R*. This indicates the direction of justification that HP ALLBASE/4GL applies when a value is entered into or displayed in this field. *R* indicates right justified, and is the default entry for a numeric edit code.

Generally, you won't need to worry about entering a justification code in this field. HP ALLBASE/4GL selects the justification code that matches the entry in the edit code field.

Pad Character. Accept the default value (a space).

This entry specifies the single character that is used to fill out the field after justification has been performed. Padding is done only when the field contains less than the maximum number of characters specified for the field. In this case the field is padded with spaces.

Decimal Places. Accept the default value of 0

This indicates the maximum number of decimal places for a numeric item. Entering a space in this field is equivalent to entering 0. In this example, the customer number does not require any decimal places.

Blank When Zero. Enter Y

This specifies that a numeric value, when zero, is displayed as blank rather than zero. In this case an entry of Y is required. This is mainly for screen appearance. You will see that when the application runs and there is no entry in the field, the field is displayed as blank. If *Blank When Zero* is set to N, the field appears as a zero and could detract from the appearance of the screen.

Validating Data

Whenever the user enters a value for a screen field defined with a field specification, HP ALLBASE/4GL tests the entered data against the following criteria:

- Maximum length (*Field Length*).
- Minimum length.
- Edit code.
- Decimal places.
- Range (this is optional—it defines upper and lower limits).
- Table (this is optional—it defines discrete values for validation).

Range. Don't enter anything at this stage; just press .

Table. Don't enter anything at this stage; just press .

The screen processing tutorial in Lesson 6 describes and demonstrates the automatic data validation performed by HP ALLBASE/4GL.

User Help Facility

Help Name. Don't enter anything at this stage; just press .

HP ALLBASE/4GL allows you to define a help screen for every input field on an application data screen. The help screen name you enter here is written into the field definition screen in the screens menu. This is the name of the help screen that HP ALLBASE/4GL displays if the user presses the **Help** function while the cursor is on this field. Later in this training course you will create a help screen for this field.

Documentation

HP ALLBASE/4GL has a number of developer documentation facilities to help you keep track of all of the items in your application. When you print or display the definition of this field specification, the associated description is included with it. This provides you with a description of the field, its use and its meaning within the application.

The *Description* fields on this screen are typical of all the documentation fields in HP ALLBASE/4GL. You will use the description fields on a number of screens as you complete this application.

The description fields consist of a “short” description field, three “long” description fields, and two display-only fields for the date and time.

When the cursor first moves on to the first description field, the display in the field defaults to the name of the item you are defining. You can type over this default to enter a more meaningful description, or press **Return** to retain the default value. For this example, enter **Customer Number** and press **Return**.

The display in the first “long” description field defaults to **AUTHOR:** followed by your HP ALLBASE/4GL sign-on name. Once again, you can type over this default, or extend the entry to provide a more meaningful description.

For this example, press **Return** once to accept the default entry in the first long description field, and enter an appropriate description in the second and third long description fields. For example:

This field forms a unique six digit

For all description fields, HP ALLBASE/4GL automatically updates the *Last Modification* fields to show the time and date of the last modification to any of the fields on the screen.

HP ALLBASE/4GL uses the contents of the description fields when you print a report of the components in your application. (The application reporting facilities are available from the developer *Utilities* menu, described in a later chapter.) Depending on the type of report you choose to print, HP ALLBASE/4GL uses the contents of the short description field, or the long description field, in the report.

It's good practice to enter a comprehensive description for all items as you define them. This will help you keep track of the items you define, and their purpose in the application.

If You Make a Mistake ...

If you make a mistake, use the cursor keys, **Tab** or **Shift** + **Tab** keys to move to the field containing the error. Then you can correct the error.

Completing and Committing the Screen

When you have completed the entries on this screen and all fields contain the correct values, press the **Commit Data** function key to actually create the field specification.

HP ALLBASE/4GL creates the field specification that you have just defined, clears the screen and returns the cursor to the *Field Spec. Name* field so you can enter a new field specification name.

You can now define the rest of the field specifications for the customer details. The remaining field specifications are described on the following pages.

To Complete the Exercise

For the remaining field specifications, this lesson only lists the values for fields that require an entry. For all other fields, simply press the **(Return)** or **(Tab)** key to accept the default. Enter some meaningful text in the description fields, and don't forget to press the **Commit Data** function key after entering each field specification.

Field Specification – *Customer Name*

This is a 30 character alphanumeric field.

Field Spec. Name	<code>customer_name</code>
Field Length	30
Edit Code	X

Field Specification – *address*

Since this field could be used for more than just customer records, it is called *address*, not *customer.address*. It contains three lines of 30 characters each, which is the reason for the multiple number of occurrences. As you use the address field specification through the system, you will see how the various HP ALLBASE/4GL facilities handle multiple occurrences of fields.

Field Spec. Name	<code>address</code>
Field Length	30
Repeated <input type="checkbox"/> Times	3
Edit Code	X

Field Specification – *state_code*

For this field, you are specifying that the edit code is *forced uppercase*. This means that if any entered data is lowercase, it is automatically converted to uppercase.

By specifying a minimum entry length equal to the field length, you are ensuring that only a two character entry is valid for this field.

Field Spec. Name	state_code
Field Length	2
Minimum Entry Length	2
Edit Code	U

Field Specification – *zip_code*

Once again all five characters of this field must be entered, so you must specify a minimum entry length of five characters.

Field Spec. Name	zip_code
Field Length	5
Minimum Entry Length	5
Edit Code	N
Blank When Zero	Y

This completes the definition of the field specifications that are used in the *customer* file. The next step is to create the record layout for the file. Exit from this screen by pressing the **Previous Menu** function key. You can return to the main menu by pressing this key again, or continue to the next lesson from the *Dictionary* menu.

Summary

The field specifications stored in the dictionary define the essential characteristics of the fields used in your application. As you store field specifications in the dictionary, HP ALLBASE/4GL does not allocate any physical file space for the fields. The field specifications details that you enter are only definitions.

The actual allocation of physical file space only occurs when you create the files required by your application.

HP ALLBASE/4GL allows you to define the following field specification parameters:

- The field specification name.
- Whether the field specification is secured or not.
- How long the field is, and whether it is repeated or not.
- If the field has a minimum entry length.

- What the edit code of the field is, and how it is justified.
- What pad character is used for the field.
- If there are any decimal places associated with the field.
- If a numeric field is displayed as blank when it only contains zeros.

You can also specify how data entered for the field is validated.

HP ALLBASE/4GL allows you to define help screens for the fields you are specifying. You can also enter descriptive documentation that will be helpful when you review your application.

You have just defined a number of field specifications. On their own, they have no data storage capabilities. Typically, the next step is to define the record layouts of your files or SQL tables. Defining a record layout is not necessarily the next step. However, it is much easier to create your screens, logic blocks, and reports if your record layouts and files or SQL tables are defined.

Lesson 2 – Records

Objectives

In this lesson you will create a record layout using the field specifications you have just defined.

To create a record layout you will:

1. Define a record header.
2. Define the record layout details.
3. Generate the record layout.

Record Layouts

A record layout defines the format of the records in a data file. The layout specifies the names of the fields, and their sequence in the record.

The record layout definition also specifies which fields are used as key fields for accessing records in the file. The KSAM file manager requires at least one key to a record so it can maintain an index of the records in a data file.

Creating a record layout also defines a file record buffer for reading and writing file records.

Record Layout Header

A record can be associated with one or more files and is used in logic blocks, screens, and reports to qualify a field reference. To create a record layout you must complete the record layout header screen and then complete the record layout details screen.

Menu Path

Dictionary, Record Layouts, Header.

Description

This screen enables you to describe a record for use in the application.

Developer		Record Layout Header		record_header	
Record Name	customer_rcrd	Secured	N	(Y/N)	
Description					
customer_rcrd					
AUTHDR: rmjones					
This record layout contains the customer details. It is the only record layout for the customer file.					
Last Modification:		Date		Time	
Field Specs	Record Details	File Defn.	Work Areas	21 51	System Keys
					Commit Data
					Help
					Previous Menu

Entering the Field Values

Record Name. Enter customer_rcrd

This is the name used to refer to the record in a file definition or file record field reference.

Secured. Accept the default value of N

The entry in this field specifies whether this item is secured against modification by an unauthorized developer.

Description. Enter a meaningful description for this item.

Completing and Committing the Screen

When all of the values on the screen are correct, press the **Commit Data** function key to create the record layout header. Now you must define the record layout details. Press the **Record Details** function key to go to the record layout details screen, or use **Previous Menu** function key to go to the record layout details screen via the menus.

Record Layout Details

After you have defined the record layout header, you can define the details of the record layout.

Menu Path

Dictionary, Record Layouts, **Details**.

Description

This screen enables you to define the field specifications making up this particular record layout. You can also state whether a field is to be a key for the file, and if it is, whether duplicate values are permitted in the field.

As with the process of defining a field specification name, you are not allocating any physical file space. The physical file space is allocated when you create the data file. When the application is running, HP ALLBASE/4GL uses the record layout to allocate memory space for a buffer that is used for accessing the file record.

Developer		Record Layout Details			record-details	
Record		customer.rcrd		Secured N		
Field Number	Action (A/C/I/D)	Field Spec. Name	Key Number	Duplicate Keys		
1	A	customer.no	1	N (Y/N)		
Field Specs.	Record Header	File Defn.	Generate Record	7	4	System Keys
						Commit Data
						Help
						Previous Menu

Record. Accept the default entry of `customer_rcrd`

This is the name of the record layout. Processing can only continue if you have already completed the header for this record layout.

If you have not used either of the suggested ways to get to this screen you may not have a default entry of `customer_rcrd`. You can enter the name yourself if it is not displayed in the *Record* field on this screen.

Field Number. Accept the default value of 1

This defines the order of the fields contained in the record. As you define the structure of the record layout, you must declare each field in ascending field order. All the numbers must be contiguous, but it is possible to insert an extra field into the record layout at a later stage. This is the first field in the `customer_rcrd` record.

Action. Accept the default action of A

This defines the action you wish to perform on this field. An entry of A indicates *Add* a new field.

When defining items there are a number of occasions where you will be prompted for an action to be performed. The codes are universal, but some items use more actions than others. The codes are: A = Add, C = Change, I = Insert, and D = Delete. If an item does not exist, the default is A; if the item already exists, the default action is C.

Field Spec. Name. Enter `customer_no`

This is the name of a previously defined field specification. If you try to enter the name of a field that you have not defined, HP ALLBASE/4GL displays an error message.

HP ALLBASE/4GL uses this entry to determine how much space to allocate for the data stored in this field, as well as the valid data types for the field.

Key Number. Enter 1

This entry is for keyed file access purposes.

The key number defines the index number to be associated with this field. Every key number in a record must be unique. Key number 1 is called the primary key.

The KSAM file manager requires at least one key so it can maintain an index for the location of a record in a file. Without a key, no indexed access to a record via that field is possible. The key number has no relationship to the field number.

Duplicate Keys. Enter N

This field indicates, for a key field, whether duplicate records with the same key value can occur on the one file. An entry of *N* indicates that duplicate keys are not permitted. That is, a given key value may occur in only one record in the file.

When the Line is Complete

When you press the **Commit Data** function key, a scroll area below the data entry line shows the details of your entry. As you enter more fields into the record layout, this display reflects the current state of the record layout.

The display shows the field numbers, names, key numbers, and whether duplicate keys are permitted.

To Complete the Exercise

Only the values that require an entry are shown below. If you make an error in a field you have committed, you can rectify it easily. First, enter the field number of the line that you want to edit, then accept the action code of *C*, rectify the error and then press the **Commit Data** function key.

You will see that three of the fields have key numbers. In total, the customer file has four keys. This allows you to read the file (or print reports) according to the customer number, customer name, state code, or zip code. Only the customer number field is defined as a unique key since duplicate values may occur in the other fields.

The entries shown here are required to complete the record layout details.

Fields	Your Entry
Field Number	2
Action	A
Field Spec. Name	customer_name
Key Number	2
Duplicate Keys	Y
Field Number	3
Action	A
Field Spec. Name	address
Field Number	4
Action	A
Field Spec. Name	state_code
Key Number	3
Duplicate Keys	Y

Field Number	5
Action	A
Field Spec. Name	zip_code
Key Number	4
Duplicate Keys	Y

This screen image shows how the record layout screen should appear when you have entered all the field details.

Developer		Record Layout Details		record.details	
		Record	customer_rord	Secured N	
Field Number	Action (A/C/I/D)	Field Spec. Name	Key Number	Duplicate Keys (Y/N)	
1		customer_no	1	N	
2		customer_name	2	Y	
3		address			
4		state_code	3	Y	
5		zip_code	4	Y	

Field Specs.	Record Header	File Defn.	Generate Record	7	4	System Keys	Commit Data	Help	Previous Menu
--------------	---------------	------------	-----------------	---	---	-------------	-------------	------	---------------

Generating a Record

Before a record layout can be used in an application, it must be generated.

Generating the record layout creates a run-time version of the record layout and also performs some additional validation and processing. The generation process resolves pointers to the field specifications, and produces an executable form of the record definition.

Generate the record layout by pressing the **Generate Record** function key. When you use the generate program for the first time two messages are displayed. The first message tells you that *generate* is being called, and the second confirms that the record itself is being generated.

Generate is a large program and is a separate unit in the HP ALLBASE/4GL suite of programs. To avoid the need to call generate repeatedly, it remains resident in memory until you exit from the developer. The next time you use generate in this session, the *calling generate* message will not be displayed.

If you receive an error from the generate process, a generate error screen shows the number of the field that is in error, and the type of error encountered. Note these details and use them to assist you in correcting the record.

If the generate is successful you will receive a *Generation Successful* message indicating that the record layout has been correctly generated.

You have now created and generated a record layout. In the next lesson you will use the record layout to create a data file.

While developing an application you may want to look at the details of an existing record layout. To do this, enter the record layout name in the *Record* field of the *Details* screen and then press **Return**. The contents of the record layout will then be displayed.

Summary

In this lesson you created a record layout.

To create the record layout you performed the following tasks:

1. You defined a record layout header. The record layout header allows you to name a record layout, and enter a description of the record layout.
2. You defined the record layout details. The record layout details screen allows you to specify the fields that make up the record layout, and the order of the fields in the record.

The record layout details screen also allows you to specify whether a field is a key field. If you do specify that a field is a key field, you must specify whether duplicate values are permitted for that field.

3. You generated the record. The generation process calls an external program that validates the details of a record layout. Generation resolves references to the fields on the record, and creates an executable form of the record layout.

Lesson 3 – Files

Objectives

In this lesson you will define and create the KSAM data file for this part of the application.

To create the file you will:

1. Complete the file definition.
2. Create the physical file.

Data Files

HP ALLBASE/4GL stores data for an application in data files. You have already defined the fields that make up the file records. You have also defined the record layout that is to be associated with the data file.

The data file you will create in this lesson is a KSAM file. KSAM is an abbreviation for “Keyed Sequential Access Method”. HP ALLBASE/4GL includes a KSAM data manager that allows you to create and access KSAM data files.

When you created the record layout in Lesson 2, you allocated a key number to several fields on the records layout. The KSAM file manager uses the key fields to define the indexes for the file. Key number 1 in a record layout is called the primary key.

Defining a Data File

There are two steps involved in creating a data file. First, you must define the characteristics of the file, and then physically create it on disk.

Menu Path

Dictionary, Database Items, **Data File/SQL Table Definition**

Description

In this screen you specify the name of the file to be used in HP ALLBASE/4GL, the name by which it is known to the host operating system, and the record layout to be used for the file. You can also enter a brief description that is used for documentation purposes.

Developer		File/SQL Table Definition		file_defn	
File Name	customer	File Type	I (I/S/F/V)		
Description	[Redacted]				
Last Modification:	Date	Time			
Field Specs.	Record Header	SQL Sel. Details	Create File	3 31	System Keys
					Commit Data
					Help
					Previous Menu

This screen uses two different windows. One window allows you to define KSAM data files and serial data files, and the other allows you to define HP ALLBASE/SQL tables. Your entry in the *File Type* field determines which window is used.

Entering the Field Values

File Name. Enter *customer*

This is the internal name of the data file used by the application. Note, the name can be no longer than be eight characters.

File Type. Enter *I*

This indicates the type of file interface to be used. In this case it is the KSAM file interface. HP ALLBASE/4GL displays the KSAM file window automatically.

External Name		custmr	
Record Layout List		Default Record	
1	cust-rec	7	
2		8	
3		9	
4		10	
5		11	
6		12	
Field Specs.	Record Header	SQL Sel. Details	Create File
6	12	System Keys	Commit Data
		Help	Previous Menu

The other options for the *File Type* field allow you to define SQL base tables (S), fixed length record serial files (F), and variable length record serial files (V). Refer to the *HP ALLBASE/4GL Developer Reference Manual* for more information about these file types.

Description. Enter a suitable description of the file.

External Name. Enter `custmr`. (Type over the default value in the field.)

This is the name of the file in the MPE XL environment. By default this is the same name as the internal file name. You can enter a different MPE XL file name that can contain up to seven characters.

You can add a group and account name to this file name if required. Refer to the *HP ALLBASE/4GL Developer Reference Manual* for more details.

For each HP ALLBASE/4GL file, the KSAM data manager creates a number of physical files to store the data and the key file. To allow for data file reformatting, HP ALLBASE/4GL creates a file that contains information about the structure of the data file.

Default Record. Accept the default value of 1

This value indicates the record layout number to be used if no record name is specified in a file field reference. It is also used by the KSAM file manager to create the file. You can't use a value other than 1 if you only have one record layout.

Record Layout List

The next 12 fields on this screen allow you enter the names of up to 12 different record layouts to be associated with this file.

HP ALLBASE/4GL always builds the physical disk file records according to the default record layout.

Using multiple record layouts in the file definition allocates a file buffer in memory (when the application is running) for each record layout assigned to the file. If you use multiple record layouts for a file, you can read more than one physical file record at the same time, or you can read the same physical record through different file buffers, allowing you to modify one buffer without destroying the contents of another.

If you do use multiple record layouts with a data file, make sure that all record layouts have the same length, or that the default record layout is the longest. If any record layout is longer than the default record layout, the extra characters will be truncated when the record is written to the file.

Record Layout List – 1. Enter `customer_rcrd`

This is the name of the record layout to be associated with this file.

Completing/Committing the Screen

When you have entered all the values for this screen, press the **Commit Data** function key to create the file definition.

This is the logical file definition for use in HP ALLBASE/4GL. Your next step is to create the physical disk file. Go to the next step directly by pressing the **Create File** function key, or via the menus using the **Previous Menu** function key and selecting *Create File/SQL Table*.

Creating the Data File

This is the second step in creating a data file. During this procedure the KSAM file manager is automatically invoked to create the physical files that are necessary to support the data file that you have defined.

Menu Path

Dictionary, Database Items, **Create File/SQL Table**

Description

On this screen you can enter the name of a data file you have already defined. HP ALLBASE/4GL reads the file definition and displays the details so you can confirm your selections. You can then create the file.

Developer		File/SQL Table Creation		create_file	
File Name	customer				
File Type	I				
External Name	custmr				
SQL Access Class					
SQL DBEfilesset					
		Key Field List			
Key	Dup	Key	Dup	Key	Dup
1. customer_no	N	7.		13.	
2. customer_name	Y	8.		14.	
3. state_code	Y	9.		15.	
4. zip_code	Y	10.		16.	
5.		11.			
6.		12.			
Description					
customer					
AUTHOR: rmjones					
This file contains the customer number, address, state code, and zip code for all customers.					
Last Modification: Date 10/11/88 Time 09:57:47					
File Defn.	Reformat File	Delete File	3 34	System Keys	Commit Data
					Previous Menu

Entering the Field Values

File Name. Enter `customer`

This is the name you used when you created the file definition.

This completes the data entry required on this screen. The remaining fields display the data you entered on the record layout screen.

Completing the Procedure

When you press **Return** the details of this file are displayed on the screen. The MPE XL external file name is displayed first, followed by the key fields from the default record.

You will see that the four keys, defined when you created the record definition, are listed, along with an indicator specifying whether duplicate values are allowed. The description of the file is displayed in the lower portion of the screen.

To create the file on disk, press the **Commit Data** function key. When this process has completed, HP ALLBASE/4GL displays a message indicating that the file has been successfully created.

Summary

In this lesson you created the KSAM data file for your application.

To create the file you completed the file definition, and then used the file creation screen to build the physical disk files for the data file.

The KSAM file manager uses the key fields on the default record for the file to define an index for the location of each record in the data file. Key number 1 in a record layout is called the primary key.

Contents

Chapter 5: Screens

Screen Types	5 - 1
Menus	5 - 1
Data Screens	5 - 2
Windows	5 - 2
Function Keys	5 - 2
System Items	5 - 3



Lesson 4 – Menus

Objectives	5 - 4
Developing a Menu	5 - 4
Screen Header	5 - 4
Screen Painter	5 - 6
Painting a Menu	5 - 7
Entering Text	5 - 8
Layout Keys	5 - 9
Defining Blocks	5 - 10
Defining System Items	5 - 11
Saving the Screen	5 - 16
Summary	5 - 16

Lesson 5 – Data Screens

Objectives	5 - 17
Defining Data Screens	5 - 17
Defining the Screen Header	5 - 18
Painting a Data Screen	5 - 20
Entering Text and System Items	5 - 21
Using Vertically Aligned Prompts	5 - 21
Defining Data Fields	5 - 22
Using the Dictionary Field Specification	5 - 22
Specifying the Number of Characters	5 - 23
Creating the Remaining Fields	5 - 24
Creating Display Only Fields	5 - 24

Summary of Field Creation	5 - 24
Saving the Screen	5 - 25
Specifying Screen Field Details	5 - 25
Introducing Data Movement	5 - 27
Behavioral Characteristics	5 - 29
Function Specifications	5 - 30
Validation and Format Specification	5 - 31
Generating a Data Screen	5 - 33
Summary	5 - 33

Lesson 6 – Screen Processing Tutorial

Objectives	5 - 35
Screen Fundamentals	5 - 35
What is an HP ALLBASE/4GL Screen?	5 - 36
Screen Types	5 - 36
Menus	5 - 36
Data Screens	5 - 37
Windows	5 - 37
Displaying Screens	5 - 38
Function Keys	5 - 38
Menus	5 - 39
Menu Actions	5 - 39
Data Screens	5 - 40
Running the Tutorial Application	5 - 40
Data Screen Items	5 - 41
Other Screen Features	5 - 42
Data Screen Processing – Overview	5 - 43
Screen Level Processing	5 - 43
Screen Commit Actions	5 - 44
Field Level Processing	5 - 45
Display Processing	5 - 45
Input Field Processing	5 - 45
Field Commit Actions	5 - 45
Input Processing Sequence	5 - 46
Data Formatting	5 - 46
Data Validation	5 - 48
Data Movement	5 - 51
Data Buffers	5 - 51
Screen Field Buffers	5 - 51
Data Movement Fields	5 - 52
The Primary Data Movement Field	5 - 53

The Default Data Movement Field	5 - 53
Other Data Movement Fields	5 - 53
Automatic Data Movement	5 - 54
Primary Data Movement	5 - 55
Default Data Movement	5 - 55
Screen Field Functions	5 - 56
The SHOW Command	5 - 56
SHOW Command Processing	5 - 57
Summary	5 - 58
Screen Fundamentals	5 - 59
Data Screen Processing	5 - 59
Field Display Processing	5 - 59
Input Field Processing	5 - 60
Screen Field Data Buffers	5 - 60
The SHOW Command	5 - 60

Intentionally Blank

5 Screens

The lessons in this chapter describe the different types of screens available in HP ALLBASE/4GL and show you how to create them. HP ALLBASE/4GL screens are quite different from the screen handling techniques you may have used in more conventional programming environments. In particular, HP ALLBASE/4GL screens provide extensive facilities for automatic data formatting, data validation, and data movement.

The lessons in this chapter include a screen processing tutorial that introduces you to some of the techniques and terminology associated with HP ALLBASE/4GL screens.

In the next two lessons you will create the following screens:

1. A main menu to provide the user with an initial choice of actions.
2. A data screen to allow the user to enter data into the customer file.

Screen Types

HP ALLBASE/4GL uses the following screen types:

- Menus.
- Data screens.
- Window screens.

Menus

Menus present the end user with a choice of options within the application. When the user chooses an option from a menu, HP ALLBASE/4GL executes the selected option. A menu allows users to control the application by initiating a menu selection to suit their requirements. HP ALLBASE/4GL can execute the following types of actions as a result of user selections from a menu:

- A further menu, or a data screen.
- A function.

- A help screen.
- A process.
- A background process.
- A report.
- An external program.

When HP ALLBASE/4GL displays a menu, any current activity in the application is terminated immediately.

Data Screens

Data screens allow the user to enter data for the application, and allow the application to display information for the user. Data screens contain input fields, display fields, and literals or titles. Data screens can also have a scroll area for information display. This is particularly useful if you have more information to display than can readily fit on the screen.

A data screen can be overlaid with a window screen. A window screen can present additional information to the user as well as accepting input from the user.

Windows

A window is a screen image that is overlaid on an existing data screen to allow entry or display of additional data. You can display a number of successive windows on the same screen. This is useful if you have a basic screen that you want to keep in front of a user, but cannot fit all of the necessary input and display fields into the available screen area. Windows cannot be used on menus.

Function Keys

All screens can use a set of function keys. The function keys allow the user to select various application options or system functions that you may need in your application.

System Items

All screens can contain system items. These are fields that display the contents of various system or application defined fields. System items can display the following information for the end user:

- The current date and time.
- Communication area fields.
- Master titles.
- Application titles.
- Numeric or alphanumeric constants.
- Variables or calculated items.
- Scratch-pad fields.

Lesson 4 – Menus

Objectives

In this lesson you will create the main menu for the *training* application.

To create the main menu, you will:

1. Use the screen header screen to define the name, and the main characteristics for the main menu. You will also enter some descriptive documentation.
2. Use the screen painter to perform the following tasks:
 - a. Enter and edit text items on the menu.
 - b. Move, copy, and delete single items on the screen, or a group of items on the screen.
 - c. Move, copy, and delete items within a defined “block” on the screen using the block move, copy, and delete functions.
 - d. Create a number of system items.
 - e. Create a menu action item, and then define the text for the action item.

Developing a Menu

Defining a menu involves two steps. You must first define the screen header screen to specify the main characteristics of the menu. You can then use the screen painter to create the actual menu image.

The menu you are about to create initiates a process that controls the *customer_scrn* data entry screen. You will define the process itself in lesson 7.

Screen Header

Menu Path

Screens, Header

Description

This same *Screen Header* screen is used for menus, data screens and window screens. When you specify that you are defining a menu, certain fields on the screen header screen default to entries that are appropriate for menus.

Developer	Screen Header	screen_header
Screen	main	Secured <input checked="" type="checkbox"/> (Y/N)
Screen Type	M	(M/D/W)
Automatic Numbering	Y	(Y/N)
Clear Fields on Commit	B	(I/D/B/N)
Function Keys	main	
Screen Help Name	main	
Scroll:	Top line	Bottom Line Direction (U/D)
Window Starting Line Number		
Description	main AUTHOR: rmjones This is the main menu for the training application.	
Last Modification:	Date	Time
Function Keys	Screen Painter	Field Details
Generate Screen	21	27
System Keys	Commit Data	Help
Previous Menu		

Entering the Field Values

Screen. Enter *main*

This is the name of the menu to be used in this application. This name is important to a more experienced end user, as it is the name which is used to select this menu using the menu bypass feature.

Secured. Accept the default value of

This entry specifies whether this item is secured against changes by an unauthorized developer.

Screen Type. Enter *M*

This entry indicates the type of screen that you are defining. In this case *M* indicates that the screen is a menu. The other types of screens are described later in this guide.

Automatic Numbering. The default entry of *Y* cannot be modified when you are defining a menu.

Clear Fields on Commit. The default entry of *B* cannot be modified when you are defining a menu.

Function Keys. Accept the default value.

Screen Help Name. Accept the default value.

These fields are covered in more detail in a later lesson.

Scroll and Window. As scrolling and windows are not available on menus, these fields are automatically bypassed.

Description. Enter some suitable text.

When the Entries are Complete

When you have completed the entries, press the **Commit Data** function key to create the screen header.

The screen header is now complete. You can now paint the screen image associated with this header using the screen painter. Select the painter with the **Screen Painter** function key, or return to the previous menu and then select *Painter* from the screens menu.

Screen Painter

The screen painter allows you to interactively create the menu image that is seen by the end user.

Menu Path

Screens, Painter

Description

You use the screen painter to create all types of screens. HP ALLBASE/4GL varies some of the available functions depending on what type of screen you are developing.

A menu is composed of literal text, system items such as the date and time, and menu items. A menu item is composed of two parts; its prompt (what the user sees) and its action (what is executed when the user activates the item).

When you use the screen painter, you build an exact image of the screen as it appears in the application. You use the cursor keys to position items on the screen and the function keys to select the various available screen painter options. Any screen item can be located anywhere on the screen.

The screen painter also provides facilities that allow you to modify screens by copying, moving, and deleting single items or groups of items on a screen.

Painting a Menu

When you enter the screen painter for the first time, the screen is cleared and the following prompt appears at the base of the screen.

The screenshot shows a terminal window with the prompt "Enter screen name" at the top left. Below the prompt, the word "main" is entered. To the right of the input field, there are several function keys: "Abort Painter", "More Keys", "24", "1", and "Abort". The "Abort Painter" key is highlighted with a shaded background.

Enter the name of the screen you want to paint. The last screen name used in any of the screen definition screens appears as a default entry. In this example the name should default to *main*. Press **(Return)** to start the screen painter.

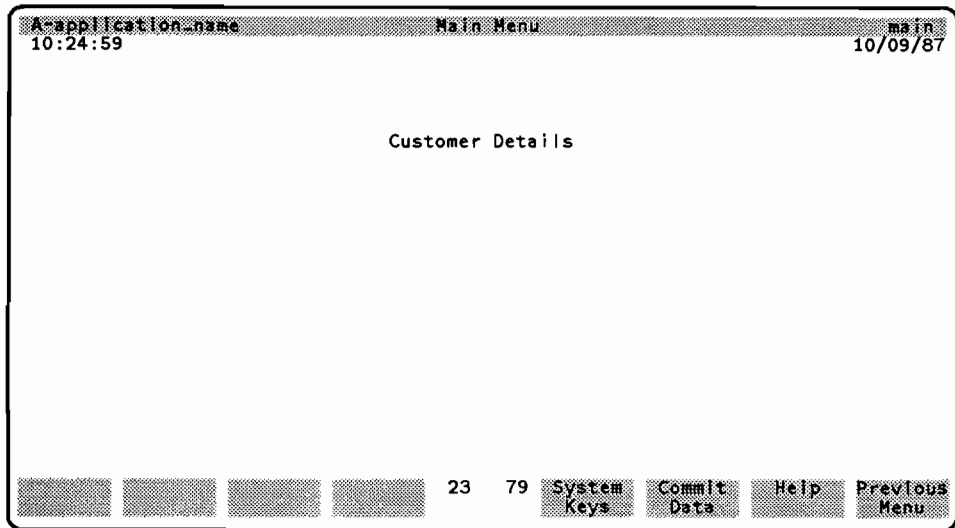
Note

If you want to abort the screen painter at any stage you can press the **Abort Painter** function key, which is accessed through the **More Keys** function key. The screen painter asks you to confirm the request. Enter **Y** in response to this message to abort the screen painter.

When you enter the screen name, the painter positions the cursor at the top left corner of the screen, displays some screen identification details at the bottom of the screen, and displays the screen painter function keys.

On most terminals, the numbers corresponding to the current cursor position are displayed between the two sets of four function key labels. This information helps you position your screen items exactly where you need them.

This is the screen image you are about to paint.



Entering Text

The first item you need to enter on this menu is the screen title. It is located on the top line of the screen.

The top line of a screen is known as the “banner line”. When the screen is displayed at run-time, the banner line is displayed in half-bright inverse video. The system administrator can change the display highlighting of all screen items.

Use the cursor keys, **▶**, **◀**, **▲**, and **▼** to move the cursor to line 1, column 36 on the screen. The cursor coordinates are displayed between the two sets of function key labels at the bottom of the screen. Press **Return** when you have moved the cursor to the correct position. The painter displays a message telling you that it is in *Enter Text* mode. Type in the text **Main Menu**. Terminate the text item and the *Enter text* mode by pressing **Return**.

When in text entry mode, you can use the **Insert char**, **Delete char**, **Clear line** and **Back space** keys to edit the text you are entering. Try it out. Type in a few extra items of text anywhere on the screen. Remember to press **Return** to start

text entry mode, and to press **Return** again to terminate each entry. Don't worry about cluttering up the screen with extra entries because you will soon learn how to delete them.

Try moving from one field to another by using the **Tab** key. Notice that as you move onto a defined item it is highlighted and it becomes the current item. When an item is highlighted, you can modify it by pressing **Return** and typing additional characters over the item. The screen painter enters the *Modify text* mode if you press **Return** while an item is highlighted. Press the **Return** key to complete a text entry or an alteration.

After you've typed in a few text items, try pressing the **Tab** key. Each time you press it, the next item on the screen is highlighted until you get to the last item on the screen. Pressing the **Tab** key again returns the highlight to the first item on the screen. Pressing **Shift** + **Tab** followed by **Return** takes you backwards through the fields. Notice that as you move from one field to another using this method the cursor is positioned on the first character of the current item. Now try pressing the **⏪** key followed by **Return**. The cursor is positioned in the upper left corner of the screen. When you press **Shift** and **⏪** followed by **Return** the cursor is positioned on the last item on the screen.

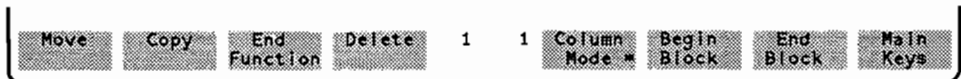
If you've tried out a few of the items described above, your screen is probably a scattered mess of text items. If you don't have more than one text item on your screen, enter some now as you'll need them for the next exercise in this lesson.

Layout Keys

The screen painter layout keys are a set of function keys allowing you to move, copy, or delete items on a screen.

To use the layout keys, make sure you are at the main level of function keys. You can tell this by the fact that one of the function keys is labelled **Exit**. If it isn't, press the **Main Keys** function key.

Press the **Layout Keys** function key and you will see a new set of keys displayed. The **Layout Keys** allow you to modify the layout of existing items on the screens.



By using the layout keys you can move, copy and delete single items or blocks of items on the screen. For now all you want to do is delete the text fields that you don't need on this screen. Use the **Tab** key, or cursor keys to select the first field you want to delete, and then press the **Delete** function key. The screen

painter asks you to confirm the deletion. Press the **Y** key and then press **Return** to confirm the deletion. The item is erased from the screen. This is how you can delete a single item from a screen.

Select another item that you want to delete. So you can delete a group of fields, choose one that has a few more items below or to the right of it. If you don't have an area that looks like this, type in a few extra text items so it does. Then press the **Delete** function key. The screen painter displays the confirmation message again. Now press the **Delete** function key again. The next field on the screen is highlighted as well. As you keep pressing the **Delete** key the highlight keeps advancing until you reach the last field on the screen. Each time you press the **Delete** function key, the confirmation message is displayed. If you keep pressing the **Delete** function key (and no other) you will stay in *delete mode*. When you have highlighted the last field you wish to delete, press **Y** and then press **Return** to confirm the deletion. All of the highlighted fields are deleted in one operation.

The same method also works for moving and copying screen items. If you highlight a number of fields by repeatedly pressing a function key, the action will be performed on all of the highlighted fields.

When you're moving or copying items, the painter issues a warning message if you specify a destination that causes a clash of fields or does not allow sufficient room on the screen for any of the items to be moved or copied. If this happens, move the cursor to a new destination and then attempt the move or copy again.

Defining Blocks

The screen painter provides another way of performing a layout function with a group of screen items. The block function works by including all those fields that are within (or partially within) a rectangular block or area that you define on the screen.

Move the cursor to one corner of the rectangle which is to become the defined block, and then press the **Begin Block** function key. Then move the cursor to the diagonally opposite corner of the block and press the **End Block** function key. All the fields that are partially or wholly within the block are highlighted and the function you choose is performed on all of them. You are **always** asked to confirm your actions when you are using the block delete function. This helps ensure that you don't accidentally destroy part of a screen that you are creating or modifying.

Try it out now. Define a block of fields on your screen, and then press the **Move** function key. When you're asked to move the cursor to the new position, move the cursor to the new position of the upper left corner of the block and then press the **Return** key. The fields within the block are moved, retaining their relative

positions within the block. If any of the fields to be moved or copied clash with any other fields already on the screen, the painter issues a warning message and the move or copy is not allowed.

To turn *copy* or *move* modes off, or abort the *delete* mode, press the **End Function** function key.

Create a few more text items and try moving or copying single fields about the screen. Try pressing the **Move** or **Copy** key successively to use a group of items in *move* or *copy* mode. When you feel comfortable with these keys, move on to the next part of the lesson.

Note

The exercises you have completed so far have only used text items. All field types, system items, action items, text, and input or display fields behave exactly the same way. As well as moving the images of fields on the screen, the screen painter also moves or copies the attributes of the fields.

Clean up the screen so you have only the text item **Main Menu** located on the first line at column 36. Then return to the main screen painter function keys (by pressing the **Main Keys** function key if necessary), and complete painting this menu.

Defining System Items

System items are items like the date and time, the current screen name, and some dictionary defined items.

You create these system item fields on the screen by using the *System Item* prompt which is accessed with the **System Item** function key.

```

Enter System Item type
Application Title
Previous Next 24 1 Restore
Type Type Original
  
```

Some of the available system items are:

- Communication areas.
- The date.
- Application titles.

Displaying the Screen Name

You can use a system item to display the name of the current screen.

Move the cursor to column 76 on the first line of the screen and follow these steps to create the screen name item:

Action	Notes
Press System Item	This displays the system item prompt at the base of the screen.
Press Next Type	This displays another system item. Pressing the function key again specifies and displays the next item type until you reach the end of the list. The list then starts again.
Press Next Type until * Communication Area appears	This is an abbreviation for <i>Communication Area Field</i> . You will learn more about these fields later but for now it is sufficient to know that they are values maintained automatically.
Press Return	This moves the cursor from the small single character prompt in the left field to the larger field on the right.
Enter SCREEN	This is a reference to the communication area field *SCREEN, which contains the name of the current screen.
Press Return	This creates the system item field on the screen. The prompt is cleared and the current screen name, along with the cursor, is displayed at line 1, column 76.

Displaying the Date

You can also use a system item to display the current system data on a screen.

Move the cursor to line 2, column 72 and follow the next steps to create the system maintained date item.

Action	Notes
Press System Item	This displays the system item prompt at the base of the screen.
Press Previous Type	This displays another system item type. Pressing this function key specifies the previous type until you reach the start of the list of available types. You then start stepping through the list from the end again.
Specify D Date	This is the system date field. It is maintained by the system. The current system date is displayed whenever this screen is used by the application.
Press Return	This creates the system item date field on the screen. The prompt is cleared and the date, along with the cursor, is displayed at line 2, column 72. Note that the date is displayed in either US or European format. The actual format on your system is set by the system administrator.

Displaying the Time

A communication area field called *TIME holds the current system time. You can use a system item to display this field on a screen.

Move the cursor to line 2, column 2 and follow the steps below to create a field that displays the current time.

Action	Notes
Press System Item	This displays the system item prompt at the base of the screen.
Specify * Communication Area	*TIME is a communication area field.
Press Return	
Enter TIME	This refers to the *TIME communication area field.

Displaying an Application Title

An application title is a literal string that can appear anywhere on a screen. Application titles are stored in the dictionary, and any changes you make to an application title are automatically reflected in all screens that use the title. A typical use for application titles is to display a corporate or department name on all screens in an application. In this example, you will use an application title to display the name of the application on the screen.

Move the cursor to line 1, column 2 and follow the steps below to create a field that will display an application title. You will create the application title in a later lesson.

Action

Notes

Press **System Item**

This displays the system item prompt at the base of the screen.

Specify **A**
Application Title

This is another type of system item.

Press **Return**

Enter
`application_name`

The painter displays a warning message informing you that this item hasn't been defined in the dictionary. You can still create the system item on the screen, creating a reference to the named *application title* even though the item hasn't been defined yet.

At this stage you will only see *A-application_name* displayed. The screen painter does not need to have this field defined when you first refer to it. When you define the field later, it is displayed automatically in this area.

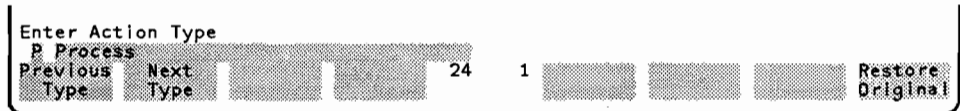
Action Items

So far you have only entered a text item and some system items. On a menu you must also define at least one action to be performed when the user selects the associated prompt.

A menu action can execute any of a number of items in HP ALLBASE/4GL. These include processes, background processes, reports, or other screens.

The following steps allow you to create an action which, when selected, executes an HP ALLBASE/4GL process. Position the cursor at line 7, column 32 and follow these steps.

Action	Notes
Press Action	This displays the action item prompt at the base of the screen.



Complete the following steps to create the action item.

Action	Notes
Press Next Type	Pressing this key specifies and displays the next action type until you reach the end of the list of available action items. You then start stepping through the list again.
Specify P Process	This indicates that this item executes a process when it is selected. You will create the process itself in a later lesson.
Press Return	This moves the cursor from the small single character prompt on the left field to the larger field to the right. The system is now waiting for you to enter the name of the process associated with this item on the menu.
Type customer_proc	This is the name of the process that is executed when the user chooses the <i>Customer Details</i> selection from the menu.
Press Return	This creates the name of the action item on the screen. The prompt is cleared and the process name, along with the cursor, is displayed at line 7, column 32. Now you can overwrite the name of the action with the prompt that is seen by the end user.
Enter Customer Details	This action automatically invokes the text editing mode and overtypes the process name that formed the original action title.

You can edit the text of an action title just like any text field. However, the highlighting differs from that of a text field to enable you to distinguish between the two.

An end user can select the various choices on a menu by typing the first letter of an item and then pressing **Return**. If you use unique initial letters for each of the items on any one menu, an experienced end user can move rapidly through menus.

Saving the Screen

Press the **Exit** function key to exit from the screen painter. The screen painter automatically saves the screen under the name that you allocated. In this case, the name is *main*. The screen is now ready to be used in the application. Menus don't need to be generated.

Summary

In this lesson you defined the main menu for the *training* application.

Menus allow users to control the application by selecting the item they want. HP ALLBASE/4GL can execute the following types of actions as a result of user selections from a menu:

- A further menu or a data screen.
- A function or a process.
- A help screen.
- A background process.
- A report.
- An external program.

Menus are created in two steps:

1. Use the screen header screen to define the menu header. This defines details such as the menu name, and the names of the function key set and help screen associated with the menu. The screen header also allows you to enter a description of the menu.
2. Use the screen painter to create the menu image. The screen painter allows you to define text literals, action items and system items on a menu.

Menus are automatically saved when you exit from the screen painter. Menus don't need to be generated.

Lesson 5 – Data Screens

Objectives

In this lesson you will create a data screen for the *training* application. While completing this lesson you will:

1. Use the screen header screen to define the name and characteristics for a data screen.
2. Use the screen painter to perform the following tasks:
 - a. Paint system items and text items on a data screen.
 - b. Use “column mode” to enter vertically aligned screen items.
 - c. Create a number of data input fields on the screen.
3. Use the screen fields details screen to perform the following tasks:
 - a. Specify the data movement details for the input and display fields on a data screen.
 - b. Specify the behavioral characteristics of screen fields.
 - c. Associate a logic function with a screen field.
 - d. Specify the data validation and data formatting performed for each screen field.
4. Generate the data screen.

This lesson introduces a number of new concepts, and some new terminology. Lesson 6 is a tutorial style lesson that will help to clarify these concepts, and chapter 7 describes how HP ALLBASE/4GL processes data screens.

Defining Data Screens

Data screens allow the user to enter data and allow the application to display information to the user. Data screens can consist of a combination of literals, system items, input and display fields.

Creating a data screen involves three steps:

1. Define the screen header.
2. Paint the screen image with the screen painter.
3. Define the screen field details.

Data screens must be generated before they can be used in an application.

The data screen you create in this lesson allows the user to enter the information necessary to add, delete, modify, or review records in the customer data file.

Defining the Screen Header

Menu Path

Screens, Header

Description

This screen allows you to define the essential characteristics of a data screen. It's the same screen you used to define the header for the menu.

```

Developer          Screen Header          screen.header
-----
Screen  customer.scrn          Secured  # (Y/N)

Screen Type          D (M/D/W)
Automatic Numbering Y (Y/N)
Clear Fields on Commit B (I/D/B/N)
Function Keys       customer.scrn
Screen Help Name    customer.scrn

Scroll:          Top line  Bottom Line  Direction  (U/D)
-----

Window Starting Line Number  #

Description
customer.scrn
AUTHOR: rmjones
This is the data screen that is used
for entering data about the customer.
Last Modification:          Date          Time

Function  Screen  Field  Generate  21  10  System  Commit  Help  Previous
Keys     Painter  Details  Screen  Keys  Data  Menu

```

Entering the Field Values

Screen. Enter `customer_scrn`

This is the name of the data screen in this application.

Secured. Accept the default value of `N`

Screen Type. Enter `D`

This indicates the screen type you are defining. `D` indicates a data screen.

Automatic Numbering. Accept the default value of `Y`

This field specifies the way in which the data fields on the screen are numbered. An entry of `Y` specifies automatic numbering. This means that the fields are numbered sequentially from left to right, and top to bottom, starting at the top leftmost field. This determines the order in which the fields are processed as the user follows the standard tabbing sequence through the screen.

You can assign the field numbering sequence manually if required. However, this exercise uses the automatic numbering facility.

Clear Fields on Commit. Accept the default value of `B`

This indicates which fields are cleared when the user presses the `Commit Data` function key. Possible entries for this field are:

I	=	Input.
D	=	Display.
B	=	Both.
N	=	None.

Function Keys. Accept the default value.

Screen Help Name. Accept the default value.

Scroll: Top line. Don't enter anything in this field.

Window Starting Line Number. Don't enter anything in this field.

Note

The *Function Keys*, *Screen Help*, *Top Line* and *Window Starting Line Number* fields are described in detail in later lessons.

Description. Enter a suitable description.

Completing and Committing the Screen

When you have completed all of the field entries, press the **Commit Data** function key to create the screen header.

The screen header is now complete, and you can create the screen image with the screen painter.

Run the screen painter by pressing the **Screen Painter** function key, or by pressing the **Previous Menu** function key and then selecting the *Painter* option.

Painting a Data Screen

When the screen painter starts, HP ALLBASE/4GL clears the screen and displays the following prompt at the base of the screen.



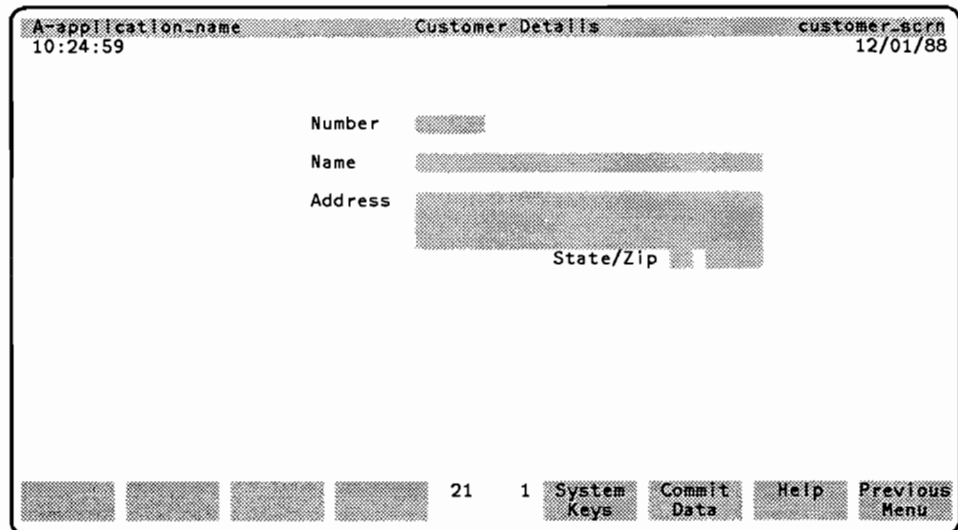
Enter screen name
customer_scrn

24 1

Abort

Check the name of the screen entered here. In this case the name should be *customer_scrn*. It defaults to this name if you have just defined the *customer_scrn* data screen in the screen header screen.

Here is the screen you are going to paint:



A-application_name Customer Details customer_scrn
10:24:59 12/01/88

Number

Name

Address

State/Zip

21 1 System Commit Help Previous
Keys Data Menu

Entering Text and System Items

First create the system items, as you did in the previous lesson.

Display the application title *application_name* at the top of the screen at line 1, column 2.

At line 2, column 2 display the communication area field *TIME.

Enter the screen heading **Customer Details** as a text literal at line 1, column 35.

At column 67 of the same line, display the communication area field *SCREEN.

At column 72 of the same line, display the *DATE system item to complete the system items required on this screen.

This completes the definition of the header lines for this screen.

Using Vertically Aligned Prompts

This data screen has a number of vertically aligned prompts. Each of them has a data input field to the right of the prompt.

During the creation of text items on the menu that you have already defined, the cursor remains at the end of a defined item when you press the **Return** key.

When you are entering a number of vertically aligned text items, you can use the screen painter *column mode*. In column mode, the cursor returns to the beginning of the text item, one line down, when you press **Return** to terminate an entry. Then you can immediately enter the next prompt, vertically aligned with the previous.

To set the column mode on, press the **Layout Keys** function key and then press the **Column Mode** function key. An asterisk appears in the function key label indicating that column mode is now on. To turn it off, simply press the **Column Mode** function key again. The asterisk is cleared from the label when you turn the column mode off.

Set the column mode on, position the cursor at line 6, column 26 press **Return** and type in **Number**. Press the **Return** key to terminate the entry. The cursor will be directly under the “N” in the word “Number”. Press **▼** to move the cursor to line 8, press **Return** and then enter **Name**. Enter **Address** at line 10, and then leave three blank lines before you enter **State/Zip**. Enter **State/Zip** at line 13, column 47. If you prefer, you may enter the words “State/Zip” directly under the word “Address” and then use the move function to move the text item to its correct position.

Turn column mode off and press **Main Keys**. You can now define the data fields for this screen.

Defining Data Fields

To complete this screen you must define the input fields on the screen image. This is done with two function keys at the *Main Keys* function key level: the **Input Field** function key, and the **Output Field** function key.

Enter the numeric size or the Field Spec. name...									
Input Field	Output Field	System Item	Special Text	23	79	Layout Keys	Sequence Numbers	More keys	Exit

You can use two different methods for defining input and display fields. You can retrieve the details of a field specification from the dictionary by entering its name, or you can indicate the length of the field now and complete the other details later.

Using the Dictionary Field Specification

Move the cursor to line 6, column 35 and follow these steps to create an input field for the customer number, using the dictionary field specification.

Action

Press

Input Field

Type

customer_no

Press **Return**

Notes

The create field prompt appears at the bottom of the screen.

The painter creates an input field at the base of the screen as you type the name. This is the name of an existing dictionary field specification. If the field specification does not exist, the field creation is aborted.

This terminates the entry of the field specification name. A field of the correct length is displayed at line 6, column 35.

By using the dictionary field specification name, you have linked the dictionary definition of the field to this particular screen field. When you define further options for some of the field details, you will see that the painter has automatically retrieved the dictionary field specifications for this field.

Use the same procedure to create the customer name field. Move the cursor to line 8, column 35 and follow these steps.

Action	Notes
Press Input Field	This displays the create field prompt at the bottom of the screen.
Type customer_name then press Return	This creates the customer name field on the screen.

Move the cursor to line 10, column 35 and follow these steps to create the input field for the first address line.

Action	Notes
Press Input Field	This displays the create field prompt at the bottom of the screen.
Type address then press Return	This creates the first line of the address.

Remember that *address* is a repeated item. The painter only displays the length of one occurrence of the field. The screen painter always treats a repeating item this way.

Normally you would use the screen painter copy facility to make copies of a field in another location on the screen. The following procedures are included to show you the other ways of creating fields on the screen.

Specifying the Number of Characters

You can also create an input field by specifying the number of characters for the field.

Move the cursor to line 11, column 35 and follow the steps below to create the second line of the address field.

Action	Notes
Press Input Field	This displays the create field prompt at the bottom of the screen.
Type 30	This is the size of this field in characters.
Press Return	This displays the 30 character field that you have just specified.

The field has been created with no reference to a dictionary defined field specification. You will enter the details later in the screen field details screen.

For the final line of the address move the cursor to line 12, column 35 and follow these steps.

Action	Notes
Press Input Field	This displays the create field prompt at the bottom of the screen.
Type 30, press Return	This completes the creation of the field.

As with the address line above, no reference has been made to a dictionary field specification. You will do this later using the screen field detail screen.

Creating the Remaining Fields

Create the two remaining input fields, state code and zip code, using the dictionary field specification method.

Their defined names are *state_code* and *zip_code*. The state code input field should start at line 13, column 57, and the zip code input field should start at line 13, column 60.

Creating Display Only Fields

You can create display only fields using the painter in the same way you create input fields. The only difference is that you use the **Output Field** function key instead of the **Input Field** function key.

You will define some display only fields in one of the later lessons.

Summary of Field Creation

To create an input or display field on the screen, you can:

- Use a dictionary field specification name.
- Specify the field length in number of characters.

Saving the Screen

The screen is now complete and you can save it. Press the **Exit** function key, and the screen is saved automatically. Control then passes back to the level from which you called the painter.

You can now proceed to define some details for the fields on this data screen. This includes the details omitted because you didn't use the field specification name method to define some fields, as well as some other details you will learn about in the next section. If you are at the screens menu select *Details*. From the screen header screen press the **Field Details** function key.

Specifying Screen Field Details

After you have created the image of a data screen with the screen painter, you usually need to specify some further details for the data fields on the screen.

These details include automatic data movement, field-associated functions and some special field processing indicators. For most fields on your data screen, some default values will already be created based on the field specification you used to define the field. In most of these cases you will only need to add the automatic data movement specifications.

Menu Path

Screens, **Details**

Description

This screen allows you to complete the definition of each input or display field on an application data screen.

Developer		Screen Field Details		screen-fields	
Screen	customer_scrn	Sequence Number	1	Secured	N
DATA MOVEMENT	Field Spec. Name customer_no				
Primary	F=customer_no.customer				Data<-->Screen
Default					Data -->Screen
Other 1					Data<-- Screen
2					"
3					"
Input/Display	I (I/D)	Required Field	Y (Y/N)		
Help name	customer_no	Clear on COMMIT	N (Y/N)		
Include in SHOW	Y (Y/N)	Automatic Flow to Next	N (Y/N)		
Function	customer_func	Prior/After/Both	A (P/A/B)		
Perform On SHOW	N (Y/N)	After Function if Blank	N (Y/N)		
Edit Code	N (X/A/U/K/N/S/Q/D/T)		Justification	R (L/R/C/N)	
Field Length	1 Min	6 Max	Decimal Places	0	
Range			Pad Character		
Table			Blank When Zero	Y (Y/N)	
Screen Header	Screen Painter	Function Keys	Generate Screen	17	19
			System Keys	Commit Data	Help Previous Menu

First you must enter the name of the screen, and then step through the fields on the application data screen completing their respective details. The complete screen field details screen is used for each field on your application screen.

Entering the Field Values

Screen. Enter `customer_scrn`

This is the name of the data screen that you are defining field details for.

Sequence Number. Accept the default value of 1

This is the field number, defined by the sequence of data fields on your data screen. The field sequence for this screen is set automatically from top left to bottom right. This sequence of fields and allocation of field numbers is determined by the *Automatic Numbering* flag on the screen header screen.

When you press the **Return** key the details already recorded for this field are displayed on the bottom of the screen. The dictionary specification details have been automatically transferred to this screen, and many of the fields contain a default value.

Field Spec. Name. Accept the default.

This field defaults to the dictionary field specification name used to define the attributes of this screen field in the screen painter.

Note

Blanking out this field changes the last nine fields on the field details screen to input fields. These fields cannot be changed when a link to the dictionary exists through a field specification name. If you have blanked out the field specification name, re-enter the name **customer_no** to protect the screen field specification again.

Introducing Data Movement

The *Primary*, *Default* and *Other* fields define the automatic data movement that occurs as the user moves through the fields on the application screen (The screen data movement system is explained in detail later in this guide.). The three types of data movement are:

- Primary** When HP ALLBASE/4GL displays a screen field, it moves the contents of the *primary* data movement field to the screen and displays the field contents. For an input field, HP ALLBASE/4GL moves the contents of the screen field to the primary data movement field after the user completes data entry for the screen field.
- Default** If the screen field is blank, HP ALLBASE/4GL moves the contents of the *default* field to the screen and then displays its contents. If the screen field is not blank, no action takes place.
- Other** For input fields HP ALLBASE/4GL moves the contents of the screen field to the *other* data movement fields when the user completes data entry for the field. Each input field can have up to three *other* data movement fields.

Note

You cannot specify both a primary and a default data movement field. They are mutually exclusive. A field can only have one source for its data, however, you can combine either of these “source” movement fields with any number of *Other* data movement specifiers.

The fields you name in these automatic movement specifiers can be any of the following:

- A file record field reference.
- A screen field reference.
- A scratch-pad field reference.
- A variable.
- A numeric or alphanumeric constant (default data movement field only).

Primary. Enter `F-customer_no.customer`

This entry specifies that when the cursor is moved to this screen field, the current contents of the field `customer_no` from the file buffer `customer` are moved to the screen buffer and then displayed. On completion of data entry, the new contents of the screen field are moved to the file buffer.

Note

The file record field specifier consists of three components. The prefix *F-* indicates this is a file buffer field reference. Next the field name `customer_no` indicates the field to be used, and the file name `customer` specifies the file buffer to be used. These last two terms are separated by a single period (.). Refer to the *HP ALLBASE/4GL Developer Reference Manual* for further information on naming and referencing rules.

As the *Primary* and *Default* move specifiers are mutually exclusive (you can only move data into the screen field from one source), HP ALLBASE/4GL skips over the *Default* move field. If you wanted to alter this field, you would need to delete the *Primary* move specifier, and then specify the *Default* move field. No *Other* move fields are to be entered in this example. When you leave the first *Other* move field blank, HP ALLBASE/4GL skips over the next two.

Behavioral Characteristics

The next six fields define the way the field behaves on the screen. This is not dependent upon any data type, format, or validation specifications.

Input/Display. Accept the default value of **I**

This indicates whether this data field is an input field (I), or a display field (D). It enables you to override the setting you specified in the screen painter.

Required Field. Accept the default value of **Y**

This indicates whether this field must be non-blank for alphanumeric, or non-zero for numeric fields, for the field value to be accepted.

When the user attempts to move off a required field that has been left blank, the field is highlighted and the cursor remains on it. The user can move back to a field with a lower sequence number, or enter data into the current field. The user cannot commit the screen with the **Commit Data** function key while a required field remains blank.

Help Name. Accept the default value of **customer_no**

This is the name of the help screen that is called if the user presses the **Help** function key while the cursor is on this field.

If you have defined a help screen name for this field specification in the dictionary, that name is automatically transferred to this field. If you haven't specified a help screen name, HP ALLBASE/4GL creates a default help screen name with the same name as the field. You can change the help screen name here without affecting the dictionary specification.

Clear on COMMIT. Accept the default value of **N**

This field works in conjunction with the *Clear Fields on COMMIT* setting specified on the screen header for this screen. The relationship is an OR relationship. That is, if the setting for either is Y, the field is cleared. Only if both are set to N, will the field not be cleared.

Include in SHOW. Accept the default value of **Y**

This specifies whether the field should be manipulated when it is referred to from a logic block SHOW command. With some fields you may want to ensure that the only time anything is ever done to them is when an explicit reference is made to them.

The SHOW command is used to redisplay data from HP ALLBASE/4GL logic in a screen field. You will learn about the SHOW command in a later lesson.

Automatic Flow to Next. Accept the default value of **N**.

So far, all the fields that you have used require you to press the **Return** key to signal that the field entry is complete. In the same way, the fields you define for an application screen normally require the user to press the **Return** key to complete a field entry. Under some operating systems, setting the *Auto Flow to Next* field to **Y** causes HP ALLBASE/4GL to generate an implicit **Return** when the user fills the field. This facility is not available under the MPE XL operating system.

Function Specifications

You can associate a logic function with each screen field. The next four fields specify how the function for this field is executed. This function enables you to perform some data manipulation or validation when data is displayed in the field, or entered into the field.

The field function you are naming here will read the customer file, for a record with a key value matching the value entered by the user.

Function. Enter `customer_func`

This is the name of the function to be associated with this field. If you leave this field blank, entry into the next three fields is not possible.

Prior/After/Both. Enter **A**

This entry specifies when the function is executed. There are three available options. **P** indicates that the function is executed *prior* to data entry, when the user moves the cursor onto the field. **A** indicates that the function is executed *after* the completion of data entry for the field; this is only valid for input fields. **B** indicates that the function is executed both prior to and after data entry.

Perform Function on SHOW. Accept the default value of **N**

Again, this field relates to the behavior of the field under a **SHOW** command from a logic block. At times the function may perform some data formatting or extraction that is essential for the correct data to be displayed. In this case, the function needs to be executed when the field is displayed under control of a **SHOW** command.

After Function if Blank. Accept the default value of **N**

This field only applies to non-required screen fields. It indicates whether the function is executed even if the screen field has been left blank.

Note

You may have noticed that only one function can be called from the field, yet the function can be called at three distinctly different times; prior, after, or on SHOW. HP ALLBASE/4GL has two switches (*ENTERED and *SHOWING) that you can test in the function to determine when the function is being executed. This allows you to use the one function to perform three different operations, depending on the processing stage at which the function is executed.

Validation and Format Specification

The next nine fields on the screen are a copy of the same fields from the dictionary field specification screen. If you have entered a field specification name in the *Field Spec. Name* field at the top of this screen, you cannot change any of this group of fields. You can only enter local field definitions if there is no field specification name associated with the application screen field.

The fields are not described here as they have exactly the same meaning as their counterparts in the field specification screen.

Completing the Screen

Once you have completed all of the details for this particular screen field, press the **Commit Data** function key. When you press this key, the cursor returns to the *Sequence Number* field.

The details for the next field on the application screen are then displayed, enabling you to enter the remaining details.

As you go through the screen fields, HP ALLBASE/4GL creates a primary data movement identifier automatically. This data movement identifier consists of the field specification name for the current field and the file record specified for the first field on the screen.

To Complete the Exercise

As with previous exercises, only those fields that require an entry other than the default are listed here. After entering the details for each field, press the **Commit Data** function key.

If you make an error in a particular field of the screen, you can display its details by entering its sequence number and pressing the **Return** key. The field details are then displayed so you can modify them, and commit them again.

The following entries are required to complete the screen field details.

Customer Name

Sequence Number	2
Primary	F-customer_name.customer

Customer Address Line 1

Sequence Number	3
Primary	F-address(1).customer

The address field above is a repeated item and it must be subscripted. The default entry will not be correct. You can edit it by inserting the (1).

Customer Address Line 2

Sequence Number	4
Field Spec. Name	address
Primary	F-address(2).customer
Required Field	N

When you defined this field in the screen painter, you didn't use the dictionary field specification name method. For this reason a default name is not displayed with the rest of the field details. When you enter the name *address*, the fields at the base of the screen become display fields.

Customer Address Line 3

Sequence Number	5
Field Spec. Name	address
Primary	F-address(3).customer
Required Field	N

State Code

Sequence Number	6
Primary	F-state_code.customer

Zip Code

Sequence Number	7
Primary	F- <code>zip_code.customer</code>

You have now completed the definition of the fields for this screen. As with a record layout, you must generate the screen before it can be used.

Generating a Data Screen

A data screen must be generated to produce an executable version of the screen. To do this, press the **Generate Screen** function key on the screen field details screen. You will see a message verifying that the screen is being generated.

As with record layout generation, the generate program validates field names and resolves pointers. If you have incorrectly specified an item on the field details, an error screen indicates where the error lies. Take note of this and go to the particular screen field, rectify it, and generate the screen again. Once the screen is successfully generated, you can continue to the next task.

You have now completed this lesson.

Summary

In this lesson you created a data screen.

Data screens allow the user to enter data and allow the application to display information to the user. Data screens can have both input and display fields, and can also use system items and text literals.

Data screens allow you to specify the following:

- Screen field data validation characteristics.
- Data movement associated with screen fields.
- Screen field behavior
- Logic functions associated with fields.

Creating a data screen involves four steps:

1. Define the name and characteristics of the data screen on the screen header screen.

2. Paint the screen image with the screen painter. This involves painting system items, input fields, display fields, and text items on the screen.
3. Define the screen field details. This involves specifying the data validation, data movement details, and behavioral characteristics for the input and display fields on the data screen. You can also associate logic functions with screen fields.
4. Generate the screen.

The data screen you created in this lesson allows the user to add, delete, modify, or review records stored in the *customer* data file.

Lesson 6 – Screen Processing Tutorial

Objectives

By now you have been introduced to quite a number of terms and features related to HP ALLBASE/4GL screens. This lesson brings these various terms together in a more complete description of the HP ALLBASE/4GL screen system.

This lesson is presented in a tutorial format. In this lesson, you will be asked to perform some activities that are not directly related to the *training* application. Instead, these activities are designed to demonstrate particular aspects of HP ALLBASE/4GL screens.

This lesson is only an introduction to HP ALLBASE/4GL screens. A review in Chapter 7 describes parts of the screen processing logic in more detail.

By the end of this lesson, you will know:

- The types of screens HP ALLBASE/4GL uses and the types of items that can appear on these screens.
- What causes HP ALLBASE/4GL to display a screen.
- What user actions control screen processing.
- What HP ALLBASE/4GL does when it displays data on a screen.
- The actions performed by HP ALLBASE/4GL when it processes an input field on a data screen.
- The purpose of the logic command SHOW.

Screen Fundamentals

By now you are aware of many of the basic HP ALLBASE/4GL screen concepts, and you have seen some of these concepts in practice. The next few paragraphs revise and summarize some of these concepts.

All of the developer screens you use in developing the *training* application are in fact HP ALLBASE/4GL screens. They demonstrate the features you can include in your own applications. During this lesson you will occasionally be referred back to some of the screens you have already used so you can look at these features in more detail.

What is an HP ALLBASE/4GL Screen?

HP ALLBASE/4GL screens and the automatic logic associated with them form a fundamental part of any application you develop. When HP ALLBASE/4GL displays a screen, the screen and logic associated with the screen control the application. In particular, screens control all interaction between your application and the user. The type of screen, and the way the screen is called, determines where control resumes after the screen.

Screen Types

HP ALLBASE/4GL uses three types of screens. They are:

- Menus,
- Data screens, and
- Windows.

Menus

Menus have the following features:

- They present the user with a range of options in the application. You have already been using menus to select options in the developer application.
Frequently, an option on a menu will call a further menu. For example, this occurs when you select the *Dictionary* option on the developer main menu.
- They allow the use of the keyboard to select options. You have already used the keyboard facilities to select options from the developer menus. Exactly the same keyboard facilities are available to the end users of your applications.
- No action occurs until the user selects and activates an option. When HP ALLBASE/4GL displays a menu, the system enters a “wait state”. Nothing happens until the user selects an option from the menu, and executes the selection by pressing the **Activate Item** function key, or by pressing the **Return** key.

Data Screens

Data screens have the following properties:

- They form the basis of all interactive data input and information display for the application.
- They are the primary means of user data entry.
- They are the primary means of information display.
- They can usually perform most of the data formatting, data validation, and data movement required by an application.

Windows

Windows operate in conjunction with data screens to provide additional data input and information display capability.

A window uses the same types of screen items, and can perform the same data manipulation as a data screen. However, HP ALLBASE/4GL can only display a window on an already displayed data screen. Menus cannot use windows.

You have already seen and used a window on the File/SQL table definition screen in the dictionary menu. You may want to go back to this screen now to refresh your memory. To display this screen, the menu path is:

Dictionary, Database Items, **Data File/SQL Table Definition**

HP ALLBASE/4GL displays two different windows on this screen. These windows allow you to define KSAM data files, or HP ALLBASE/SQL base tables. To see these windows, enter the name of a file (for example *customer*) in the file name field, and then enter **I** in the file type field. When you press **Return**, HP ALLBASE/4GL displays the KSAM file definition window. You have already used this window to define the file *customer*.

Move the cursor to the file type field and enter **S**. This displays the SQL table definition window. Later chapters in this guide describe the use of this window in the SQL part of the training application. For the moment, this example shows how HP ALLBASE/4GL displays two or more windows on the same data screen. Using windows allows you to present the user with data input and display fields in addition to those on the current data screen.

Displaying Screens

HP ALLBASE/4GL displays a screen when it receives a `SCREEN` command in logic, or a `D-` action from a menu, a function key, or a decision table. You have already created an action item on a menu. The action item you created executed a process, but other action items can display further screens. You will learn more about function key actions and decision table actions in later lessons.

The `SCREEN` command is one of the logic commands that can be called in a function or a process. In the next lesson, you will create a process that contains a `SCREEN` command.

Once HP ALLBASE/4GL displays a screen, the screen itself controls the sequence of events occurring in the application. In general, the screen remains in control until the user initiates an action that terminates the screen. This occurs regardless of the type of screen.

For a menu, no further actions or processing occurs until the user selects and activates a menu option.

For a data screen, the screen remains active until the user initiates an appropriate action to terminate the screen, such as the “Commit Data” action.

In general, interaction with the user is not possible if HP ALLBASE/4GL is not displaying a screen.

Function Keys

All screens can have a set of function keys. You have already seen and used the function keys associated with the screens in the developer.

The function keys associated with a screen can call any of the actions that can be called from a menu. Function keys can also call a number of internal actions such as calling a further set of function keys. Two important internal actions are the *Commit Data* action, and the *Previous Menu* action. These two actions are described in more detail later in this tutorial, and in later chapters in this guide.

HP ALLBASE/4GL automatically provides a default set of function keys that meet the basic needs of all screens. Later in this training course you will see how you can define specialized function keys for your application screens.

Menus

When HP ALLBASE/4GL displays a menu, all current activity in the application (except background processing) is terminated immediately. Nothing further happens until the user selects and executes one of the options on the menu, or initiates an action from a function key.

In general, HP ALLBASE/4GL displays a menu when an application first starts, and when the current process terminates.

In many situations, you will deliberately design processes so they are only terminated when the user presses the **Previous Menu** function key. When you return to the previous menu, the current process is terminated. You may have noticed that HP ALLBASE/4GL issues a warning message if you have started to change a screen, but you have not committed the changes before you press the **Previous Menu** function key. HP ALLBASE/4GL presents the same warning message to your application users.

Menu Actions

Menus can call any of the following types of actions:

- A process.
- A further menu.
- A data screen.
- A report.
- An external program.
- A background process.
- A function.
- A help screen.

The menu you created in lesson 4 calls a process.

Data Screens

The remainder of this tutorial concentrates on data screens. Data screens can usually perform most, if not all, of the data validation, data formatting, and data movement required by an application.

A good understanding of the working of data screens can simplify your task in developing applications, and at the same time can help you present your end users with screens that make their tasks easier.

This tutorial is intended to provide an introduction to some of the concepts involved with screen processing. Later chapters in this guide cover these aspects, and others, in greater detail. Don't worry if some of the concepts seem new or strange at this time. You will soon feel comfortable with them. You may want to return to this tutorial at a later stage to review some of this material.

To demonstrate some of the topics covered in this tutorial, we have supplied a small application as part of the developer system. While you are working through the rest of this tutorial, you will be referred to this tutorial application at various times so you can observe the screen processing features.

The developer source code for the *tutorial* application is included with the application. This means you can sign on to the application as a developer to examine the details of the application.

Note

The source code screens for the *tutorial* application has the *Secured* field on the various screens set to **Y** to prevent accidental modifications. If you do want to experiment by changing parts of the application, you must set the *Secured* field for the component concerned to **N**.

Remember to return the component to its original state if you do make any changes.

Running the Tutorial Application

You may want to look at the *tutorial* application before you read through the rest of this lesson. To start the application, exit from your current application, sign on to HP ALLBASE/4GL as normal, and enter the name **tutorial** in the *Application or Version to be developed* field. This takes you into the developer main menu

as normal. To look at the *tutorial* application, select the *Application Testing* option on the developer main menu. You are taken into a new HP ALLBASE/4GL environment allowing you to run the application under development.

In the application testing environment, an application behaves exactly as it will appear to the end user. (The testing environment does provide some additional features to assist testing applications. These are explained in a later chapter.)

When you enter the application testing environment, you will see the main menu of the *tutorial* application. This tutorial application does not perform any useful data processing function. In fact, any data you enter is sent to the “bit bucket” as soon as you press the **Commit Data** function key. The application itself has deliberately been kept very simple so you can examine its various features easily.

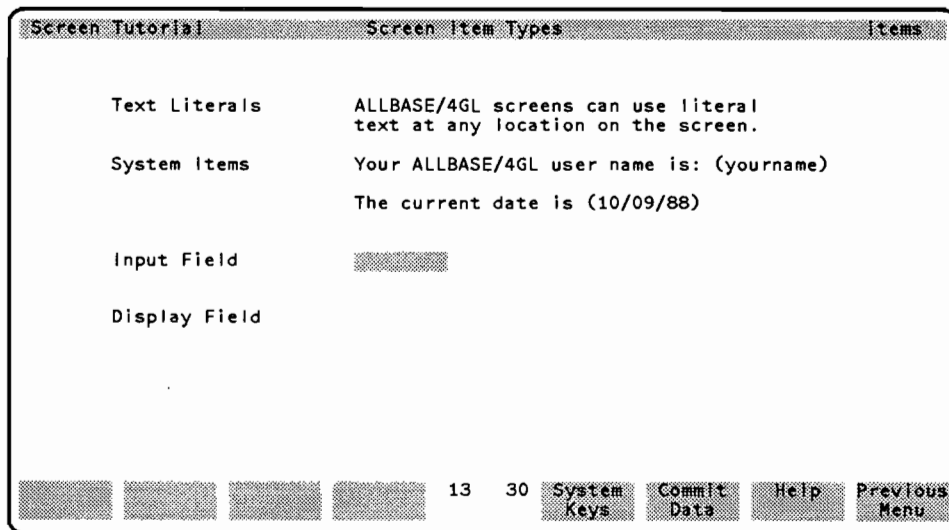
The main menu of the *tutorial* application shows the name of the actions called by the various menu options. Normally you wouldn't show this information on a menu, but in this case its inclusion allows you to examine the application source easily.

Data Screen Items

In the previous two lessons you have seen the following item types that data screens can use:

- Text literals
- System items
- Input fields
- Display fields

To review the various screen item types, select the *Screen Item Types* option on the main menu of the *tutorial* application. You should see this screen.



This screen is a simple demonstration of the items types that HP ALLBASE/4GL can display on a data screen.

The first items on this screen are simple literals, and system items. The actual system items are enclosed in parentheses (...) to distinguish them from the adjacent literal text. With the standard display highlighting, system items appear exactly the same as text literals.

Now make an entry in the input field, and then press **(Return)**. The data you enter is immediately displayed in the display field below. The input field is highlighted in inverse video, and the display field uses full bright normal video.

HP ALLBASE/4GL displays the message *Please press [Commit Data] to complete this screen* when you press **(Return)**. This is a standard system defined message that HP ALLBASE/4GL displays when the user completes the last input field on a screen.

Ignore this message throughout this lesson. Since the *tutorial* application doesn't have any data files, your data entry is discarded immediately if you press the **Commit Data** function key, or the **Previous Menu** function key. Pressing either function key takes you back to the main menu of the *tutorial* application.

Other Screen Features

Data screens can also use a window to display additional information or input fields to the user. The processing associated with fields on windows is exactly the same as the processing involved with fields on a data screen.

For the moment, this tutorial will only describe the processing involved with fields on data screens.

Data screens can also use a *scroll* area. You have already seen and used a scroll area on a screen on the dictionary file record layout details screen. You may want to look at this screen again now to refresh your memory. When you are in the developer, the menu path is:

Dictionary, Record Layouts, **Details**

As you committed the details of each successive record layout field on the screen, HP ALLBASE/4GL displayed the details of the existing fields in the screen scroll area. Later lessons in this guide show you how to create a scroll area on your application screens.

All screens can also use a feature called *special text*. Special text items are simply literals that have specified highlighting attributes. A typical use for special text items is to use the HP line drawing character set to create rules or boxes on the screen. The default HP ALLBASE/4GL sign-on screen uses special text to display the HP ALLBASE/4GL logo. Later in this course, you will use special text items on your application screens.

Data Screen Processing – Overview

HP ALLBASE/4GL displays a data screen as the result of a D- action from a menu, or a SCREEN command in a logic block. (HP ALLBASE/4GL can also display a menu as the result of a D- action or a SCREEN command. However, the system response to a menu is not the same.)

While HP ALLBASE/4GL is displaying a screen, screen processing activity occurs at two levels, referred to as the *screen* level, and the *field* level. Two basic user actions influence the processing of a screen, *field commit* actions, and *screen commit* actions. These processing levels and user actions are described below.

Screen Level Processing

Screen level processing controls:

- The initial display of a screen.
- The processing sequence of fields on the screen.
- What happens after the screen finishes.

Generally, HP ALLBASE/4GL processes fields on a screen in order from left to right, and top to bottom on the screen.

Figure 5-1 shows the essential elements of the screen level processing.

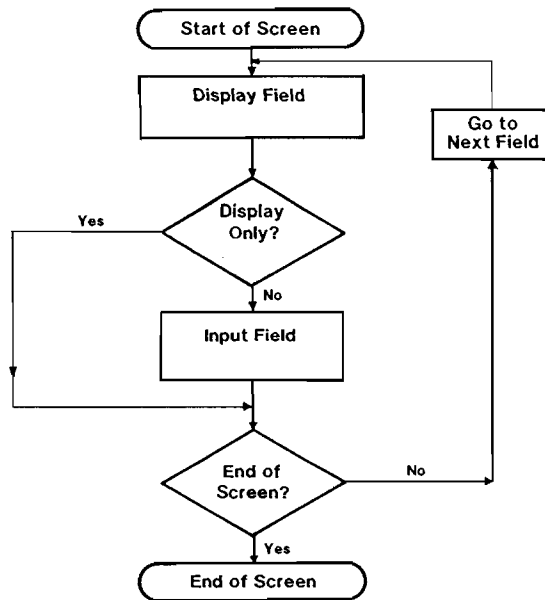


Figure 5-1 Screen Level Processing

Once HP ALLBASE/4GL starts processing a screen, processing continues on a field by field basis. When you move to a field, field processing takes over. When processing for a field finishes control passes back to screen level processing, which determines the next field to be processed.

After HP ALLBASE/4GL displays a screen, the screen remains active until the user initiates an action to terminate the screen. Typically this action is the *Commit Data* action initiated by pressing the **Commit Data** function key.

Screen Commit Actions

The user initiates a screen commit action by pressing the **Commit Data** function key. This function key calls an internal routine (the I-COMMIT action) that signifies the end of the current screen. Provided the user input up to this point has not generated an error, the screen terminates, and control returns to the entity that called the screen.

If the screen was called directly from a menu, control returns to the menu. If the screen was called by a SCREEN command in a logic block, control returns to the next command in the logic block.

Field Level Processing

The field level processing takes control when you move to a field on a screen. It consists of two distinct stages:

- Display processing.
- Input processing.

Display Processing

The display processing is identical for all screen field types, and occurs as soon as control passes to the field. Display processing consists of the following actions:

- Automatic data movement.
- Data formatting.
- Executing a function.
- Displaying the buffer contents.

In most respects the actions performed during display field processing are similar to the actions performed during input field processing. These terms are explained below in the context of the input processing for a screen input field.

Input Field Processing

Input field processing occurs after the user completes data entry to a field. After display processing occurs for an input field, HP ALLBASE/4GL enters a “wait state”, allowing the users to enter data. Input processing commences when HP ALLBASE/4GL receives a valid field commit action.

Field Commit Actions

A field commit action is the action that terminates user input for an input field. Any user action that moves the cursor off the current field to start data entry into a different field initiates a field commit action. (There are some other ways that fields can be committed, but they are not important for the moment.) The important point is that the user cannot leave a screen field and commence data entry in another field without committing the current field.

Input Processing Sequence

After receiving a field commit action, HP ALLBASE/4GL performs the following actions as part of the input field processing sequence.

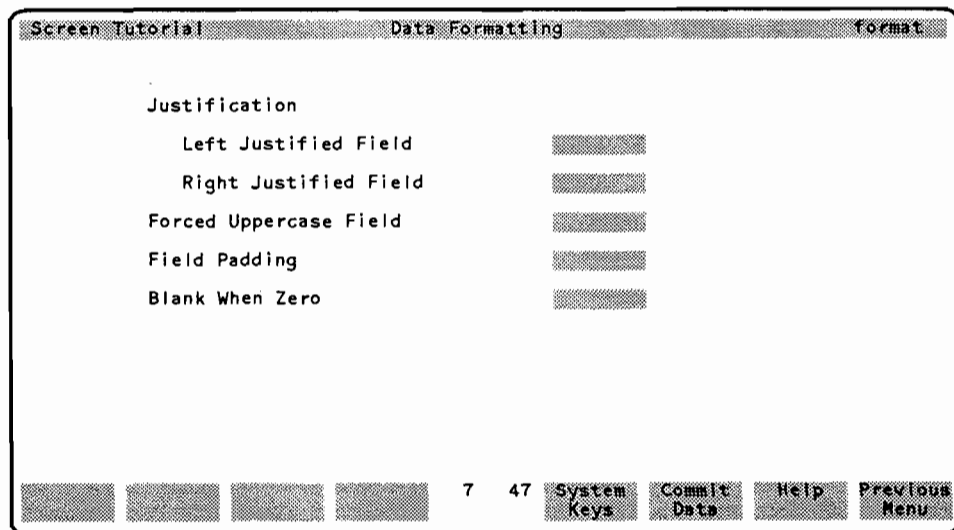
- Automatic data formatting.
- Automatic data validation.
- Function execution.
- Automatic data movement.

The *tutorial* application contains some simple demonstrations of these actions.

Data Formatting

HP ALLBASE/4GL provides a number of ways of formatting the data displayed on screens. The data formatting is identical for input fields and display fields, and in most cases also applies to fields printed on reports.

To see a demonstration of the various types of data formatting, select the application testing mode and then select the *Data Formatting* option on the main menu of the *tutorial* application. You should see a screen like this.



This screen demonstrates the automatic formatting that is available by specifying various options on the field specifications used to define the fields on the screen.

You have already seen these options on the field specifications screen when you defined the field specifications for the *training* application. None of these formatting options require any additional logic or programming effort.

Field Justification

Try entering any characters in the first input field. This is a typical *left justified* field. You have already used quite a few fields like this in the developer. Press **Return** after you have made an entry.

Now try entering some characters in the second field. You will see that this field is a right justified field. This particular field allows you to enter alphabetic characters. Normally, you would use right justified fields for numeric values only.

HP ALLBASE/4GL also allows you to specify that data is centered in a field, or that no justification at all is performed. These options are not shown on this screen.

Forced Uppercase

Enter some alphabetic characters in the third input field. Notice that they are automatically shifted to the uppercase form. Now try entering some numeric characters, and some punctuation characters. These characters are displayed normally.

This field has a type U edit code. This edit code specifies that the user input data is shifted to uppercase.

Field Padding

Enter one or two characters in the fourth field on the screen, and then press **Return**. Notice that all remaining spaces in the field are automatically filled with the character “*”, as this is the pad character for the field. You can use any acceptable character for the field as the pad character. Typically you would use a space character or zero as the pad character.

Blank When Zero

When the cursor is on the last field on the screen, simply press **Return** without making an entry in the field. Notice that the field remains blank.

Now try entering a numeric value in this field. This time, the remaining spaces in the field are filled with zeros when you press return.

This field is defined with a pad character of zero, and has the *Blank When Zero* field set to yes.

Data Validation

HP ALLBASE/4GL provides a number of automatic validation tests for user input data. The *tutorial* application provides a simple demonstration of the automatic validation tests. Select the *Data Validation* option from the main menu of the tutorial application. You should see this screen.

Screen Tutorial	Automatic Data Validation	Validate
Required Field	<input type="text"/>	
Number of Characters (field must be filled)	<input type="text"/>	
Edit Code Checking (alphabetic characters)	<input type="text"/>	
Decimal Places (two decimal places)	<input type="text"/>	
Table Validation (A, B, or C only, with system error message)	<input type="text"/>	
Range Validation (numeric value 3333 to 5555, with developer defined error message.)	<input type="text"/>	
	4 48	System Keys Commit Data Help Previous Menu

Once again, most of the data validation tests demonstrated on this screen use the various options available on the field specifications screen, or the screen field details screen. The last two validation tests shown on the screen introduce some new features. These are validation tables, validation ranges, and messages. You will use these features in your own application soon.

Required Fields

The first field on this screen is a required field.

Leave the first field on the screen blank, and press **Return**. HP ALLBASE/4GL displays a message telling you that the field is a required field, and it cannot be left blank. This is a standard system defined error message. Experiment with the **Tab** key and the other cursor control keys on your keyboard. You will find that you cannot move the cursor off the first field and enter data into a different field.

Make an entry in the field now, and press **Return** to move to the next field.

Number of Characters

The second field demonstrates two features. First, it is not a required field, so you can leave it blank. Try pressing **(Return)** now. The cursor moves to the third field without generating any error message. Press **(▲)** to return to the second field.

Now try entering one or two characters in the field. Don't fill it completely. Then press **(Return)**. HP ALLBASE/4GL displays an error message telling you that the entry you have made is less than the required length. Once again, this is a standard system defined error message.

Now fill the field by entering four characters. When you press **(Return)** the cursor moves to the third field without generating an error message.

This behavior is typical of all fields that have a value greater than one specified for the minimum entry length. If the field is not a required field, the user can leave it empty. If the user makes an entry into this field, it must be at least as long as the minimum entry specified for the field.

In this case, the minimum number of characters for the field is four, so you must fill the field completely.

Edit Code Checking

The third field on this screen is defined with a type A edit code. This edit code specifies that the field can only contain alphabetic characters.

Enter some numeric or other non-alphabetic characters. When you press **(Return)**, HP ALLBASE/4GL displays a system defined error message.

Decimal Places

This field shows how HP ALLBASE/4GL handles decimal numbers.

Enter the number 123 in this field. When you press **(Return)**, this value is displayed as 123.00. Now try entering the value 123.456. This time HP ALLBASE/4GL displays a system error message telling you that you have specified too many decimal places.

This field is defined as a numeric field (edit code N) with two decimal places.

Table Validation

This field introduces a new feature. All user input for this field is compared against a *validation table*. In this case, the table contains only the values A, B, and C.

Enter "a" in this field. HP ALLBASE/4GL displays a system defined error message when you press **(Return)**. Now try entering some other characters. You will find that the only acceptable characters are A, B, and C. HP ALLBASE/4GL is case

sensitive, and your entry must be uppercase to be acceptable. (You can leave this field blank as it is not a required field.)

The message displayed in this case is a system defined default error message. When you use a validation table, you can define your own message rather than use the system default message.

You can look at the validation table definition for this field if you like. To do this, exit from the tutorial application by returning to the main menu and then pressing the **Exit** function key, or the **Previous Menu** function key. Either of these actions take you to the developer main menu.

From the main menu, the menu path to access the validation tables screen is:

Dictionary, Validation Items, **Tables**

On the validation table screen, enter **example_table** in the Table Name field. The table can hold up to 51 different values, although this table only has three values.

Range Validation

The last field on this screen introduces a validation range. Validation ranges are similar to validation tables except that HP ALLBASE/4GL checks input data to determine if it lies within a specified range, rather than comparing the data against a table of discrete values.

Enter the value 1234 in this field. HP ALLBASE/4GL displays the message:

"This is a developer defined error message"

The range associated with this field is defined as being from 3333 to 5555. As 1234 lies outside this range, the error message associated with the range is displayed.

You can examine the definition of the validation range for this field by looking at validation ranges screen. The menu path to access this screen is:

Dictionary, Validation Items, **Ranges**

Enter **example_range** in the Range Name field on this screen. You will see the name **range_message** displayed in the message name field. This is the name of the message that HP ALLBASE/4GL displays when user input data fails the validation test. To look at this message definition, press the **Messages** function key on the validation ranges screen.

Enter the name **range_message** on the messages screen. HP ALLBASE/4GL then displays the definition of the message.

Later in this course, you will use a number of messages, and you will also use a validation range as part of your application.

Data Movement

HP ALLBASE/4GL provides extensive capabilities to move data to and from screen fields and other data fields during screen field processing.

A good understanding of the data movement facilities is central to your ability to use HP ALLBASE/4GL efficiently. A central part of this understanding is a knowledge of the data buffers that are associated with each screen field. The next few paragraphs introduce these data buffers.

Data Buffers

Every screen field has an internal screen field buffer associated with it. Through the internal screen field buffer, every screen field can be associated with up to four data movement fields. Figures 5-2 and 5-3 show the relationships between the screen field, the internal screen field buffer, and the data movement fields.

Screen Field Buffers

HP ALLBASE/4GL moves data from the internal screen field buffer to the screen display, and receives your input into the internal screen field buffer during the screen field processing sequence. The exact timing of this data movement is described in a later lesson.

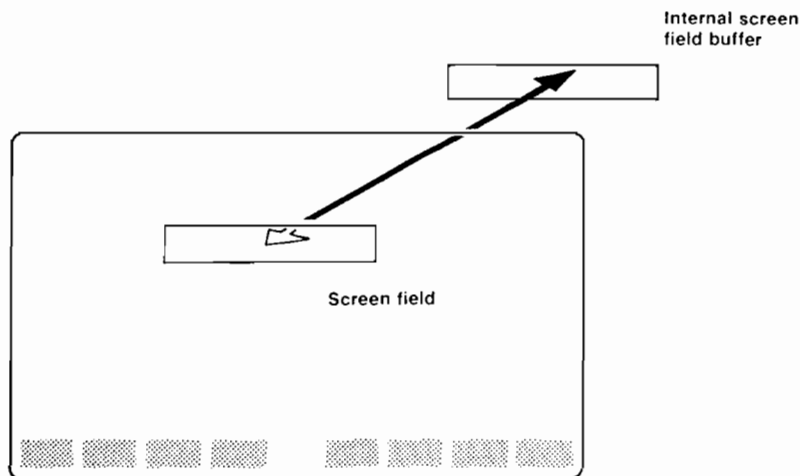


Figure 5-2 Screen Field Buffer

The application logic can access, and if required, change the data in the internal screen field buffer. HP ALLBASE/4GL also has a logic command (the SHOW command) that allows you to refresh the screen display using the contents of the internal screen field buffer.

Data Movement Fields

HP ALLBASE/4GL allows you to associate up to four other fields with a screen field as data movement fields. In the previous lesson, you specified some data movement fields for the fields on the *customer-scrn* data screen in the *training* application. Figure 5-3 shows the relationship between the screen display, the internal screen field buffer, and the data movement fields.

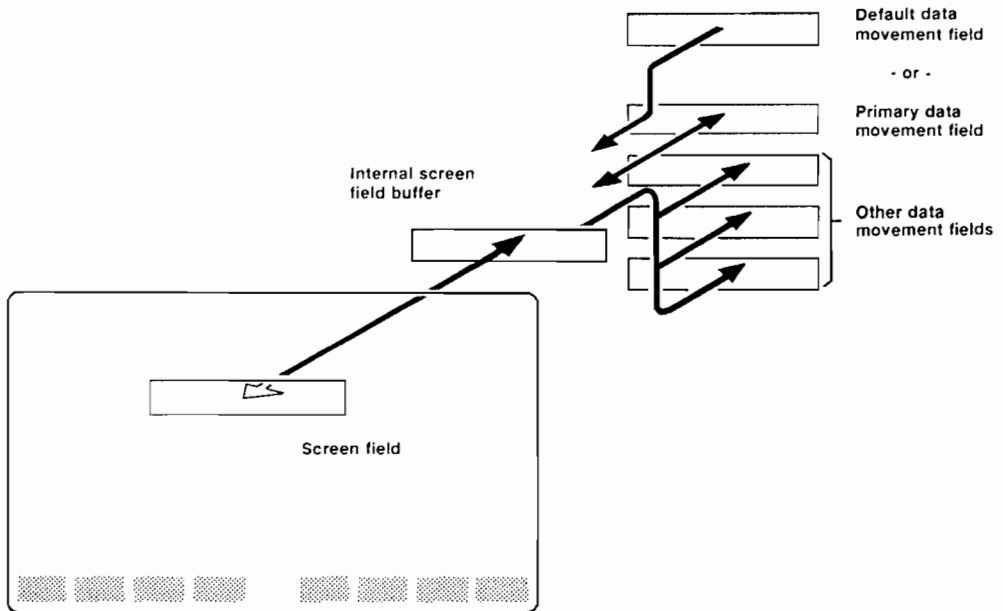


Figure 5-3 Data Movement Fields

You may want to refer back to this diagram while you are following the examples in the next few pages.

As you saw when you completed the screen field details screen for the fields on the *customer-scrn* data screen, the data movement fields are:

- The primary data movement field.
- The default data movement field.
- The other data movement fields.

These data movement fields are optional. A screen field does not need to have any data movement fields.

For any given screen field, the primary data movement field and the default data movement field are mutually exclusive. You can use one or the other, but not both.

You can use most types of data items as data movement fields for a screen field. In many cases, the data movement fields for a screen field are fields on file record buffers.

The Primary Data Movement Field

Any type of screen field can have a primary data movement field, provided it doesn't have a default data movement field. During the screen field display processing, HP ALLBASE/4GL moves data from the primary data movement field to the internal screen buffer.

For input fields, HP ALLBASE/4GL moves data from the internal screen field buffer to the primary data movement field during the input processing.

The Default Data Movement Field

Any type of screen field can have a default data movement field, provided it doesn't have a primary data movement field.

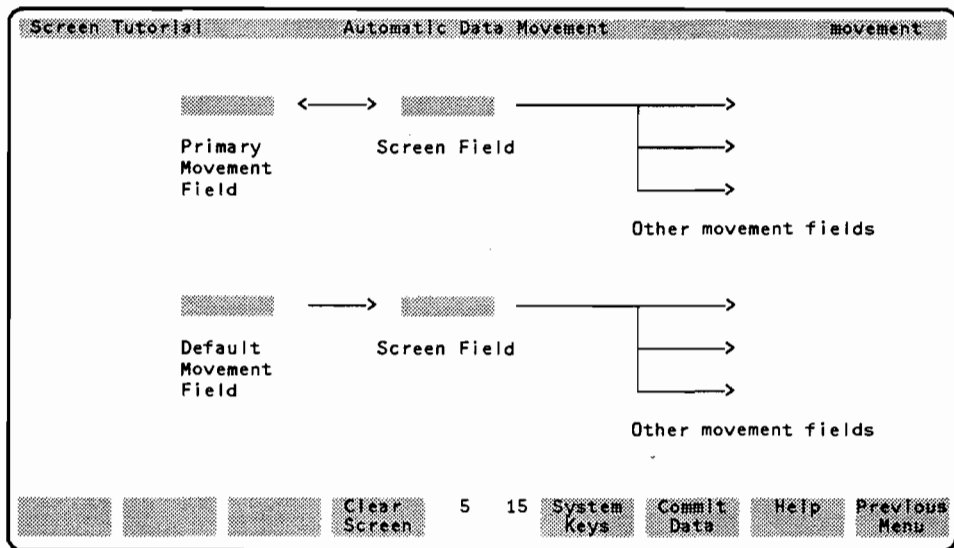
HP ALLBASE/4GL moves data from the default data movement field to the internal screen field buffer during the display processing. This movement only occurs if the internal screen field is blank.

Other Data Movement Fields

Only input fields on screens can have "other" data movement fields. During input field processing, HP ALLBASE/4GL moves data from the internal screen field buffer to the other data movement fields.

Automatic Data Movement

The *tutorial* application contains a screen that allows you to experiment with some aspects of the automatic data movement system. To access this screen, select the *Data Movement* option on the main menu of the tutorial application. You should see this screen.



This screen uses a number of screen fields to simulate the data movement fields for a screen field. In a normal application situation, the data movement fields for screen fields are usually file record buffer fields.

To use this screen, follow the instructions listed below. After you have tried the exercises suggested below, you may want to experiment with the screen to try out other combinations and situations.

If you have made any entries on this screen, press the **Clear Screen** function key. This clears the screen display, the internal screen field buffers for all fields, and then positions the cursor on the first field for the screen.

You can press the **Clear Screen** function key at any time to clear the existing data from the screen fields and the internal screen field buffers. (This function key performs exactly the same action as the **Clear Screen** function key in the system keys function key set.)

Primary Data Movement

Start off by entering the characters **abc** in the field labelled *Primary Movement Field*. When you press **(Return)**, the same entry is echoed in the second field. (That is, the field labelled *Screen Field*.)

Now enter the characters **xyz** in the *Screen Field* field. When you press **(Return)**, the entry **xyz** is echoed in the *Primary Movement Field* and all three *Other Movement Fields*. This simple example demonstrates a number of important features of the data movement system.

- HP ALLBASE/4GL moves data from the primary data movement field to a screen field, and displays the data during the display processing for an input field.
- After you complete your input, HP ALLBASE/4GL moves data back to the primary data movement field, and also moves data to up to three “other” data movement fields.

You may want to try this sequence of actions again with different combinations of input data.

Default Data Movement

The second set of fields on this screen demonstrate the behavior of the default data movement system. Press the **Clear Screen** function key, and then position the cursor on the field labelled *Default Movement Field*. Then try the following actions.

Enter **abc** and then press **(Return)**. HP ALLBASE/4GL echoes the characters **abc** in the field labelled *Screen Field* and positions the cursor on this field. Now type **xyz** and press **(Return)**. HP ALLBASE/4GL echoes the entry **xyz** in the three other data movement fields, but your entry is **not** echoed in the *Default Movement Field*.

Now go back to the *Default Movement Field*. Don't press the **Clear Screen** function key because you need some data in the next screen field to complete this demonstration. Type the characters **def** in the *Default Movement Field* and then press **(Return)**. This time you will see that HP ALLBASE/4GL does **not** move the contents of the default movement field to the screen field.

Complete the exercise by typing **abc** in the screen field and pressing **(Return)**. Once again, HP ALLBASE/4GL echoes your entry in the three other data movement fields, but does not echo your entry in the default data movement field.

This exercise demonstrates a number of features about default data movement fields.

- HP ALLBASE/4GL moves data from the default data movement field to a screen field if (and only if) the screen field is empty. If the screen field already contains data, the default data movement does not occur.
- HP ALLBASE/4GL does not move data from the screen field back to the default data movement field.

Once again, you may want to try this exercise again with some different sets of data.

Screen Field Functions

You can associate a logic function with each screen field. The function can be executed at the following times:

- During the field display processing,
- During field input processing (for input fields only),
- During the processing that occurs while a `SHOW` command is being executed.

You have already used the screen field details to associate a function with a field on the *customer_scrn* screen.

It's important to realize that you can only associate one function with each screen field, but the function can be called at three distinctly different times.

The SHOW Command

So far, this tutorial has told you the truth, but not the whole truth. There is one more important command involved with screen processing. This command is the `SHOW` command. In fact, the screen that demonstrates data movement uses the `SHOW` command.

The `SHOW` command redisplayes the contents of the internal screen buffer for a screen field, or a range of screen fields.

In some circumstances, the application logic may change the contents of the internal buffer for a screen field. Any change to the contents of the internal field buffer for a screen field other than the field that is currently being processed is **not** automatically reflected on the screen.

Exactly this situation occurs in the data movement demonstration in the tutorial application. When you make an entry in the second field on this screen,

HP ALLBASE/4GL moves the input data back to the primary data movement field for the screen field. The primary data movement field for this screen field is the internal buffer for the field labelled *Primary Movement Field*. Why then, did your input appear in the screen display for the field?

Screen field number three on this screen has a prior function associated with it, and this function contains a SHOW command. When you enter data in the second field, the following sequence of events occurs:

1. When you press the **Return** key, HP ALLBASE/4GL recognizes this as a field commit action and commences the input field processing for field number two. During this processing, HP ALLBASE/4GL moves your input data to the primary data movement field for the screen field, and to the other data movement fields.
2. At the end of the input processing for field number two, control passes to field number three.
3. HP ALLBASE/4GL commences the input processing for field number three. During this processing HP ALLBASE/4GL executes the *prior* function for the field. This function contains a SHOW command. (In fact, the SHOW command is the only command in the function.) The SHOW command redisplay the current internal field buffer contents for all fields on the screen. At this stage, your entry is redisplayed in field number one.
4. Processing then continues for the remaining fields on the screen.

The SHOW command accepts a number of arguments. These arguments allow you to specify that the command only applies to one field, a range of fields, or all fields on the screen.

A second argument (the *REFRESH argument) specifies that the contents of the internal screen buffer itself is refreshed from the primary or default data movement field **before** the buffer contents are displayed on the screen.

Figure 5-4 shows the data movement that occurs for a field when it is displayed by the SHOW command.

SHOW Command Processing

The SHOW command also operates on a screen level and a field level. At the screen level, the SHOW command processes fields in the defined sequence for the screen. Processing continues until all fields on the screen, or all fields in the defined range for the SHOW command have been processed.

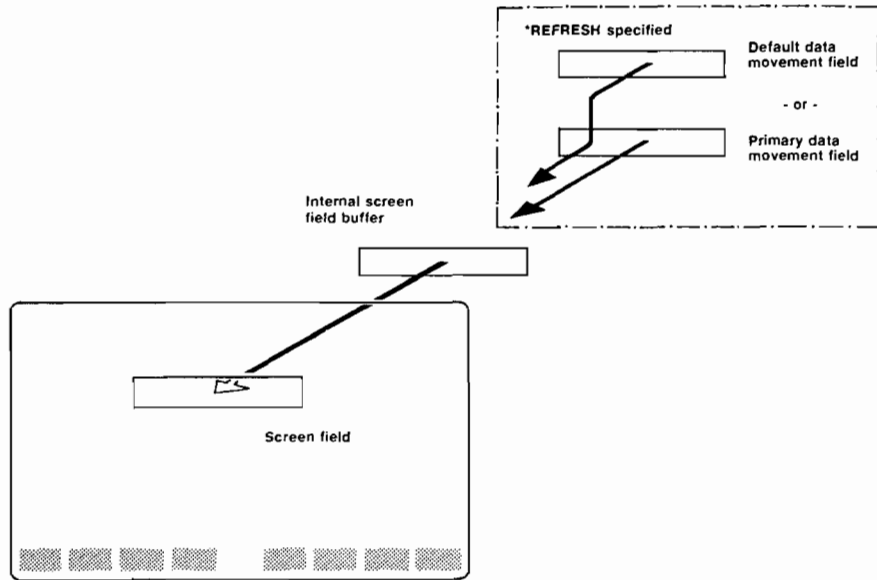


Figure 5-4 SHOW Command Data Movement

At the field level, the SHOW command processing is similar to the field display logic. During the SHOW command processing for a field, HP ALLBASE/4GL can perform the following actions:

- Moving data to the internal screen buffer.
- Formatting data.
- Executing a function.

You will use the SHOW command in your own application soon.

Summary

This tutorial has introduced you to some of the important concepts involved with screens. These are summarized below.

Screen Fundamentals

- HP ALLBASE/4GL uses three types of screens; menus, data screens and windows.
- HP ALLBASE/4GL displays a screen as the result of a SCREEN command in logic, or a D- action from a menu.
- When HP ALLBASE/4GL displays a data screen, the screen and the logic associated with the screen control all activity in the application until the screen terminates.
- When HP ALLBASE/4GL displays a menu, all current activity in the application is terminated.

Data Screen Processing

- HP ALLBASE/4GL processes data screens at two “levels”: the screen level, and the field level.
- Data screen processing is controlled by field commit actions, and screen commit actions, initiated by the user.
- The standard order of processing for fields on data screens is left to right, and top to bottom. This standard order can be changed by the application logic if needed.
- The initial processing of all fields (input and display) is identical.
- A logic function can be associated with a screen field. This function can be executed at three different times.

Field Display Processing

- The main actions performed during the display processing for a screen field are automatic data movement, justification, padding, and executing a function.

Input Field Processing

- HP ALLBASE/4GL enters a “wait” state at the end of the display processing for a input field. No further action occurs until the user initiates a field commit action. A field commit action is any action that moves the cursor off the current field.
- During input field processing, HP ALLBASE/4GL performs automatic data validation, executes a function, and automatically moves data.

Screen Field Data Buffers

- Each screen field has an internal screen field data buffer, and can have a number of data movement fields associated with it.

The SHOW Command

- The SHOW command allows you to redisplay the contents of the internal screen buffer on the screen display.
- SHOW command processing is similar to the display processing for a screen field. The SHOW command processing can involve data movement, data formatting, and executing a function.

This tutorial has introduced quite a few new terms and concepts. Most likely, the details of HP ALLBASE/4GL screen processing look a bit formidable to you at the moment. Don't worry. In the next two lessons you will develop some logic so you can run your own application. You can then see the screen processing logic at work in your application.

The next chapter contains a more detailed explanation of the screen processing system. You may want to refer back to this lesson and the *tutorial* application while you are working through the next chapter.

Contents



Chapter 6: Logic

What Are Processes and Functions?	6 - 1
Logic Blocks	6 - 1

Lesson 7 – Developing a Process

Objectives	6 - 3
The Customer Process	6 - 3
Defining the Process Header	6 - 3
Defining the Process Details	6 - 4
The Process	6 - 5
Generating a Logic Block	6 - 9
Summary	6 - 10

Lesson 8 – Developing a Function

Objectives	6 - 11
Functions	6 - 11
Defining the Function Header	6 - 11
Defining the Function Details	6 - 13
Generating the Function	6 - 16
Summary	6 - 16

Intentionally Blank

6 Logic

So far you have defined a number of field specifications and used them to create a data file and a data screen. The basic data requirements for this part of the application are now complete. Next you need to define some simple logic to get the application running.

HP ALLBASE/4GL uses logic constructs called processes and functions. The lessons in this chapter describe the techniques to develop a process and a function so you can get your application running.

What Are Processes and Functions?

A process is similar to a program in a conventional language. When a process starts, HP ALLBASE/4GL initializes the environment to a known state. It closes all data files, clears all data buffers, and terminates any current non-background process, screen, function, report or decision table.

A function is similar to a conventional subroutine. A function can be called from many places without automatically affecting the current environment. When a function is complete, the application returns to where the function was called from.

Logic Blocks

Processes and functions are collectively referred to as logic blocks. They both share the same constructs and generally the same commands. They each have a maximum of 30 lines of commands, and are created by entering the details of each line into formatted windows. Using windows means that you don't need to remember the exact syntax for every command.

The logic blocks you are about to define specify how the application displays the *customer_scrn* screen you just defined, and how the data entered on the screen is handled.

These logic blocks allow you to add data to the customer file, or modify existing data in the customer file. The logic blocks are:

- A process, called from the menu, to control the screen and the manipulation of the data file.

- A function, called from a data screen, to check for the existence of the record requested by the user.

The logic blocks have been kept as simple as possible so you can get a basic file manipulation system running quickly. In later lessons, you will enhance this simple application, and discover other ways to perform these file manipulation tasks.

Lesson 7 – Developing a Process

Objectives

In this lesson you will develop a process. To develop the process you will:

1. Define the process header.
2. Define the process details.
3. Generate the process.

The Customer Process

The process you are about to develop calls a data screen. When the user completes the screen, the same process writes the data entered by the user to the customer file. The process then loops back to the beginning, to start again.

Many processes call data screens. This is a standard method of using a menu item to call a process, which in turn displays a data screen and then manipulates data in a file.

The user must press the **Previous Menu** function key to exit from the process. After pressing the **Previous Menu** function key the user is taken back to the menu that called the process.

Defining the Process Header

Defining a process involves two steps. First you must define a process header, and then define the steps that make up the process on the process details screen.

Menu Path

Logic, Processes, **Header**

Description

This screen enables you to enter a brief description of the process for documentation purposes.

Developer		Process Header		process-header					
Process Name		customer-proc							
Description									
customer-proc									
AUTHOR: rmjones									
This process is called from the Customer Details action									
item on the main menu of the training application.									
Last Modification: Date Time									
Command	Process	Function	SQL Blk	21	22	System	Commit	Help	Previous
Help	Detail	Menu	Menu			Keys	Data		Menu

Entering the Field Values

Process Name. Enter `customer-proc`

This is the name of the process within the application.

Description. Enter a suitable description.

This brief description of the process is used for documentation purposes.

Press the **Commit Data** function key to create the process header. Once you've done this, go to the *Process Details* screen to create the process logic block.

Defining the Process Details

Now you've created the process header you can define the details of the process.

Menu Path

Logic, Processes, Details

Description

This screen allows you to create the command lines that make up the process logic block. The screen is divided into two distinct regions.

The upper region displays 10 of the 30 lines allowed in the logic block. This range is scrolled up and down to display the current line in the logic block as you work through it.

The lower region allows you to enter commands. Three fields are always displayed in the lower region. These are the current step number, action, and command name. Below this is a window where the remainder of the command line is entered or displayed.

HP ALLBASE/4GL displays special windows for commands that have a more complex syntax. This special window steps you through the syntax of the particular command. HP ALLBASE/4GL displays the window after you enter a command name in the *Command* field.

The simpler commands use a free format window for you to enter the rest of the command. Some commands don't have options, so no window is displayed.

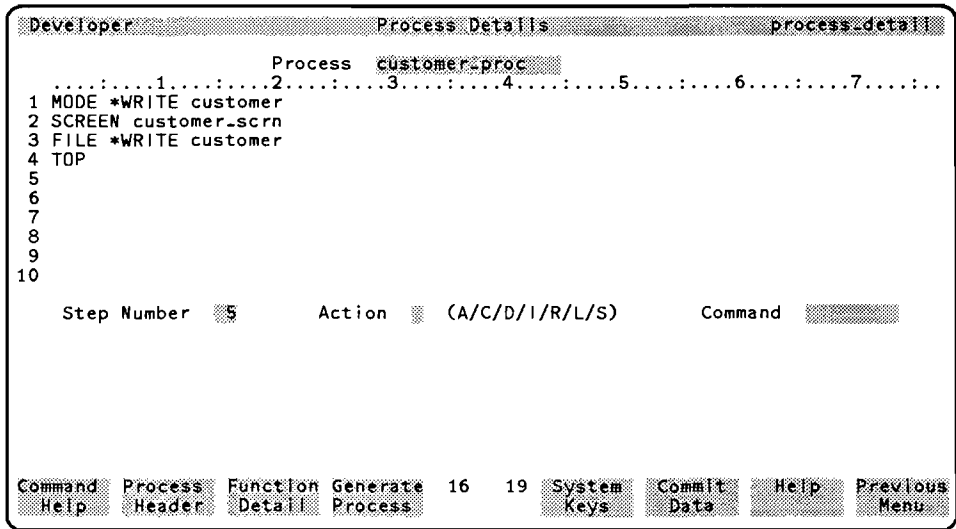
You will see examples of each of these methods as you enter the steps for this process.

The Process

This is the process you are going to create.

- 1 **MODE** *WRITE customer
- 2 **SCREEN** customer_scrn
- 3 **FILE** *WRITE customer
- 4 **TOP**

This process is quite a simple one. In the first line the **MODE** command specifies how the application can access data files. In this case the application has write access to the data file called *customer*. The **SCREEN** command in the second line displays the *customer_scrn* screen. The **FILE** command in the third line writes data from the file record buffer to the *customer* file. The **TOP** command in the last line causes the process to execute again from step number 1. The user can exit from the process by pressing the **Previous Menu** function key at any time.



Entering the Field Values

You can now enter the data required to define the process. Each entry is detailed below.

Process Name. Enter `customer_proc`

This will be the default name in this field if you have just defined the header for the process.

Open Window Commands

The first two commands use the open window method for you to enter the remainder of the step.

Step Number. Accept the default value of 1

This is the number of the step in the logic block that you are defining. Notice that it is automatically incremented as you create each line of the logic block. If the current step is not displayed in the upper region of the screen, then the logic block is scrolled to ensure that the step is displayed.

It is not necessary for step numbers to be contiguous. You can leave blank lines within a logic block.

Action. Accept the default action of **A**

This is an action code much like the one you used to create the record layout details. It defaults to **A** indicating *add* for a new step, or **C** indicating *change* for an existing step.

The other valid actions available are **D** to *delete* a step, **I** to *insert* a step, **R** to *replace* a step, **L** to *list* the whole logic block, and **S** to *swap* the display pages if your logic block is longer than the ten lines displayed in the window.

Command. Enter **MODE**

When you enter this command name, HP ALLBASE/4GL presents a free form data entry area at the base of the screen so you can enter the remainder of the command.

Notice that you don't have to enter upper-case characters, HP ALLBASE/4GL automatically upshifts a lower-case entry.

The **MODE** command sets the usage mode for a file while a process is active. By default, HP ALLBASE/4GL opens files for read access only. Refer to the *HP ALLBASE/4GL Developer Reference Manual* for more information about this command and any others you encounter in this manual.

You can obtain a brief description of a command by pressing the **Command Help** function key on the process (or function) details screens. This function key displays several further sets of function keys that allow you to select the help screen you want.

To Complete the Command. Enter ***WRITE customer**

This specifies that you want to write to the *customer* file. Note that you don't need to enter the word *write* in uppercase letters.

Press **Commit Data**

This commits the step. HP ALLBASE/4GL clears the window and shows the new command line in the display region of the screen. The cursor returns to the *Step Number* input field so you can proceed to enter the next command. The step number is automatically incremented as you build the logic block.

Step Number. Accept the default value of **2**

Action. Accept the default action of **A**

Command. Enter **SCREEN**

To Complete the Command. Enter **customer_scrn**

This command displays the *customer_scrn* screen. Control returns to the process when the end user terminates the screen by pressing the **Commit Data** function key on this screen.

Press **Commit Data**

This displays the second command line in the display region of the screen. It also clears the command window and returns the cursor to the *Step Number* input field.

A Special Window Command

The next step uses a command sensitive window to enter the remainder of the command line.

Step Number. Accept the default value of 3

Action. Accept the default action of A

Command. Enter FILE

The file command displays a special window to prompt for further details. The window appears after you enter the command name *FILE*.

File Operation. Enter *WRITE

Try entering an invalid file operation. HP ALLBASE/4GL displays a message informing you that the operation is invalid. The cursor remains on the field and you must correct the entry before you can move on. This shows you how the command sensitive window helps you enter the correct command structure. Also note that you don't need to enter the asterisk before the command name. HP ALLBASE/4GL enters it for you.

Since the file used in this process has unique primary key values, it is possible to use the *WRITE option to both add or modify a record in the file. Refer to the *HP ALLBASE/4GL Developer Reference Manual* for further information.

***NOLOCK.** Accept the default value of N

The entry in this field specifies whether this command imposes a lock on the file as it is accessed. The *N* option specifies that normal locking applies. (That is, the *NOLOCK* option is not used.) Refer to the *HP ALLBASE/4GL Developer Reference Manual* for more information about file locking.

File ID[.Record]. Enter customer

This is the name of the file that is used for this operation. You can append a record specifier to the file name. This is only necessary if you want to access a file using a record layout other than the default record layout.

The remaining fields on the FILE command window are optional. You will use some of these fields later in this training course. For the moment, leave them blank.

Press **Commit Data**

When you press this key, HP ALLBASE/4GL clears the window and displays the command on the third line of the process logic block. Notice that the fields for which you made no entry are omitted from the command.

A No Window Command

The next command has no options, so it doesn't use a window.

Step Number. Accept the default value of 4

Action. Accept the default action of A

Command. Enter TOP

The TOP command passes control back to the first line of the process to execute it again. HP ALLBASE/4GL does not display a window for the TOP command as it does not have any options.

Press **Commit Data**

The new command line is displayed at the fourth line and the cursor returns to the *Step Number* field.

Generating a Logic Block

Like data screens and record layouts, processes and functions also require generation. Generating the process checks for the existence of the data names, records, and files you have named, and resolves pointers to these items. Press the **Generate Process** function key to generate the *customer_proc* process.

If you entered the process correctly you won't receive any error messages. If you have made an error, you will receive a message listing the step number and character position value that was found to be in error, and a brief description of the error. The error message is displayed for about 20 seconds. HP ALLBASE/4GL then returns you to the details screen. You can press the **(Stop)** key to freeze the display.

The *step number* column refers to the step number in the logic block, the *character position value* column refers to the ruler line at the top of the display region of the logic block screen. By using the ruler line, you can determine the horizontal location of the incorrect value.

Once you have determined the cause of the error, you can change the step by entering its step number and accepting the *C* action that is displayed by default. This displays the command line at that step in the appropriate window, and you can correct the error. (If the error is in the Command field, press **(Shift) + (Tab)** to move to this field from the window.) Once you have corrected the entry, press the **Commit Data** function key, and generate the logic block again.

When the process has been successfully generated, you can continue on to develop a function for this application.

Summary

In this lesson you have created a process.

A process is similar to a program in a conventional programming system. It terminates any current activity in the application, and initializes the environment to a known state.

The process you have created displays the *customer_scrn* data screen and updates the *customer* data file when the user terminates the screen.

You will see this scenario of a process driving a data screen, which in turn calls a function, repeated many times as you develop applications.

There are three steps involved in developing a process. They are:

1. Define the process header.
2. Define the process details.
3. Generate the process.

Lesson 8 – Developing a Function

Objectives

In this lesson you will develop a function. To develop this function you will:

1. Define a function header.
2. Define the function details.
3. Generate the function.

Functions

Functions are similar to subroutines in a conventional programming language. When a function ends, control returns to the item that called the function. Functions use the same constructs, and generally the same commands, as processes.

The function you are about to create is executed by the screen logic as an *after* function for a field on the screen. This means that it is executed **after** the user enters data to the screen field.

Functions can also be called from a function key, through a VISIT command in any logic block, as well as a number of other methods. It's important to remember that a function does not alter the application environment when it's called.

Developing a function is similar to developing a process. You must define the header first, and then the details. A function must be generated before the application can use it.

Every logic command except the MODE command can be used in a function. The MODE command is only permitted in a process.

Defining the Function Header

The function you are about to create is executed on completion of data entry for the *customer_no* field on the *customer_scrn* data screen. The function reads the customer file using the value just entered as the key for the file search, and then displays the data from the file if a matching record is found.

Menu Path

Logic, Functions, Header

Description

This screen enables you to enter a brief description of the function for documentation purposes.

Developer		Function Header		function_header					
Function Name	customer_func								
Description	customer_func								
AUTHOR:	rmjones								
	This function is called after data is entered in the customer number field on the customer data screen.								
Last Modification:	Date	Time							
Command Help	Function Detail	Process Menu	SQL BIK Menu	21	22	System Keys	Commit Data	Help	Previous Menu

Entering the Field Values

Function Name. Enter `customer_func`

This is the name of the function within the application.

Description. Enter a suitable description.

Press the `Commit Data` function key to create the function header. Now go to the function detail screen to create the logic block commands.

Defining the Function Details

Having created the function header you can now define the function details.

Menu Path

Logic, Function, **Details**

Description

This screen is identical in appearance and operation to the process details screen. The only difference is that you are creating a function, not a process. It uses the same step numbers, actions, and commands as the process details screen and it behaves in a similar manner.

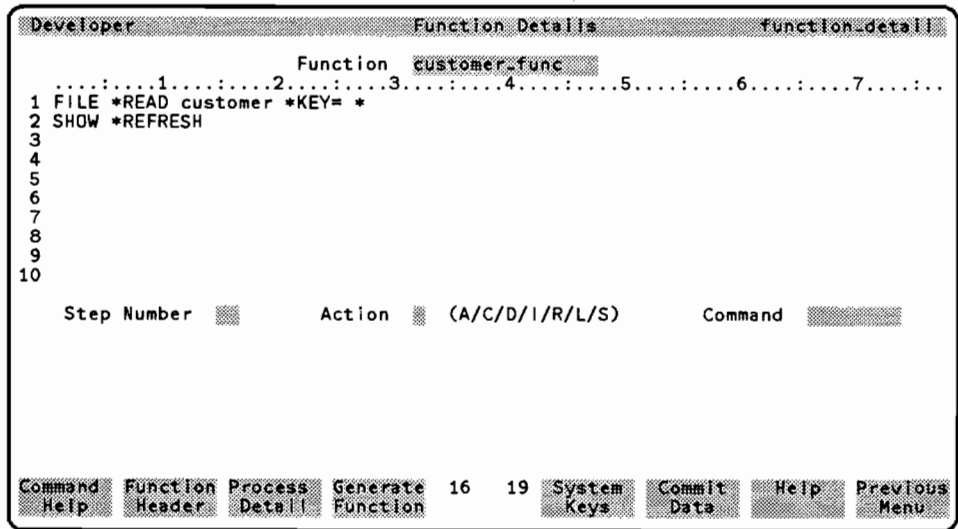
This is the function that you will create:

- 1 **FILE *READ** customer *KEY= *
- 2 **SHOW *REFRESH**

When the user enters data into the *customer_no* field on the screen and presses **Return**, the **FILE** command searches the *customer* data file for a record that has a key that matches the data entered by the user. If a matching record is found, the **SHOW** command displays the data contained in the file record on the screen.

If no match is found, the **SHOW** command displays the entered data again, and the user can continue to complete the screen.

The screen image shows you what the screen will look like as you enter the function.



Entering the Field Values

Function Name. Enter `customer_func`

This will be the default value if you have just created the header for this function.

Step Number. Accept the default value of 1

Action. Accept the default action of **A**

Command. Enter `FILE`

Again you will see the FILE window appear. The window is exactly the same as the one you used when you were defining the process.

File Operation. Enter `*READ`

This time you want to read a record from the file. You can enter this command as `READ` without the asterisk.

***NOLOCK.** Accept the default value of **N**

File ID[.Record]. Enter `customer`

Again, this is a reference to the default record layout on the customer file.

***INDEX=.** Leave blank.

This field allows you to specify which index is used to access the file. Leaving it blank means that the file is accessed using the index specified by the current value in the communication area field *INDEXNO. HP ALLBASE/4GL initializes the value in *INDEXNO to 1 when an application starts, assuming that all file access is to be via the primary key. Refer to the *HP ALLBASE/4GL Developer Reference Manual* for more information.

***KEY=.** Enter *

The *KEY= * clause specifies the value to be moved to the key field of the file buffer before the file read is performed.

The * refers to the internal buffer of the current screen field. As this function is called from the *customer_no* field, the * refers to the contents of the *customer_no* field on the data screen.

If you don't specify a key, the file read takes place using the current contents of the key field in the file buffer.

Press **Commit Data**

When you press this key, HP ALLBASE/4GL clears the window and displays the entered command at the first line of the function. You will see that the *KEY= * clause is included in the command line.

Step Number. Accept the default value of 2

Action. Accept the default action of A

Command. Enter SHOW

To Complete the Command

Enter *REFRESH

Notice that you can enter either lowercase or uppercase characters for the word REFRESH. This command redisplay the fields on the screen. The *REFRESH clause specifies that data is moved from the data movement fields to the internal screen buffers before the fields are redisplayed. All the fields on the screen have been created with primary movement fields from the customer file. This means that the current contents of the file buffer are displayed by this command. Refer to the *HP ALLBASE/4GL Developer Reference Manual* for further information about this command.

Press **Commit Data**

HP ALLBASE/4GL displays the new command line and clears the window. This function is now complete.

Generating the Function

The function must be generated before it can be used in the application. The procedure for generating a function is identical to that used to generate a process.

Press the **Generate Function** function key. HP ALLBASE/4GL displays a message to indicate that the function is being generated. The function will either generate correctly, or a listing of any errors encountered will be displayed.

If any errors are detected, you can correct them in the same way described for the process logic block error correction. Refer to *Generating a Logic Block*.

Once you have successfully generated the function, you can start testing the application. The next lesson explains the procedures for testing your application.

Summary

In this lesson you created a function.

The function is executed as an after function for the *customer_no* screen field, and performs a file lookup after the user enters data in the field.

There are three steps involved in developing a function. They are:

1. Define the function header.
2. Define the function details.
3. Generate the function.

Contents



Chapter 7: Running Your Application

Lesson 9 – Running Your Application

Objectives	7 – 2
Application Testing	7 – 2
Any Problems?	7 – 2
Testing the Application	7 – 2
Trace Mode	7 – 4
Handling Problems	7 – 6
Items Not Found	7 – 6
The Initial Action	7 – 8

Screen Processing Review

Full Data Screen Processing	7 – 10
Display Field Processing	7 – 12
Input Field Processing	7 – 14
Summary	7 – 17
Data Movement	7 – 18

Lesson 10 – Developer Utilities

Objectives	7 – 19
The Utilities Menu	7 – 19
Displaying Catalog	7 – 20
Copying Catalog Items	7 – 21
Deleting Catalog Items	7 – 23
Printing Catalog Items	7 – 24
Generating Items	7 – 26
Summary	7 – 27

Intentionally Blank

7 Running Your Application

This chapter introduces you to a number of features.

More importantly, it shows you how to run the application you have just developed. It also contains a detailed explanation of the HP ALLBASE/4GL screen processing logic.

This chapter also introduces a number of utility functions that are available in the developer.

Lesson 9 – Running Your Application

Objectives

In this lesson you will use the *Application Testing* mode to run the application you have just developed.

You will also use the *Trace Mode* facility to examine the step by step operation of the application.

Application Testing

You have now completed the development stages of a small, but workable application. The next step is to test it.

Any Problems?

Sometimes, things can go wrong when you test your application. The application may not behave as expected or it may even abort and return you to the developer main menu.

The remainder of this lesson assumes that nothing does go wrong. If things don't work as you expected, refer to *Handling Problems*, page 7 – 6. It covers the most common problems that may occur. If any problems you encounter are not explicitly described, refer to the *HP ALLBASE/4GL Developer Reference Manual* for further information.

Testing the Application

As a developer, you can test your application without leaving the *developer* application. The *Application Testing* option on the developer main menu runs your application as it appears to the end user. When you test an application using the application testing option, you have access to some facilities that are not available to end users. These facilities can help you identify any problems.

To test your application, return to the developer main menu and perform the following actions:

Action	Notes
Activate <i>Application Testing</i>	This loads a new HP ALLBASE/4GL environment and executes the initial action for the application. The initial action is either a <i>menu</i> or <i>process</i> . This is defined in the administrator definition for the application. The main menu for the application should appear.
Activate <i>Customer Details</i>	This executes the <i>customer_proc</i> process. This process in turn displays the <i>customer_scrn</i> screen.
Now you can enter data into the screen as follows:	
Enter abcdef	HP ALLBASE/4GL responds with a message to tell you that the data is not numeric. This is a standard edit validation performed by HP ALLBASE/4GL.
Enter 1	Before moving the cursor to the next field HP ALLBASE/4GL executes the <i>customer_func</i> function associated with this field. This function is an <i>after</i> function because it is executed <i>after</i> the field is committed. The function reads the customer file and searches for a record with a primary key that matches the customer number just entered. This record does not exist, so HP ALLBASE/4GL displays a warning message.
Press Return	The <i>Name</i> field has been defined as a required field. You cannot move to the next field if the <i>Name</i> field is blank. HP ALLBASE/4GL displays a message informing you that the field cannot be left blank.
Enter test name	The value for the customer name is accepted. The cursor moves to the first line of the address area when you press Return .
Press Shift and Tab . Then press Return	This is also a required field. However, you can leave it blank and move back to the previous field on the screen.
Press Tab	This moves the cursor back to the first line of the address field.
Enter 20 Test St	This value is accepted and the cursor moves to the next line of the address field after you press Return .
Enter Newtown	This is acceptable, and the cursor moves down one more line when you press Return .

Press Return	This is not a required field, so you can leave it blank by pressing either Return or Tab .
Enter NY	The cursor moves to the zip code field.
Enter 1234	This entry initiates a system message stating that the minimum number of characters required for the field have not been entered. When you defined this field you specified a minimum entry length of 5 digits.
Enter 12345	This value is accepted and you are prompted to press the Commit Data function key to complete the screen.
Press Commit Data	This terminates the screen and returns control to the Process driving the screen. The process writes the new record to the file and then returns to the start, displaying the screen again.

Trace Mode

When you first test your application it may not appear that all of the activity you have defined has actually taken place. Trace mode enables you to see it all.

HP ALLBASE/4GL has a trace mode to help you test your applications. It lets you see each step of your application as you test it, by displaying a line-by-line description of the logic being executed as the application runs.

To use trace mode, press the **Previous Menu** function key to return to the main menu of the training application. Then follow the next steps:

Action	Notes
Press System Keys	A new function key set is displayed.
Press More Keys	Another set of function keys is displayed.
Press Trace Mode	This key turns the trace mode on. An asterisk appears in the function key label indicating that trace mode is now on. When you want to turn trace mode off again, simply press the function key again.

With trace mode on, go through the procedure to create a customer file entry again. This time you will see every logic block line echoed at the base of the screen as the application runs.

Activate the *Customer Details* menu item. The first line of the *customer_proc* process is displayed in the following format:

```
P-customer_proc: 1:MODE:MODE *WRITE customer
```

The colons (:) separate the four parts of a trace statement. The four parts of a trace statement are:

1. Logic block type and name. In trace mode, action prefixes are recognized, so P- indicates a process followed by the name of the process *customer_proc*.
2. Step number within the *customer_proc* logic block. In the example above this is step number 1.
3. Command name. In this case it is the MODE command.
4. The full command line. In this case MODE *WRITE customer.

When the line has been executed, an end step message is displayed.

```
P-customer_proc: 1:END STEP
```

The next step, the SCREEN command, is displayed and executed. The *customer_scrn* screen is displayed.

Enter a customer number.

After you press **(Return)**, the first line of the *customer_func* function is displayed.

```
F-customer_func: 1:FILE:FILE *READ customer *KEY= *
```

This shows that HP ALLBASE/4GL is executing the function associated with this field after you entered data in the field. When the final step from the function finishes the cursor moves to the next field and you can continue with data entry.

If you enter the number of a *customer_rcrd* record that already exists, the details of that record are displayed by the SHOW *REFRESH command.

Enter some details for each field, and then press the **Commit Data** function key. If you leave any required fields blank, HP ALLBASE/4GL displays an error message and returns you to the blank field. Otherwise you will exit from the screen processing and the following message is displayed:

```
P-customer_proc: 2:END STEP
```

This indicates that you have just completed the SCREEN command and have returned to the *customer_proc* process. The remainder of the process is executed, returning you to the start of the process. After the record has been written to

the file, you are returned to the start of the *customer_scrn* screen. The screen is not cleared at the completion of the SCREEN command. The screen display is only altered when a SCREEN command for a different screen is issued, or the user returns to a previous menu.

Note

Trace mode creates a file containing all the lines that are echoed to the screen while trace mode is active. This file defaults to a temporary file called HP4TRACE in your current logon group and account.

Handling Problems

Occasionally, your application will not behave as you expect, or you will receive error messages while you're testing the application. The next few pages show you how to tackle some common problems.

Items Not Found

The most common problem encountered when testing an application is discovering a reference to an undefined item. HP ALLBASE/4GL detects references to undefined items at different stages and the system response depends on the type of item that has not been defined.

Generating an item resolves references to certain other application items. The nature and extent of the check applied to a referenced item depends on what type of item it is.

In general, the generate program doesn't attempt to resolve references to other generated items when generating an item. The exceptions are record layouts and calculated items.

The generate program attempts to resolve references to most non-generated items during generation. The exceptions are menus, help screens, and function keys.

The following table summarizes the various components that require generation, and the components that do not require generation. The table also indicates the items for which references are resolved during generation.

Generated Items	Non-Generated Items
Record layouts* Decision tables Calculated items* Data screens Processes Reports Window screens Functions Messages	Ranges* Field specifications* Alphanumeric constants* Files* Help screens Numeric constants* Variables* Menus Scratch-pad field names* Validation tables* Function keys Application titles*
* References to these items are resolved during generation.	

Since some references are not checked during generation, there is a possibility that you can successfully generate an application with unresolved references to the following items:

- Data screens.
- Windows.
- Functions.
- Menus.
- Processes.
- Decision tables.
- Reports.
- Messages.
- Help screens.
- Function keys.

If the application encounters an unresolved reference at run time, you will receive an error message telling you that an item cannot be found. Usually processing can

continue. If an undefined reference is critical to the logical flow of the application, you may not want processing to continue. In cases such as missing messages or help screens, the absence of the item may not have adverse effects on the logical flow of the application.

It's up to you to decide if you want to continue with testing the application when you run into this sort of situation. HP ALLBASE/4GL has a number of utilities to help you locate items in your application. Refer to Lesson 10 Developer Utilities, page 7 – 19 for a description of these utilities.

The Initial Action

If HP ALLBASE/4GL can't find the initial action (as defined by the system administrator) for your application, processing cannot commence. If this happens, check the exact definition of the initial action for your application. You may need assistance from your system administrator to gain access to the administrator application. For details on accessing the application definition screen, refer to the *HP ALLBASE/4GL Developer Administration Manual*.

If you experience this problem you must ensure that the initial action is the correct type, and has the correct name. The initial action can only be a menu or a process. Remember, HP ALLBASE/4GL is case sensitive, so make sure you use the exact spelling of the initial action name.

Screen Processing Review

This review describes the three major components of the data screen processing cycle. These components are:

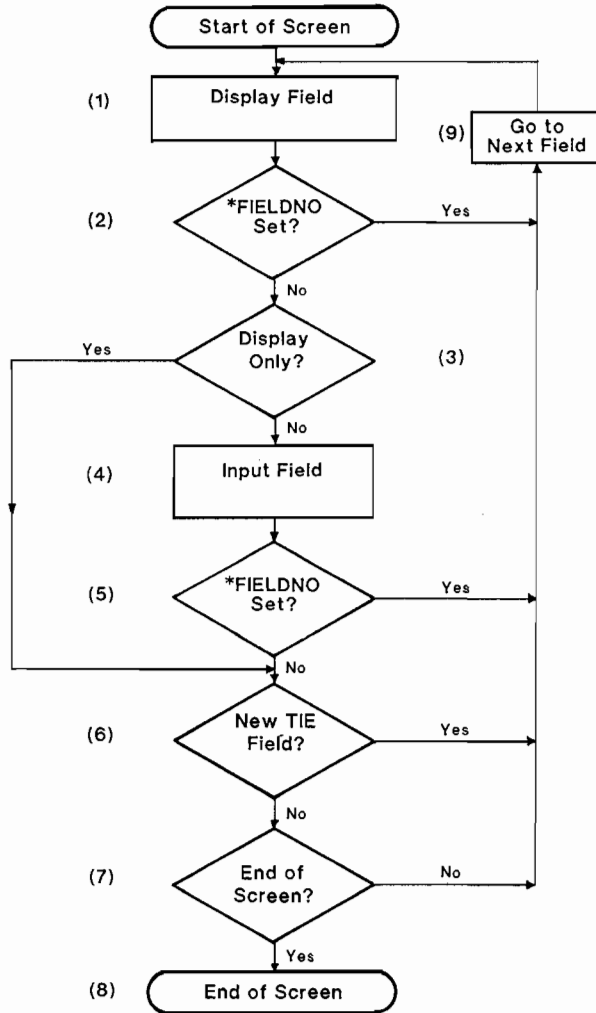
- Full data screen processing.
- Display field processing.
- Input field processing.

It is important that you understand the screen processing sequence. You may wish to review the Screen Processing Tutorial in Lesson 6 again while completing this review.

Full Data Screen Processing

This order of processing controls the way the cursor moves from field to field, and how the screen is completed.

The numbers in the following description refer to the step numbers in the diagrams.



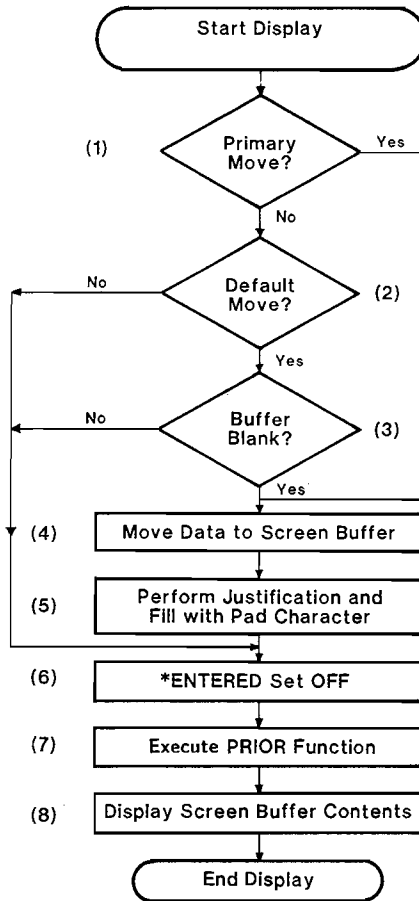
1. When HP ALLBASE/4GL displays a data screen, the display field processing is invoked for the first screen field. (This processing is described later.)
2. HP ALLBASE/4GL tests the value in the *FIELDNO communication area field. *FIELDNO is a communication area field containing the number of the current screen field. This value can be set with the MOVE command to explicitly move a screen field number to *FIELDNO. If *FIELDNO has been set, control passes immediately to the field specified in *FIELDNO. Otherwise processing continues normally.
3. The current field is tested to check if it allows data input. If it does, processing continues; otherwise it branches to step 5.
4. Input field processing is performed. (This is described later.)
5. At the end of the input field processing, HP ALLBASE/4GL tests the contents of *FIELDNO again. If the *FIELDNO value has been changed during input field processing, control passes to the field indicated by the new value in *FIELDNO. Otherwise processing continues normally.
6. This step tests the communication area field *NEWTIE to determine if it has been altered during input field processing. *NEWTIE is a communication area field that contains the number of the next field to be processed. If *NEWTIE has been altered, control passes to the screen field indicated by the value in *NEWTIE. Otherwise processing continues normally.

The value in *NEWTIE can be set by moving a new value to it, or by the TIE logic command. The TIE command takes a screen field name as its argument and moves its number to *NEWTIE.

7. HP ALLBASE/4GL determines if there are any more fields on the screen. If so, control passes to the next screen field.
8. At the end of the screen, one of two things can occur. If the screen has input fields, the cursor remains on the last input field. If the screen doesn't have any input fields, the cursor is positioned at the base of the screen. In both cases, the system waits for the user to press a valid function key.
9. This step resets system values such as *FIELDNO and *NEWTIE and passes control to the next field.

Display Field Processing

HP ALLBASE/4GL performs display field processing for every field (display and input) on the screen. The display processing for a field occurs when control first passes to the field.

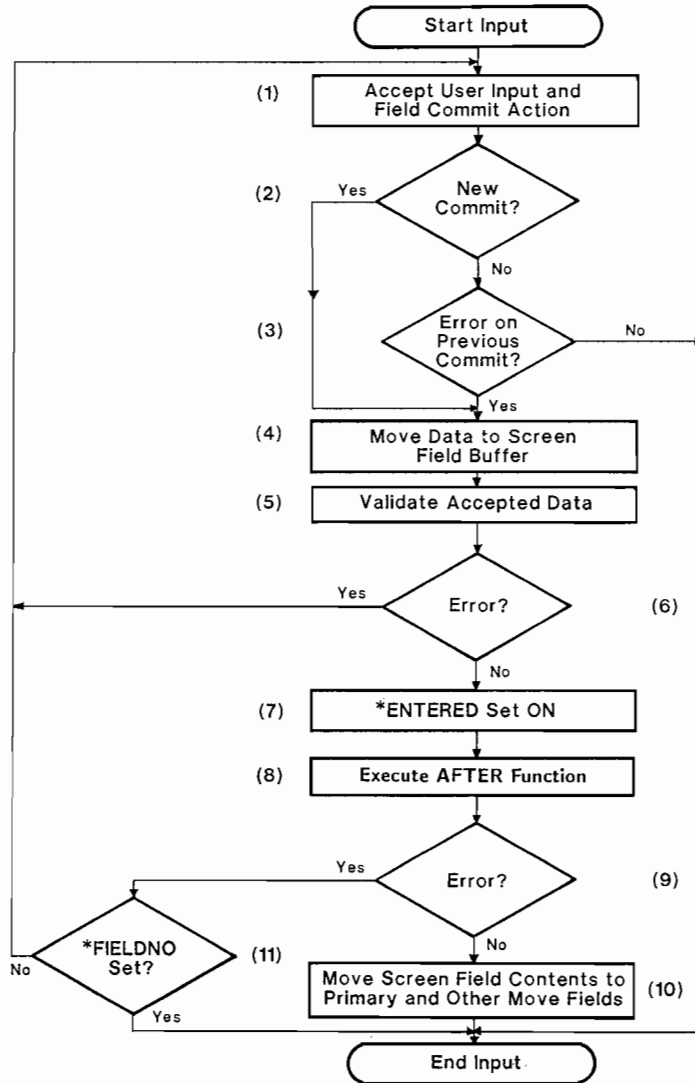


1. HP ALLBASE/4GL determines if a primary data movement field is specified for this screen field. If one is specified, processing branches to step 4.
2. This step checks if a default data movement field is specified. If not, processing branches to step 6.

3. The current screen 'field's buffer is tested. If it contains data, processing branches to step 6. HP ALLBASE/4GL only moves data from the default data movement field to the screen field buffer if the screen field buffer is blank.
4. HP ALLBASE/4GL moves data from the primary data movement field (or the default data movement field) to the screen field buffer.
5. The screen field buffer contents are justified as required and filled, if necessary, with the pad character. They are not immediately displayed on the screen.
6. The switch *ENTERED is set off.
7. If there is a prior function assigned to the field (that is, the function has been defined to be executed either prior to or, both prior and after data entry) then it is executed now. This function can test *ENTERED to determine when it is being called.
8. Finally, the contents of the screen buffer are displayed in the screen field.
Display field processing is now complete and control returns to the full data screen processing logic described earlier.

Input Field Processing

HP ALLBASE/4GL performs input field processing for input fields after display processing for the field has been completed.



1. When HP ALLBASE/4GL first enters this processing sub-system, it places the cursor in the current field and waits for user input. This is the only time that HP ALLBASE/4GL accepts any form of user input, including function keys or cursor movement requests.

A field commit action indicates that data entry for the field is complete. Some actions that constitute a field commit are:

- Moving the cursor off the field by:
 - pressing **Return**,
 - pressing **Tab** or **Shift** + **Tab** followed by **Return**,
 - moving the cursor to another field by pressing **▲**, **▼** or **▽**, followed by **Return**
- Pressing the **Commit Data** function key.

Note

In general, pressing a function key other than **Commit Data** does not initiate a field commit.

There are only two cases in which a field commit occurs as a result of pressing a function key:

- If the action behind the key initiates a commit data action, or
- If the action is a function which moves a new value to *FIELDNO, a field commit occurs on return to the screen field.

Once HP ALLBASE/4GL accepts the field commit, the remainder of the input field processing is executed.

2. HP ALLBASE/4GL determines if the field commit is a new commit action for this field. A new field commit is recognized for the following situations:
 - The data in the screen field has altered since the last commit action for the field.
 - A different numbered input field has been processed since this field was last committed.
3. If the field commit is not a new commit action, HP ALLBASE/4GL checks to see if an error was generated by the previous commit action. If an error was

generated on the previous commit, this commit action is accepted. Otherwise input processing for the field is terminated.

4. The data input in the screen field is accepted into the screen buffer.
5. HP ALLBASE/4GL now validates the screen field buffer contents. Firstly, the following automatic system tests are performed:
 - **Field Requirements.** Validation fails under the following circumstances:
 - The field is blank and it is defined as a required field,
 - Less than the defined minimum number of characters for the field have been entered.
 - **Data Type Requirements.** HP ALLBASE/4GL performs the following data type validation:
 - Correct data type for edit code,
 - If a date field, validate the date format,
 - If numeric, validate number of decimal places,
 - If signed numeric, validate sign.

If any of these criteria are not met, HP ALLBASE/4GL generates an error signal, displays a standard system error message, and terminates the validation.

If the input field data passes the system validation tests, any developer specified validation tests (if specified) are applied. These tests are as follows:

- Value within a specified range.
- Value in a specified table.

If either of these tests fail, HP ALLBASE/4GL displays the developer defined message assigned to the range or table. If there is no message associated with the range or table, HP ALLBASE/4GL displays a default system error message.

Error messages are one of five different types of messages. They are the most important type of message in terms of their effect on input field processing logic. They provide an automatic means of ensuring that the user enters valid data in a field. You will define a number of messages in later lessons.

6. If the validation tests of the screen field data initiate an error message, processing returns to step 1 and the user must re-enter the field data.

7. When the field data has been accepted and validated, the switch ***ENTERED** is set on. This allows you to determine whether the function is being performed prior to, or after, data entry for the field.
8. The function, if specified as an after function, or as both a prior and after function, is executed.
9. The function, through its processing, may initiate an error message. If so, processing branches to step 11.
10. HP ALLBASE/4GL now performs the automatic data movement for the field. The contents of the screen buffer are moved to the primary data movement field, and the other data movement fields. Processing of the field is complete.
11. If the after function generates an error message, HP ALLBASE/4GL tests the value in the ***FIELDNO** communication area field. If this value has not been modified by the function, processing returns to the beginning of the field (Step 1). However, if the function has modified the value in ***FIELDNO**, HP ALLBASE/4GL restores the contents of the field to the condition that existed at the start of the input processing, and then terminates processing of the field.

Summary

It's important to understand the actions that occur during data screen processing. A number of system items are available to help you make full use of data screens, and make them operate efficiently for your users. These system items are summarized below.

- ***FIELDNO**. A communication area field containing the number of the current screen field. You can alter this value by moving a new number to it with the **MOVE** logic command. Moving a value to ***FIELDNO** alters the screen field processing sequence, allowing you to branch to a field depending on the value of another.
- ***NEWTIE**. A communication area field that contains the number of the next field to be processed. You can alter this value manually using the **TIE** command, or by moving a new value to it with the **MOVE** command. It differs from ***FIELDNO** in that it only takes effect once all normal input processing for the field has finished.
- ***ENTERED**. A read-only switch that enables you to determine when a screen field function is being executed. ***ENTERED** is off before data entry, and on after successful data entry.

- **ERROR message.** If an ERROR message is generated by a data input validation test or by an after function, HP ALLBASE/4GL automatically requests the user to correct the data entry and commit the field again. This prevents the user committing invalid data.

Data Movement

When HP ALLBASE/4GL first displays a field, it moves data to the internal screen field buffer before it executes the field function. Data is not displayed on the screen until the function finishes. This means you can refer to the screen buffer contents and modify them with the function. The modified data is displayed upon completion of the function.

HP ALLBASE/4GL executes the after function for a field before the data movement for the field occurs. This means that the after function cannot rely on having the values available in data movement destination fields. However, the after function can access the value in the internal screen field buffer, and if required, modify this value before the data movement for the screen field occurs.

If any automatic data validation, or data validation in an after function, results in an error, the normal data movement for the screen field does not occur.

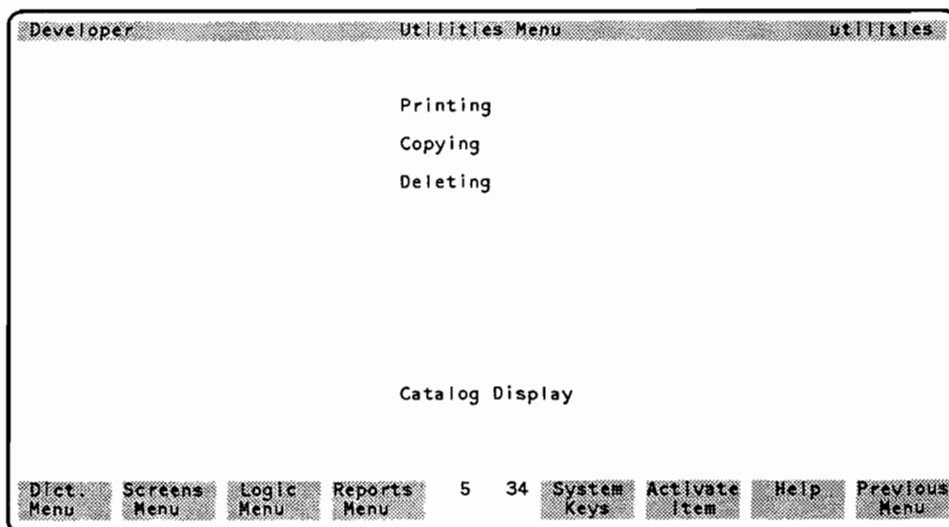
Lesson 10 – Developer Utilities

Objectives

This lesson introduces you to a number of the utilities available in the developer that allow you to manipulate and check items within your application.

The Utilities Menu

From the main menu, select the *Utilities* option to display the utilities menu.



This menu allows you to use the utilities to:

- Print a hard copy listing of your application, or selected portions of it.
- Copy an item that already exists in your application to another item. It also allows you to copy an item from another application or version that exists in the same HP ALLBASE/4GL system.
- Delete any item that you have defined in your application.
- Display a catalog of the names of items defined in your application. This item is also available from other menus in the developer.

Displaying Catalog

Select and activate *Catalog Display*. This utility allows you to look at the names of items defined in your application. This utility is available from most menus in the developer.

Developer		Catalog Display		catalog	
1	Field Specs.	10	Record/SQL Table Layouts	16	Processes
2	Ranges	11	Files	17	Functions
3	Tables			18	Decision Tables
4	Variables	12	Messages		
5	Calculated Items	13	Help Screens	19	Reports
6	Numeric Constants				
7	Alpha-Numeric Constants	14	Screens	20	SQL Select Lists
8	Scratch-Pad Fields	15	Function Keys	21	SQL Logic Blocks
9	Application Titles			22	Work Areas

Item Type Number

Printing Copying Deleting 14 45 System Commit Help Previous
 Keys Data Menu

There is only one data entry field on this screen. This is the number indicating the type of item names you want displayed. Follow the steps shown here to display the names of the screens you have defined so far in your application.

Action

Enter 14

Press

Commit Data

Press **Return**

Notes

This is the number corresponding to *Screens*. The upper part of the screen lists these items and numbers.

This displays the names of all the screens defined for this application.

Depending on how many of these items you have defined for this application this has one of two effects. If there is more than one screen of names, another screen will be displayed. This process continues until all the names have been displayed. Once the last screen of names has been displayed you are returned to the catalog display screen.

You can use the same method to display the name list of any items in your application.

With the catalog display utility, you can determine if the item you are looking for actually exists. If you have mistakenly created an item with an incorrect name, you can use the copy and delete utilities to correct the error. To rename an item, you must copy it to a new name and then delete the original.

Step through the item numbers and see the items you have created in your application.

Copying Catalog Items

Select *Copying* from the utilities menu. This allows you to copy an item that already exists in your application. You can also use this screen to copy an item from another application or version in the same HP ALLBASE/4GL system to an item in your application.

Developer		Catalog Copying		copying
1	Field Specs.	10	Record/SQL Table Layouts	16 Processes
2	Ranges	11	Files	17 Functions
3	Tables			18 Decision Tables
4	Variables	12	Messages	
5	Calculated Items	13	Help Screens	19 Reports
6	Numeric Constants			
7	Alpha-Numeric Constants	14	Screens	20 SQL Select Lists
(n/a	Scratch-Pad Fields)	15	Function Keys	21 SQL Logic Blocks
9	Application Titles			22 Work Areas
Application to copy FROM		training		
Item Type Number		1		
Item Name to Copy FROM		customer_no		
Item Name to Copy TO		customer_no.new		
Description				
customer_no				
AUTHOR: rmjones				
This field forms a unique six digit customer identifier				
Last Modification:		Date	Time	
Printing	Deleting	Catalog Display	16 63	System Keys
				Commit Data
				Help
				Previous Menu

Note

You are not required to make the following entries. However, you may if you wish to practice using the copying utility.

First you are given the option to enter the name of the application or version you want to copy from. The display in the *Application to copy FROM* field defaults

to the name of the application or version you're currently developing. You would type over this default to copy an item from a different application.

You then need to enter the number that indicates the type of item you want to copy; enter 1

The next field is for the name of the item you want to copy; enter **customer_no**

Finally, the last field is the name of the new item you want to create. If an item of this type with this name already exists, the copy is not allowed. Enter **customer_no_new**

To actually copy the item, press the **Commit Data** function key. When the copy is finished, HP ALLBASE/4GL prompts you for the name of the next item to copy.

Now *customer_no_new* has the same definition as *customer_no*.

Note

Since you can copy items from another application, you could set up a "library" application on your system to store a variety of standard items. For example, you may have some standard field specifications or functions that are used by a number of applications. Instead of creating the item each time it's needed, you can create the item once in the library application, and then copy the items as you require them.

Deleting Catalog Items

Select *Deleting* from the utilities menu. This allows you to delete any item defined in an application.

Developer		Catalog Deletions		deleting	
1	Field Specs.	10	Record/SQL Table Layouts	16	Processes
2	Ranges	11	Files	17	Functions
3	Tables			18	Decision Tables
4	Variables	12	Messages		
5	Calculated Items	13	Help Screens	19	Reports
6	Numeric Constants				
7	Alpha-Numeric Constants	14	Screens	20	SQL Select Lists
8	Scratch-Pad Fields	15	Function Keys	21	SQL Logic Blocks
9	Application Titles			22	Work Areas
<p>Item Type Number 1</p> <p>Item Name to Delete customer_no_new</p> <p>Description</p> <p>customer_no</p> <p>AUTHOR: rmjones</p> <p>This field form a unique six digit customer identifier</p> <p>Last modification: Date Time</p>					
Printing	Copying	Catalog Display	15	40	System Keys
					Commit Data
					Help
					Previous Menu

Note

You are not required to make any entries here. If you did create the *customer_no_new* field specification previously, then you can follow these steps to delete it.

Caution

This is a physical deletion of the item. You cannot retrieve an item after you delete it.

Once again, you must first enter the number indicating the type of item you want to delete. Enter 1

Then you can enter the name of the item to be deleted. Enter *customer_no_new*

When you press **Return**, HP ALLBASE/4GL displays the description of the item if it has one.

Press the **Commit Data** function key to delete the item.

After you press the **Commit Data** function key, HP ALLBASE/4GL displays a message confirming that the item has been deleted.

Printing Catalog Items

Select *Printing* from the utilities menu. This enables you to print a hard copy listing of your entire application, or selected portions of it.

Developer		Catalog Printing		printing
1	Field Specs.	10	Record/SQL Table Layouts	16 Processes
2	Ranges	11	Files	17 Functions
3	Tables			18 Decision Tables
4	Variables	12	Messages	
5	Calculated Items	13	Help Screens	19 Reports
6	Numeric Constants			
7	Alpha-Numeric Constants	14	Screens	20 SQL Select Lists
8	Scratch-Pad Fields	15	Function Keys	21 SQL Logic Blocks
9	Application Titles			22 Work Areas
Catalog Item Type Number			I	(1 - 22 or A for All Item Types)
Short Index List or Full Index List			N	(S/F/None)
Print Catalog Items Details			Y	(Y/N)
All Entries or Selected Names			S	(A/S)
Selected Names				
customer.no		customer.name		
Copying Deleting Catalog Display 19 36 System Keys Commit Data Help Previous Menu				

Note

You are not required to make any entries here, however you may wish to print a list of the items you have developed so far.

Entering the Field Values

Catalog Item Type Number. Enter 1

This field allows you to indicate the type of item you want printed. You can either enter the appropriate code 1 - 22 for the group of items you want, or enter **A** to specify that you want all item groups printed.

Short Index List or Full Index List. Accept **N**

You can print an index list of catalog item names with the text of the short description field or with the full description text. To list item names and their short description, enter **S** for *short index*. To list item names and their full text description, enter **F** for *full index*. Enter **N** if you don't want an index list of catalog item names.

Print Catalog Item Details. Accept **Y**

Enter **Y** if you want to print the detailed definition of each item. Enter **N** if you only want to print an index list.

All Entries or Selected Names. Enter **S**

Here you can specify that you wish to have all the entries for the particular item group printed by entering **A**. If you enter **S** for selected entries you must complete the next fields.

Selected Names. Enter `customer_no` and `customer_name`

These fields allow you to enter the names of up to 16 items for the given type to be printed. If you enter any names in these fields, the entry in the field above automatically changes to **S**. If you don't want to enter the names of any items, press the **Commit Data** function key.

To Print the Items

Press the **Commit Data** function key to commence the printing. If no items of the type you specify can be found, HP ALLBASE/4GL displays a message and no report is produced. If you have given a list of names of items to be printed, those that can be found are printed. A message is printed on the report if a requested item could not be found.

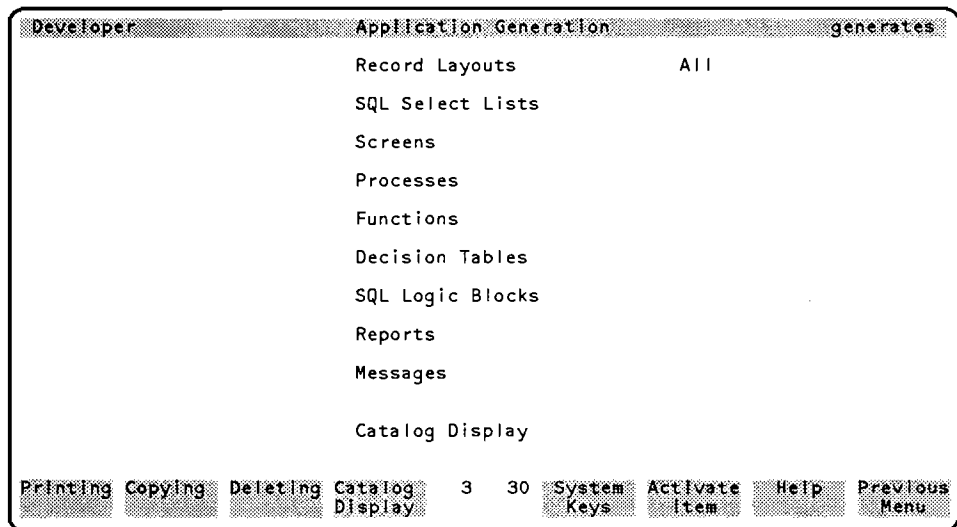
By default, the report is printed on the system line printer.

Generating Items

Sometimes your application may not proceed correctly even though you have correctly defined all the items. The most probable cause is that you haven't generated an item that requires generation.

To generate a single item, go to the screen where you defined the item, call up the item, and then press the **Generate** function key.

The *Application Generation* menu allows you to generate groups of items in your application. Select the *Generates* option on the main menu to access this menu.



Note

There is no need for you to actually do anything here, although you won't cause any damage if you do test the system.

The generates menu allows you to generate all the defined items within a particular group, or all the items within your application. If you press **Return** or the **Activate Item** function key, all the items of the selected type will be generated.

As the generation process commences, a number of messages are displayed. If necessary, the generate program is called. A message then advises you which type of items will be generated, and the generation for each item commences.

If each item generates without error, HP ALLBASE/4GL displays a confirmation message. However, if an error is encountered, a message appears detailing the errors. This display remains on the screen for about 30 seconds. Generation of the next item then commences. Generate will report up to 20 errors. If more than 20 errors are detected, the generation continues, but no more errors are reported.

Note

When you call generate, HP ALLBASE/4GL creates a file to log all generate errors. The file defaults to a temporary file called HPTGNERR in your login group and account. Each time an error is encountered by generate, the name of the item and the errors are listed to this file. This means that you can set a *generate all* going and then leave your terminal. When you return you can look at the file for any errors.

The usual reason for generating a group of items is to update the items to reflect a change in the definition of an item which is resolved during a generate. For example, if you modify the length of a field specification, the change would only be reflected in the screens referring to this specification after you generate the screens again.

Summary

This lesson introduced you to the utility functions available in the developer. These utilities allow you to:

- List the names of the items in an application.
- Copy items within an application, or copy items from other applications or versions.
- Delete items in an application.
- Print documentation about some or all of the items in an application.
- Generate all or part of an application.

Intentionally Blank

Contents



Chapter 8: Reports

What is an HP ALLBASE/4GL Report?	8 - 1
Format of a Report	8 - 1
Line Groups	8 - 2
Line Numbers	8 - 2
Totalling Facilities	8 - 3
The Customer Report	8 - 3

Lesson 11 – Report Definition

Objectives	8 - 5
Report Header	8 - 5
Report Line Header	8 - 8
Summary	8 - 14

Lesson 12 – The Report Painter

Objectives	8 - 15
Painting a Report	8 - 15
The Window Keys	8 - 17
Painting the Customer Report	8 - 17
Entering Text Items	8 - 18
Defining Data Fields	8 - 19
Using Dictionary Field Specifications	8 - 22
Saving a Report	8 - 25
Summary	8 - 25

Lesson 13 – Generating and Running a Report

Objectives	8 - 26
Report Generation	8 - 26
Executing a Report	8 - 26
Testing by Menu Bypass	8 - 27
Testing from a Menu	8 - 27
Summary	8 - 28
From Here On	8 - 29

Intentionally Blank

8 Reports

Now you have some data in your file, the next step is to produce a hard copy report, or to display a report on a terminal. The lessons in this chapter step you through the definition and creation of a typical report. The report you are about to create enables you to look at the entire contents of the customer file that you have created. The lessons in this chapter cover three areas:

- Defining a report.
- Painting a report image.
- Executing a report.

HP ALLBASE/4GL allows you to design reports that are printed on a printer or displayed on the user's terminal. A report painter enables you to quickly create an image of a report. HP ALLBASE/4GL also allows you to process data before it is included in a report.

What is an HP ALLBASE/4GL Report?

Some of the concepts in this chapter may be new to you if you haven't used a report generator before. An HP ALLBASE/4GL report is not created or processed in a physical line by line manner as most reports are. An HP ALLBASE/4GL report is an arrangement of print lines defined by you, and printed in an order determined by the report processor based on your definition of the report.

Format of a Report

The data included in a report is taken from a data file called the report's *primary file*. Data can also be included from other files as well. This is done by using *file linkages* to the other files. The report processor reads through the primary file sequentially and creates a temporary file to produce the actual report. The temporary file contains the records that meet the defined selection criteria, and they are sorted according to the specified sort sequence. Once the selection and sorting process of records from the primary file is complete, the report processor sequentially prints the records from the temporary file.

Three important concepts in HP ALLBASE/4GL reports are:

- Line groups.
- Line numbers.
- Totalling facilities.

The following descriptions provide an overview of these concepts.

Line Groups

The format of an HP ALLBASE/4GL report is defined by a series of different line groups. A line group is a logical entity that the report processor prints at a certain stage in the report printing process. HP ALLBASE/4GL allows you to use a number of different line groups. They are identified as follows:

Name	Description
P1	Top of page headings.
C1	Column headings.
B1	Bottom of page footings.
D1 to D9	Detail lines.
E1 to E9	Extra lines.
H1 to H8	Subheading lines.
T1 to T8	Subtotal lines.
TF	Final total line.

As their descriptions suggest, HP ALLBASE/4GL processes each line group at a different stage of the report. For example, HP ALLBASE/4GL prints the P1 line group at the start of a new page, followed immediately by the C1 line group. When the report concludes, HP ALLBASE/4GL prints the TF line group. A line group that is not defined for a report will not be printed. Only the D1 line group must be defined for each report. HP ALLBASE/4GL prints a D1 line group for each record selected for reporting.

Line Numbers

You may be wondering why line groups are named *groups*. A line group may, in most cases, consist of up to 99 physical print lines. The following table summarizes the number of lines that may be used within each line group.

Line Group	Line Number Range
P1	1 to 99
C1	1 to 3
B1	1 to 9
D1 to D9	1 to 99
E1 to E9	1 to 99
H1 to H8	1 to 99
T1 to T8	1 to 99
TF	1 to 99

Each time the report processor prints a line group, all of the line numbers defined for that line group are printed. They are treated as one logical entity.

Totalling Facilities

Most reports usually require some type of totalling. With HP ALLBASE/4GL reports there are a number of automatic totalling facilities available. The totals are maintained in a number of communication area fields. These are as follows:

Name	Description
*TOTALS(1) - (16)	These fields allow you to total values down your report. You can accumulate the values in one field on many lines.
*CROSS(1) - (5)	These fields allow you to total values across a physical line. You can accumulate the values from several fields that appear on one line.
*COUNT(1) - (5)	These fields will count the number of times that a specified print line has been printed.

With this basic outline of a report, you can now create a report for the *customer* file.

The Customer Report

The report that you are about to create lists all the records in the *customer* file. The following sample printout gives you an indication of what your first report will look like. This sample also shows the line numbers, discussed previously, associated with each line printed.

P1.01: Customer File ... Page: 0001
P1.02: Date: 08/08/88

C1.01:	#	Name	Address
D1.01:	1	R. M. Jones	20 Test Street
D1.02:			Newtown
D1.03:			New York
D1.04:			State / Zip NY 12345

D1.01:	201112	William Shakespeare	Stratford on Avon
D1.02:			England

D1.04: State / Zip IA 54321

TF.01: Total Number of Customers Listed 2

The two line page heading contains the date and time of the report, a report title, and the current page number. A column heading line immediately under the page heading identifies the columns of data to be printed. The body of the report contains the details of each customer, printed over a maximum of four physical print lines for each customer. For each customer, the first and fourth lines are always printed. The second and third lines are only printed if data for the second and third address lines exists.

At the end of the report, a single line summary shows how many *customer_rcrd* records have been included in the report.

Lesson 11 – Report Definition

Objectives

In this lesson you will define a report header and a number of report line headers. To define a report, you must first define the report header. Then you define the remaining details of the report in any sequence. The screens you use are:

- **The report header screen.** Allows you to define the operational environment of the report.
- **The report line header screen.** Allows you to define the characteristics of the lines that make up the report.
- **The report painter.** Allows you to paint an image of the various report lines you have defined.
- **The report sorting screen.** Allows you to define the order in which the records from the file are to be printed.
- **The selection criteria screen.** Allows you to define a number of selection criteria which must be met for a record to be included in the report.
- **The file linkages screen.** Allows you to specify when, and from which files, extra information is extracted while the report is being produced.

In the lessons in this chapter you will use the first three of these items. The others are optional. To keep this report simple, the optional items are not used.

Report Header

The first item you must define for every report is the report header.

Menu Path

Reports, **H**header

Description

This screen allows you to define the basic operational specifications of the report. These specifications include the name of the report, the primary file for the report, the type of stationery for the report, the printer to be used, and a brief description of the report.

Developer	Report Header	report_header
Report Name	<input type="text"/>	Secured <input type="checkbox"/> (Y/N)
Report File Designator	<input type="text"/>	Index <input type="checkbox"/>
Primary File[.Record]	<input type="text"/>	
Printer	(F/L/D)	
Type of Stationery	<input type="text"/>	Number of Copies <input type="text"/>
Characters per Line (maximum)	<input type="text"/>	Actual Page Size in Lines <input type="text"/>
Print Lines Used per Page	<input type="text"/>	Formfeed Skip to Next Page <input type="checkbox"/> (Y/N)
Start of Report Function Name	<input type="text"/>	
End of Report Function Name	<input type="text"/>	
Description	<input type="text"/>	
Last Modification:	Date <input type="text"/>	Time <input type="text"/>
Sorting	Record Select	Line Header
Generate Report	3 36	Help
	Commit Data	System Keys
		Previous Menu

Entering The Field Values

Report Name. Enter `customer_rept`

This is the name of the report you are creating.

Secured. Accept the default value of N

This specifies whether this report is secured against an unauthorized developer changing the definition.

Report File Designator. Enter `cust.`

This is the name of the MPE XL formal file designator for the report output file. You can use an MPE XL file equation to direct the report to a specific device, or to a disc file. By default, the report is directed to the device class LP.

Primary File[.Record]. Enter `customer`

This is the name of the data file that is the primary file for this report. In this particular report, all the records in this file are printed.

Index. Accept the default value of 1

This is the index used to read the file. Since this report is not sorted, it also determines the order in which the records are printed. In this report the file is printed in customer number order.

Printer. Enter D

The entry in this field specifies whether the report is printed to a formal file designator (F), a local printer attached to your terminal (L), or displayed on your terminal screen (D).

Type of Stationery. Leave this field blank.

This field specifies the stationery type that should be loaded on the printer for this report. If the specified stationery is not loaded on the printer, MPE XL request the system operator to change the paper to the correct type.

Number of Copies. Accept the default value of 1.

You can print multiple copies of the same report by specifying a value greater than 1 in this field.

Characters per Line (maximum). Accept the default value 80

This entry specifies the maximum allowable report line width. The width you specify also determines the maximum width that the report painter allows you to use when you create an image of this report.

Actual Page Size in Lines. Accept the default value of 22

This value specifies the physical form length of the paper for the report.

Print Lines Used per Page. Accept the default value of 22

This is the logical page size. That is, the maximum number of lines that can be printed on each physical page.

Formfeed Skip to Next Page. Accept the default value of N

This specifies the type of form feeding control that is used. If you use non-standard size stationery, you may need to use linefeeds to advance to the next page. Entering Y specifies a formfeed, and N specifies multiple linefeeds.

Start of Report Function Name. Leave blank.

This is the name of a function that is executed before HP ALLBASE/4GL starts producing the report. This function can display a screen to obtain user input for parameters that are used within the report, or data that is used in the report.

End of Report Function Name. Leave blank.

This is the name of a function to be executed after the report is produced. You could use this function to update data files as a result of totals obtained during the generation of the report.

Description. Enter a suitable description.

Completing and Committing the Screen

When all the entries on this screen are correct, press the **Commit Data** function key to create the report header.

The report header is now complete and you can go on to define the remainder of the report. The next step is to define the print lines used for this report. Press the **Previous Menu** function key to return to the reports menu and continue with the next stage.

Report Line Header

Before you can actually print the report image, you must define the lines that are used in the report. This report uses a total of eight print lines from four of the available line groups.

Menu Path

Reports, Line Header

Description

You use this screen to specify the line groups used by the report, and define the way in which each of the print lines is handled. This screen allows you to specify the following details for each line:

- How many blank lines to leave before or after printing.
- Whether the line is always printed.

- Which count field to use to count the number of times the line is printed.
- The names of functions to be executed before and after each line is printed.

Developer	Report Line Header		report_line_head	
Report Name	customer_rept		Secured	N (Y/N)
Line Group	P1 (P1/C1/B1/Hn/Dn/En/Tn/TF)			
Line Number	01			
Action	A (A/C/D)			
Skip Lines/Page:	Before Print	<input type="checkbox"/> (0-9/P)		
	After Print	<input type="checkbox"/> (0-9/P)		
Suppress Line If:	All Values Are Zero	<input type="checkbox"/> (Y/N)		
	All Values Are Blank	<input type="checkbox"/> (Y/N)		
Count Lines Printed Into Counter Number		<input type="checkbox"/> (1-5,Blank)		
Underline Numbers		<input type="checkbox"/> (P/A/B)		
Before Print Function				
After Print Function				
Grp #	Act	Skip	Suppress	Count Underline Function Function
Header	Record	File	Generate	19 45 System Commit Help Previous
Select	Linkage	Report		Keys Data Menu

Entering the Field Values

The characteristics of each line are defined one line at a time. The eight lines to be printed are P1.01, P1.02, C1.01, D1.01 through D1.04, and TF.01.

Report Name. Enter `customer_rept`

This is the name of the report that you are about to define the lines for. The name in this field defaults to the last report name used in any of the report development screens.

Line Group. Enter `P1`

This is the line group identifier. The `P1` group is the page header group and it specifies the lines that HP ALLBASE/4GL prints at the start of every new page. The full list of available line groups is as follows:

Name	Description
P1	Top of page headings.
C1	Column headings.
B1	Bottom of page footings.
D1 to D9	Detail lines.
E1 to E9	Extra lines.
H1 to H8	Subheading lines.
T1 to T8	Subtotal lines.
TF	Final total line.

Line Number. Accept the default value of 01

This is the line number specifier within the line group that you just entered above. The lines are printed on the report in ascending numeric order. The combined identifier of *line group.line number* is known as a *print line*. The available range of print lines is:

Line Group	Line Number Range
P1	1 to 99
C1	1 to 3
B1	1 to 9
D1 to D9	1 to 99
E1 to E9	1 to 99
H1 to H8	1 to 99
T1 to T8	1 to 99

Action. Accept the default value of A

This action code is used in the same manner as the other action codes, that is: *A* = Add, *C* = Change, and *D* = Delete. The action defaults to an appropriate code that depends on whether the print line already exists, or you are defining a new print line.

Skip Lines/Page:

The next two fields determine the line spacing used for this particular print line.

Before Print. Accept the default value of 0

This indicates that no line feed characters are issued before this line is printed. You can enter a value between 1 and 9 to specify a number of line feeds, or you can enter P to force a page break before this line is printed.

After Print. Accept the default value of 1

This indicates that one line feed character is issued after this line is printed. You can skip 1 to 9 lines after this line is printed, or force a page break.

Note

The print line buffer is only cleared after it is printed. If you specify that a line has 0 lines skipped after printing, the line is not printed and the buffer contents remain intact. If the next line printed skips 0 lines before printing, the data is merged over the first line and then printed.

Suppress Line If:

The next two fields allow you to specify that a line is only printed if it contains non-blank or non-zero data.

All Values are Zero. Accept the default value of **N**

When set to **Y** this suppresses the printing of a line if all the numeric fields in the line are zero.

All Values are Blank. Accept the default value of **N**

When set to **Y** this suppresses the printing of a line if all the alphanumeric fields in the line are blank.

Note

There are two sets of conditions for these tests. The tests are relevant where there are only numeric or only alphanumeric fields on the print line. However, if the line contains both field types, all numeric fields must contain zeros **and** all alphanumeric fields must be blank before printing the line is suppressed.

Count Lines Printed into Counter Number. Leave blank.

This entry indicates which communication area *COUNT field (*COUNT(1) to *COUNT(5)) is incremented each time this particular print line is printed. Leaving this entry blank specifies that you don't require automatic line counting.

Underline Numbers. Leave blank.

This field indicates whether the numeric fields on the print line are to be overlined (P = Prior), underlined (A = After), or both (B = Both). A blank entry specifies that no underlining is done.

The underline or overline is printed on a separate print line to the print line. This line is counted as a regular line as the system tests to see if a new page is necessary. The data and its underlining or overlining always appear on the same page as it is treated as a single entity at print time.

Before Print Function. Leave blank.

This is the name of a function that is executed before the print line is printed. This function can perform data formatting, or extract data from another source to be included in the print line.

After Print Function. Leave blank.

This is the name of a function that is executed after the print line is printed. This function could possibly initiate the printing of extra lines on the report. These extra lines could be based on the line just printed.

Note

The functions that are executed before or after printing a print line can set two switches. These are the *BYPASS and *ENDLINE switches. *BYPASS controls the printing of each print line and bypasses the current print line when set on. *ENDLINE controls the line group, and terminates the processing of the line group when set on. This allows you to perform more complex selection and validation of data that is to be printed.

Completing and Committing the Screen

Press the **Commit Data** function key to commit the definition of the line group. HP ALLBASE/4GL displays a summary of the line at the base of the screen and then returns the cursor to the *Line Number* field. By default, the system expects you to define another print line in the same group. If you want to define another line group, go back to the *Line Group* field. You can now proceed to define the remaining seven lines in this report.

To Complete the Exercise

Only those fields that require an entry other than the default are shown here. Accept the default value for the other fields. Press the **Commit Data** function key when you have completed the details for each line.

Page Heading Line 2

Line Number	02
Skip Lines After Print	2

Column Heading Line 1

Line Group	C1
Line Number	01

Detail Line 1

Line Group	D1
Line Number	01
Skip Lines Before Print	1
Count Lines Printed ...	1

Note

The total number of customers included in the report is printed at the end of the report. Each time this line is printed, *COUNT(1) is incremented to count the number of records that have been printed.

Detail Line 2

Line Number	02
Suppress Line If: All Values are Blank	Y

Detail Line 3

Line Number	03
Suppress Line If: All Values are Blank	Y

Note

The two lines above contain the second and third lines of the customer address. They do not necessarily contain data, so they are only printed if data exists for them.

Detail Line 4

Line Number	04
-------------	----

Final Total Line 1

Line Group	TF
Line Number	01
Skip Lines before Print	3

This completes the definition of the characteristics for each line. You can now go on to paint the image of each line using the report painter.

Summary

In this lesson you defined a report header, and you also defined the line headers for the report.

The report header defines the operational environment for a report. It includes information such as the name of the primary file for the report, the printer to be used for the report, and the size and type of stationery for the report.

The report line headers define some details for the print lines that are used on a report. For each line, you can specify details such as the before print and after print line spacing, whether lines are suppressed if they are blank, whether a count is kept of the number of times the line is printed.

The reports menu does include some other screens for defining report sorting, record selection criteria, and file linkages. These screens are optional, and these additional definitions are not required in this report.

In the next lesson, you will use the report painter to complete the definition of this report.

Lesson 12 – The Report Painter

Objectives

In this lesson you will use the report painter to create the image of the *customer_rept* report.

The report painter is a program that is similar to the screen painter. It allows you to use your terminal to interactively “paint” the image of a report.

Painting a Report

In much the same way as painting a screen, you use the report painter to interactively create the layout of the data and text fields on a report.

As a report can be wider and longer than a standard terminal, the report painter offers you a window that you can move around over the report.

You can create two different types of fields on a report; text items, and data fields. Text items are created by simply typing them in where they are needed.

A data field is where you show the data being reported. Typically the data comes from a file field, although it can come from any HP ALLBASE/4GL data item. You create a data field manually by painting its image on the report line, or automatically by referring to a dictionary field specification name.

The report painter also allows you to specify the source of the data that is printed in the report output fields.

Menu Path

Reports, Painter

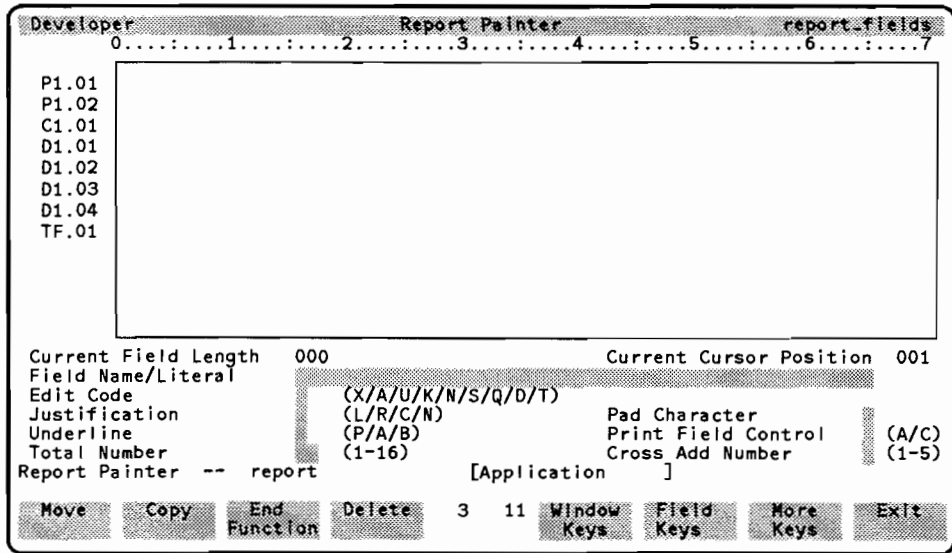
Start the report painter by activating the *Painter* menu item on the reports menu. When the report painter starts, you will see this prompt at the bottom of the screen:



The screenshot shows a terminal window with a prompt 'Enter report name' at the top left. Below the prompt, the text 'customer_rept' is entered. To the right of the text, there are several fields: a field containing '24', a field containing '1', and a field containing 'Abort Painter'.

This is where you enter the name of the report that you want to paint. The name of the report which you have been defining so far will be displayed as the default

entry. In this case it should be *customer_rept*. Enter the correct report name if the default entry is not correct and then press **Return**.



The report painter screen is divided into two regions. The upper region displays the window into the report. The lower region displays the details of the current report field.

A ruler line across the top of the report window shows your current position on the report page.

There are two important display fields at the top of this lower region. *Current Field Length* shows the length of the current field on the report line. The current field is the field currently occupied by the cursor. When you press **Return**, the report painter highlights the current field and updates the display in the *Current Cursor Position* field and *Current Field Length* field. *Current Cursor Position* shows the position of the cursor relative to the report, not the window. These fields enable you to determine where you are on the report.

A display area to the left of the report window lists the names of all of the print lines you have defined for this report. These are sorted into an appropriate sequence from the top to the bottom of the report page. If you defined more than 12 print lines, they will not all be displayed in the window.

When you first start the report painter, the window displays the upper left corner of the report.

The Window Keys

Since the upper region of the screen is a window into the report, you need to scroll the window left or right to see more of the report. For many detailed reports you also need to scroll the window up and down to see all of the lines defined for the report.

The **Window Keys** function keys allow you to scroll the report window. These keys allow you to define the distance and direction in which to scroll the window. When you press the **Window Keys** function key you will see the following keys displayed. Keys **Up Window** through **Right Window** are self explanatory.

```
Report Painter -- report customer_rept      [Application training]
Up Window   Down Window   Left Window   Right Window   3  10  Center Window   Set Horz Scroll   Set Vert Scroll   Main Keys
```

Pressing **Center Window** locates the center of the report in the window. This is the horizontal and vertical center.

Pressing **Set Horz Scroll** prompts you for the number of columns to skip whenever you press the **Left Window** or **Right Window** function keys. The default is 10.

Pressing **Set Vert Scroll** prompts you for the number of lines to skip whenever you press the **Up Window** or **Down Window** function keys. The default is 2.

Painting the Customer Report

The following layout shows the report you are about to paint.

```

1           2           3           4           5           6           7           8
P1.01:                                Customer File ...           Page: #####
P1.02:                                Date: #####

C1.01:  #           Name           Address
D1.01: #####          #####          #####
D1.02:          #####          #####
D1.03:          #####          #####
D1.04:          State / Zip AA #####

TF.01:          Total Number of Customers Listed #####
```

The following pages describe the procedure for creating this report layout.

Entering Text Items

As with the screen painter, you create text items on a report by simply positioning the cursor at the beginning of the item, pressing **Return** and entering the required text. The painter automatically recognizes the entered text as a literal.

You must have sufficient space in the window to enter a text item. You cannot create a text item that is longer than the width of the window, but you can create two text items that span across two windows for long report headings.

The first text items you need to enter on this report extend from column 20 through to column 59. To create these items, this range of columns must be visible in the current window. Enter the text items by following these steps.

Action	Notes
Press ⏏ and then press Return	The top left section of the report is displayed in the window.
Press Window Keys	The window keys function keys are displayed
Press Right Window	This moves the window 10 columns to the right. This displays the range of columns needed to input the next text item.
Press ▶	Keep doing this until the cursor is located in report Column 35. (Press Return to update the cursor position display field.)
Press Return	The painter enters the <i>text input</i> mode.
Type Customer File ...	
Press Return	This terminates the entry of the text item.
Press Shift+⏏ and then press Return	The bottom right section of the report is displayed in the window.
Press ▲	Keep pressing this key until the cursor is located on line P1.01.
Press ▶	Keep pressing this key until the cursor is located at column 87. (Press Return to update the cursor position display.)
Press Return	The painter enters <i>text input</i> mode.
Enter Page: Move to P1.02 col 87	

Assumes Report is larger than 80 columns
Report Defn Set to Use Display which is 80 char

Press **Return**

Enter

Date:

This completes entry of the text items that appear on the two header lines of the report. Now you can enter the remaining text items for the C1.01, D1.04, and TF.01 lines. Use the cursor keys to move the cursor about the window, and the window keys to move about the report when you hit the window edge.

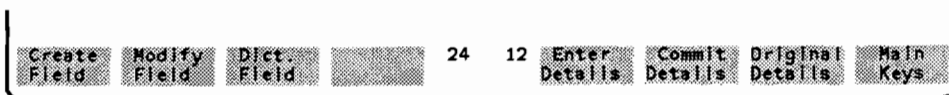
Use the cursor positions listed below as a guide.

Line Group	Text	Cursor Position
C1.01	#	003
	Name	019
	Address	055
D1.04	State / Zip	055
TF.01	Total Number of Customers Listed	010

Defining Data Fields

Next, you need to create the data fields on the report. This can be done in a number of ways with the field keys.

From the main function key set, press the **Field Keys** function key. The following function keys are displayed:



The **Create Field** and **Dict. Field** function keys allow you to define data fields on the report line. The function keys **Enter Details**, **Commit Details**, and **Original Details** allow you to define the data that is printed in the field and any formatting or totalling that is required for this field.

Painting a Field

Position the cursor at line P1.01, column 71 and complete the following steps.

Action	Notes
Press Create Field	An inverse video area extends to the right of the cursor in the report painter window.
Type NNNN	This specifies a numeric field 4 digits long.
Press (Return)	This completes entry to the field. HP ALLBASE/4GL should display the message: <i>Item passed validation</i>

The character you use to define the field identifies the type of data that the field can contain. You can use the following characters:

Character	Field Type
A	Alphanumeric
N	Numeric
Z	Zero suppressed numeric

The message that you receive confirms that you have entered a valid field definition. Now enter the details for this field using the **Enter Details** function key.

Position the cursor anywhere on the field (the current field will be highlighted) and press the **Enter Details** function key. The painter moves the cursor to the *Field Name/Literal* field in the lower portion of the screen. Now step through the fields and make the necessary entries.

Entering the Field Details

Field Name/Literal. Enter ***PAGENO**

This entry specifies the source of data for the report field. In this case, it is the communication area field *PAGENO. This field contains the number of the report page currently being printed.

You will notice that when the current report field is a literal, this field contains a message: *Literal - see line format.*

Edit Code. Accept the default value of **N**

This field has the same meaning as any other HP ALLBASE/4GL edit code field. The value in this field defaults to either N or X depending on the character you used to paint the field.

Justification. Accept the default value of **R**

Once again this defaults to a standard justification code depending on the character used to paint the field.

Pad Character. Enter **0**

This character fills blank spaces in the field after it has been justified in the correct direction.

Underline. Leave blank.

This is similar in meaning to the same field on the report line header screen. However, the underlining or overlining defined here only applies to this report line field, not to all the fields on the line.

Print Field Control. Accept the default value of **A**

This entry defines when this field should be printed. **A** means always, or each time the line is printed. **C** means only when the value in the field changes. This feature is useful in the detail lines (D1 – D9), since printing the information for a field only when its value changes makes the report appear less cluttered.

Total Number. Leave blank.

This is the automatic totalling facility for totalling fields down the report. The number you enter here relates to the communication area fields *TOTALS(1) to *TOTALS(16). When left blank it indicates that no totalling takes place for this field.

Cross Add Number. Leave blank.

This is the automatic totalling facility for use across the print line. The number you enter here relates to the communication area fields *CROSS(1) to *CROSS(5). When left blank it indicates that no totalling takes place for this field.

Having entered the details for this report line field, commit them by pressing the **Commit Details** function key. The painter returns the cursor to the report line field. The details for the field remain displayed. To display the details for a field, press **Return** while the cursor is on the field.

In summary, to enter details for a field press the **Enter Details** function key while the cursor is on the appropriate report line field. To return to the painter mode, press the **Commit Details** function key to accept the values you have entered, or press the **Original Details** function key to abandon that entry of field details. The field's original details will be restored after you press the **Original Details** function key.

Next create the field that displays the print date of the report. Position the cursor at line P1.02, column 71 and perform the following steps:

Action	Notes
Press Create Field	An inverse video area extends to the right of the cursor in the painter window.
Type AAAAAAAA	This specifies an alphanumeric field 8 characters long.
Press (Return)	The painter displays the message: <i>Item passed validation.</i>
Press Enter Details	This will move the cursor to the field details portion of the screen.
Enter *DATE	This is a reference to the communication area field that contains the current formatted date.
Enter D in the edit code field.	This formats the date according to the system wide date format.
Press Commit Details	No further details are needed for this field. Commit the details for this field and return the cursor to the painter window.

The page heading and column heading lines are now complete. You can now complete the total line.

Position the cursor at line TF.01, column 45 and create a four digit numeric field that refers to the communication area field *COUNT(1). This field maintains a count of the number of times the D1.01 line is printed.

The fields for the D1 to D4 lines will be created by referring to the dictionary field specifications.

Using Dictionary Field Specifications

You can create a data field on a report using a dictionary field specification name. The painter creates a field of the appropriate length and type. The painter also automatically constructs a data source field for the report line field. If the source field is not correct, you can easily correct it.

This is the prompt you will see when entering the field specification name:

Enter dictionary name							
customer_no				24	12	Enter	Commit
Create	Modify	Dict.				Details	Details
Field	Field	Field					Main
							Keys

Position the cursor at line D1.01, column 1 and perform the following steps:

Action	Notes
Press Dict Field	You are prompted to enter the name of a dictionary field specification.
Type <code>customer_no</code>	This is the dictionary field specification name for this report field.
Press Return	The report painter automatically creates a field of the appropriate length and type on the screen. It creates a data source in the <i>Field Name/Literal</i> field.

This field source defaults to the form *F-field name.primary file name*. In this case the source created will be *F-customer_no.customer*.

This is exactly what you require, so the report line field is complete. Position the cursor at line D1.01, column 10 and follow the same procedure to create a field using the field specification name *customer_name*. Position the cursor at line D1.01, column 45 and create a field using the field specification name *address*. Unlike text items, you can create a field that does extend beyond the boundary of the window when using an existing field specification. You only need to have the start of such an item inside the current window.

The source specifier for this field is not correct as it has an incorrect subscript value. Edit this specifier by performing the following actions when the cursor is located on this field:

Action	Notes
Press Enter Details	The cursor moves to the <i>Field Name</i> field.
Change (00) to (01)	This refers to the first occurrence of the address table in the <i>customer_rcrd</i> record.
Press Commit Details	This commits these details and returns the cursor to the field on the report painter window.

The next step is to create the address lines on the next two D1 lines. To do this, copy the address field you have already created and edit its source to obtain the correct subscripts.

Position the cursor at line D1.01, column 45. This is the beginning of the address line that you have already defined. Press **Main Keys** so that you have the main function keys displayed as shown below. Then follow these steps.



Action	Notes
Press Copy	A prompt is displayed asking you to move to the destination and then press Return .
Press ▼	This moves the cursor down to the next line.
Press Return	The item is copied, starting from the current cursor position.
Press ▼	The cursor moves to the next line.
Press Return	You are still in copy mode, so the field is copied again.
Press End Function	This terminates the copy mode, so you can now edit the field details.
Press Field Keys	This displays the field function keys.
Press Enter Details	This moves the cursor to the <i>Field Name</i> field so you can edit the subscript value displayed here.
Change (01) to (03)	This refers to the third occurrence of the address field.
Press Commit Details	This commits the new value then returns the cursor to the painter field.

Move up to the second line of the address and follow the same procedure to change the subscript in its field name from (01) to (02).

Now create the last two fields on this report. At line D1.04, column 67, create a field using the field specification name *state-code*. Similarly, at column 70 create a field using the field specification name *zip-code*. This completes the painting of this report.

Saving a Report

To save this report image, return to the main set of function keys by pressing the **Main Keys** function key. Then press the **Exit** function key. This automatically saves the report under the name that it was called up with. You are then returned to the reports menu. Now generate this report as explained in the next lesson.

Summary

In this lesson you used the report painter to create the image of a report.

The report painter allows you to position report fields and literals on the lines of the report. The painter also allows you to specify the following details of fields on report lines:

- The source of data to be printed in the field.
- The edit code and justification for the field.
- The automatic totalling performed for the field.

To create a literal on a report line, position the cursor at the starting point of the literal, press **Return** and enter the text for the literal. To create output fields on a report line, you can define the field directly, or use a dictionary field specification to define the field.

Lesson 13 – Generating and Running a Report

Objectives

In this very brief lesson, you will generate the report you have just defined. You can then run the report using the application testing mode.

This lesson also introduces the menu bypass system as a quick way of testing application items in the application testing mode.

Report Generation

A report is another item that requires generation. You cannot generate a report before you have defined all the items for the report.

Activate the *Header* item from the reports menu. The report header screen has a function key that allows you to generate the report after you have defined all of its constituent parts. You will also find this function key on some of the other report screens.

On the report header screen, enter the name of the report to be generated, if not already displayed, then press the **Generate Report** function key. As with the generation of other items, HP ALLBASE/4GL displays a message verifying that the report is being generated.

If you have specified everything correctly, the report will be generated successfully. However, if there are any errors, HP ALLBASE/4GL displays an error screen.

The error details displayed will specify which part of the report is in error (for example header, sorting, links selection, line header, or the line details). A listing below the error details shows the element that was found to be in error, and the actual error message.

Note the error, then go to the appropriate screen and correct the problem. When you have corrected it, generate the report again.

When the report has been successfully generated test it.

Executing a Report

There are two possible ways to test this report at this stage. The quickest method is to go into *Application Testing* and then use the developer menu bypass facility to execute any item from a menu. Otherwise you can create a menu item on the *main* menu of your application to run the report. Both methods are described here.

Testing by Menu Bypass

This is the quick way available to you as a developer. This method is not available to end users. From the developer main menu, follow the steps below to test the report:

Action	Notes
Activate <i>Application Testing</i>	This starts up your application in developer testing mode.
Press F7 and then press Return	This is the developer menu bypass facility command. You are prompted for the action you wish to execute.
Type R-customer_rept	This refers to the report <i>R-customer</i> that have just created.
Press Return	This executes the <i>customer_rept</i> report.

When the report commences, HP ALLBASE/4GL displays a message stating that the report is being prepared. The report is then displayed on the screen. When the report is finished, the menu screen is redisplayed.

Testing from a Menu

This is an alternative way of running the testing procedure. It involves creating a new menu item on the main application menu to execute the report. To do this, start up the screen painter and follow these steps:

Action	Notes
Enter <code>main</code>	This is the name of the screen that you want to modify. The painter displays the screen in its current state.
Press <code>Main Keys</code>	This displays the main painter keys.
Move cursor to line 9, column 32	This provides adequate spacing away from the prompt that is already there.
Press <code>Action</code>	This displays the action item prompt at the base of the screen.
Enter <code>R</code>	This retrieves the report action and then moves the cursor to the next field.
Enter <code>customer_rept</code>	This creates an action item that executes the <code>customer_rept</code> report. The prompt is the short description of the report from the report header screen.
Press <code>Return</code>	This terminates the creation of the action item prompt.
Enter <code>Customer Report</code>	This creates the text for the action item prompt.
Press <code>Exit</code>	This saves the screen and exits from the painter.

Having created this menu item, go to the *Application Testing* option on the main menu and run the report by activating the report option on the application menu. When the report commences, HP ALLBASE/4GL displays a message stating that the report is being prepared. The report is then displayed on the screen. When the report is finished, the menu screen is redisplayed.

Summary

In this lesson you generated and executed a report.

During generation, HP ALLBASE/4GL resolves references to data items in the report and also produces an executable form of the report.

After you generated the report, you tested it.

This lesson introduced the menu bypass facility in the application testing mode. The menu bypass facility allows you to execute an item directly. You can do this by entering the character / at any menu. HP ALLBASE/4GL displays a message

asking you to enter an action prefix, and an action name. The menu bypass system allows you to execute any action that can be called from a menu.

From Here On


You have now developed a simple working HP ALLBASE/4GL application.

The lessons in the next chapter show you how to include a number of enhancements to improve this application.

Intentionally Blank

Contents

Chapter 9: More Features

What Changes are Required?		9 - 1
Application Structure		9 - 2

Lesson 14 – Modifying a Logic Block

Objectives	9 - 4
The customer_proc Process	9 - 4
Modifying a Process	9 - 6
Summary	9 - 10

Lesson 15 – Storage Items

Objectives	9 - 11
Defining Application Titles	9 - 11
Defining Variables	9 - 13
Defining Numeric Constants	9 - 14
Defining Alphanumeric Constants	9 - 16
Summary	9 - 17

Lesson 16 – Function Keys

Objectives	9 - 19
Function Keys	9 - 19
Defining Function Keys	9 - 19
Four New Functions	9 - 22
The Customer Function	9 - 24
Summary	9 - 26

Lesson 17 – Messages

Objectives	9 - 28
Messages	9 - 28
Creating a Message	9 - 29
Summary	9 - 33

Lesson 18 – Validation Tables

Objectives	9 – 34
Validation Table Definition	9 – 34
Summary	9 – 37
Conclusion	9 – 37

9 More Features

So far you have developed a simple application that uses a process, a screen, a function and a report. Together, these items allow you to enter data into the *customer* file, and to produce a simple report of the data contained in the file.

The next five lessons show you how to incorporate some additional features into this application so it offers more functionality and user feedback.

These lessons describe the techniques for adding the following items to the application that you have developed so far:

- Function keys.
- Messages.
- Storage items such as:
 - Variables.
 - Constants.
 - Application titles.
- Validation tables.

To use these features, you must expand the logic blocks that you have already written. You also need to modify one of the existing field specifications.

What Changes are Required?

Currently, the *customer_scrn* screen is very simply defined and driven. It doesn't allow a user to delete a record from the *customer* file, and it doesn't provide much operational information for the user.

Apart from the data displayed on the screen, there is very little to indicate that a record is being added or modified. In addition there is no provision for trapping any file access errors that may occur.

The enhancements that you define in this chapter allow the user to specify that a record is to be added, modified, deleted or reviewed. This is achieved by using functions that are called up from a set of function keys that are associated with the *customer_scrn* data screen.

When the user selects the *Customer Details* option on the main menu, the extended application will present the following screen.

The screenshot displays a terminal window titled "Training Application" with the subtitle "Customer Details" and a window identifier "customer_scrn". The top left corner shows the time "15:27:24" and the top right corner shows the date "10/11/88". The main area contains four labeled input fields: "Number" (a short text field), "Name" (a long text field), "Address" (a long text field), and "State/Zip" (a short text field). At the bottom, a menu bar contains the following items: "Add *", "Modify", "Delete", "Review", "6", "40", "System Keys", "Commit Data", "Help", and "Previous Menu". The asterisk next to "Add" indicates it is the currently selected function.

Notice that this screen uses the first four function keys to enable the user to select the required action. This screen indicates the selected mode by displaying an asterisk (*) in the screen image for the function key.

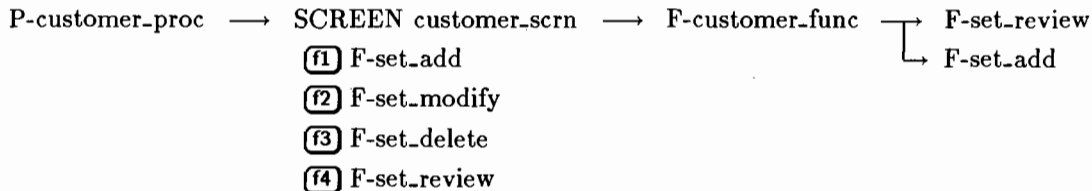
In addition, the user receives messages indicating that records can be added, modified, deleted, or reviewed as selected. On completion of a file update, the user is informed if the update was successful or not.

Application Structure

The following diagram shows the structure of the extensions to the application.

Main Menu

↓



This application still uses a process called from the main menu item. The process displays the *customer_scrn* screen, and then handles the file updates after the user terminates the screen.

The *customer_scrn* screen still calls the *customer_func* function as an after function on the *customer_no* field. This function reads the customer file for a record matching the customer number entered by the user.

The *customer_scrn* screen also uses four function keys to allow the user to determine the file update mode as *Add*, *Modify*, *Delete*, or *Review*.

The mode of update is determined by the setting of four switches. These switches are turned *on* and off appropriately whenever the user presses one of the function keys. The file is then updated according to these switch settings.

The reason for using switches to indicate the update mode is twofold. It's easy to set and test the appropriate mode, and you can display the status of these switches in the function key labels. This makes it easy to provide some additional feedback for the user.

Lesson 14 – Modifying a Logic Block

Objectives

In this lesson you will use the process details screen to expand and modify the steps in the *customer.proc* process. You will use this screen to insert new logic steps, and modify existing logic steps. This lesson also introduces the user switches, and some of the commands for manipulating switches.

The customer_proc Process

The *customer.proc* process needs to be modified from the point of exit from the *customer.scrn* screen. When the user terminates the screen, the process checks which mode the user has selected, and then updates the file accordingly. After the file update, the process displays a message to confirm its success. If an error occurs during the update, the process displays an appropriate message.

The *customer.proc* process uses four of the eight available user switches. The switches are set by the user, from the functions keys while the screen is active. The user switches are referred to by number (1 to 8) and their interpretation, when on, is shown below:

- | | |
|---------------------|---------------------|
| 1 = Add a record | 2 = Modify a record |
| 3 = Delete a record | 4 = Review a record |

The modified *customer.proc* process reads like this:

```
1 MODE *WRITE customer
2 SCREEN customer.scrn
3 IF 3 *ON THEN ENTER 8
4 IF 1 *ON | 2 *ON THEN ZIP ELSE TOP
5 FILE *WRITE customer ; ENTER 11
6 IF 1 *ON THEN MESSAGE add_ok ELSE MESSAGE modify_ok
7 TOP
8 FILE *DELETE customer ; ENTER 11
9 MESSAGE delete_ok
10 TOP
11 MESSAGE file_error
12 TOP
```

The following paragraphs explain what each step in the process does.

1. This step specifies that the process can write to the *customer* file.
2. This step displays the *customer_scrn* screen.
3. This step tests for *delete* mode and passes control to step 8 if it is set. That is, when switch 3 is on.
4. This step tests to see if *add* or *modify* mode is selected. The symbol `|` is the logical OR operator. If either mode is set, control passes to the next step (*ZIP* is a no-operation command). If both are not set then *review* mode or no mode has been selected, and control returns to the first step of the process.
5. This step writes the current *customer* file buffer contents to the file. Since the *customer* file has unique primary keys, the `*WRITE` operation can be used for both adding a new record or modifying an existing record. If an error occurs during the `FILE` command, the clause following the semicolon takes effect. If an error occurs, control passes to step 11.
6. By now the file has been successfully updated. This step *informs* the user that the update has occurred. If *add* mode is on, this step displays the *add_ok* message. Otherwise, it displays the *modify_ok* message.
7. This step passes control back to the first step in the process, allowing the user to continue.
8. Control passes to this step if switch number 3 is set on. That is, *delete* mode is selected. This command deletes the current record from the *customer* file. As in step 5, the error clause following the semicolon determines the next step if an error occurs. In this case, control passes to step 11 if an error occurs.
9. This step displays the *delete_ok* message to the user indicating that the record has been successfully deleted.
10. This step passes control to step 1.
11. This step displays the message *file_error* to indicate to the user that the file update has not succeeded.
12. This step passes control to the first step of the process.

The next steps show you how to *modify* the existing process.

Modifying a Process

This part of the lesson shows you how to use facilities that allow you to modify logic blocks. You use the same methods to modify processes and functions.

Call up the process details screen and follow these steps:

Action	Notes
Enter <code>customer_proc</code>	This displays the <i>customer_proc</i> process in its current state and places the cursor in the step number field.
Enter 3	The cursor moves to the action field.
Enter I	This indicates that you want to insert a step before step number 3. The current steps 3 and 4 move to 4 and 5, and step 3 becomes blank. Then the cursor moves to the command field.
Enter IF	This displays the IF command window. The fields on this window are explained in more detail below.

Using the IF Window

Condition 1

The first three fields of the IF window establish the first condition that the IF command tests. In many cases it is the only condition.

Data Name 1. Enter 3

This is the name of the object to be tested. In this case, the command tests user switch number 3.

IF Test. Enter *ON

This is the test to be performed. In this case the test is only on a single item. *ON is a special condition to test switches. The test is resolved as true if the switch is on.

Data Name 2. Leave blank.

Where an IF Test compares two operands with a relational operator such as =, <, >, or <>, you must enter the name of the second operand in this field. In this case, the IF command only requires one operand, so leave this field blank.

AND/OR. Leave blank.

The IF command can test two conditions and combine the results using a logical AND or a logical OR connective. The symbols & and | represent the AND and OR connectives respectively. When you leave this field blank, the cursor skips the next three fields.

THEN Statement

When you complete the condition fields, a *THEN* prompt appears. The cursor is placed in the field to the right of this prompt. This is where you enter the commands to be performed when the test is resolved as true. You can execute any command at this stage except IFLOOP, SELECT, or another IF statement. For this example enter the following command:

ENTER 8

You can specify multiple commands by separating the commands with a semicolon. For example, you could enter a string of commands in this field like this:

MESSAGE added_ok ; ENTER 10

ELSE Statement

After you enter the THEN statement the window displays an *ELSE* prompt and positions the cursor in the field beside the prompt. This is where you enter the commands to be performed if the test is resolved as false. This entry follows the same format as the THEN statement. However, it is not a required field. If you don't enter any ELSE commands, control passes to the next step in the process or function if the test is resolved as false. In this case, leave the field blank because we want control to pass to step 4.

Completing the Window

The IF command is now complete. Press the **Commit Data** function key to commit it. HP ALLBASE/4GL clears the window and displays the new command at step 3. The cursor returns to the *Step Number* field and the step number is incremented to 4.

The Next Step

Now you can insert the new step 4. Follow the same procedure as outlined above but this time use step number 4. The command is:

```
IF 1 *ON | 2 *ON THEN ZIP ELSE TOP
```

You will need to make an entry in the *AND/OR* field, and the *condition 2* fields. You will also need to enter the *ELSE* statement. Once the command is complete, commit it and go on to the next step.

Continuing the Change

Your process should now look like this:

```
1 MODE *WRITE customer
2 SCREEN customer_scrn
3 IF 3 *ON THEN ENTER 8
4 IF 1 *ON | 2 *ON THEN ZIP ELSE TOP
5 FILE *WRITE customer
6 TOP
```

The next task is to add two more steps to the end of the process. The procedure described here deliberately adds these steps in the wrong place. This is done to show you how HP ALLBASE/4GL updates step number references when you insert or delete lines from a logic block.

At steps 7 and 8 use the *Add* action to add the following lines.

```
MESSAGE file_error
TOP
```

You can now go on to modify step 5 and add the remaining steps to the process.

Modifying a Step

You can now change an existing step. Position the cursor at the *Step Number* field and follow these steps.

Action	Notes
Enter 5	The cursor moves to the action field.

Press Return	Since the default action for an existing step is <i>Change</i> , the current contents of the step are displayed in a FILE window and the cursor is positioned in the first field of the window.
Press Tab six times	This moves the cursor through the next five fields to the <i>On Error</i> field.
Enter ENTER 7	This is the file command error clause. Don't worry that you are entering 7 and not 11 as listed originally. The error step has already been created at step 7. HP ALLBASE/4GL automatically alters this value as you insert the remaining lines in this logic block.
Press Commit Data	Commit the step and display the modified command at step 5. Note that HP ALLBASE/4GL inserts a semicolon before the error command you just entered.

Completing the Changes

At step 6 insert the following line:

```
IF 1 *ON THEN MESSAGE add_ok ELSE MESSAGE modify_ok
```

At step 8 insert: **FILE *DELETE customer ; ENTER 9**

At step 9 insert: **MESSAGE delete_ok**

At step 10 insert: **TOP**

Position the cursor on the *Action* field, regardless of step number, and display the whole process on the screen.

Action	Notes
Enter L	The <i>L</i> action <i>lists</i> the current logic block to the screen. When you press Return , HP ALLBASE/4GL clears the screen and lists the entire process. Any lines that are too long to fit on the screen will be wrapped around to the next line.

Note that step 3 refers to the wrong step:

```
IF 3 *ON THEN ENTER 12
```

When first created, this step referred to step 8 which did not exist. As a result, the adjustments made to this step number have been out of synchronization. However, it's a simple task to go back and change the line to read correctly. First follow these steps to return to the process details screen.

Action	Notes
Press Return	This ends the listing of the process and redisplay the process details screen. If the process requires more than one screen to list all the steps, the second screen is displayed. Pressing Return again takes you to the process details screen.
Press Shift+Tab then press Return	This returns the cursor to the <i>Step Number</i> field.

Now modify step 3 to refer to step 8, not step 12.

Press the **Generate Process** function key to generate the modified *customer_proc* process. If you have entered the process exactly as it's listed at the beginning of this section, you won't receive any generate error messages.

Summary

In this lesson you used the process details screen to modify a process logic block.

The process details screen allows you to:

- Modify an existing logic step.
- Insert a new logic step.
- Delete a logic step.
- Display the entire logic block.

The process details screen automatically changes references to other steps in commands such as ENTER if you add or delete steps.

The function details screen provides exactly the same facilities for modifying function logic blocks.

Lesson 15 – Storage Items

Objectives

In this lesson you will define a number of storage items.

These items are used in logic blocks you will define later. The items you will create in this lesson are:

- An application title.
 - *application_name*
- A variable.
 - *file_status*
- A numeric constant.
 - *record_not_found*
- Two alphanumeric constants.
 - *record*
 - *no_record*

Defining Application Titles

An application title is a literal string that can be displayed on any screen within an application or printed on reports. Any change made to an existing application title is immediately reflected through the whole application and related versions that use the application title. You can only reference an application title directly with the screen painter.

Menu Path

Dictionary, Storage Items, Titles

Description

An application title is defined by its name and contents. The contents are stored as an alphanumeric string of length equal to the number of characters specified in the string.

Developer		Application Titles		titles				
Title Name	application_name							
Content	Training Application							
Variable	Calc. Items	Numeric Constant	AlphaNum Constant	10 30	System Keys	Commit Data	Help	Previous Menu

Entering the Field Values

Title Name. Enter `application_name`

This is the name of the application title that is used by the application.

Content. Enter `Training Application`

This is the actual application title. Any trailing spaces are stripped when you commit the title.

Press the `Commit Data` function key to create this application title.

When you next run your application you will see that the screens contain the new definition of this item.

Press the `Previous Menu` function key to return to the *storage items* menu.

Defining Variables

You can now create a variable called *file_status*. This variable is set after a file access to indicate if a record is current or not.

Menu Path

Dictionary, Storage Items, **Variables**

Description

Variables have a fixed length and type. When an application starts, they are initialized to spaces for alphanumeric variables and to zero for numeric variables. They are not initialized at any other time.

Developer		Variables		variables					
Variable Name		file_status							
Length		1							
Edit Code		X (X/A/U/K/N/S/Q/D/T)							
Number of Decimal Places		:							
Pad Character		:							
Work Area	Calc. Items	Numeric Constant	AlphaNum Constant	13	48	System Keys	Commit Data	Help	Previous Menu

Entering the Field Values

Variable Name. Enter `file_status`

When referred to in the application, a variable is always prefixed with *V-*. The full reference to this variable is *V-file_status*.

Length. Enter `1`

The length of a variable is fixed, and the maximum possible length is 99 characters.

Edit Code. Enter `X`

This edit code has the same definition as all other edit codes.

Number of Decimal Places. Leave blank.

Variables can use up to nine decimal places if they are numeric.

Pad Character. Leave blank.

HP ALLBASE/4GL automatically justifies a variable when it receives a value. Alphanumeric variables are left justified, and numeric variables are right justified. This field allows you to specify a pad character which is used to fill the remaining spaces in the variable after it has been justified. The pad character must be a valid character for the edit code specified.

Press the **Commit Data** function key to create this variable.

Press the **Previous Menu** function key to return to the storage definition menu.

Defining Numeric Constants

Menu Path

Dictionary, Storage Items, **Numeric Constants**

Description

A numeric constant is defined as a signed or unsigned number of fixed value with a fixed number of decimal places. A numeric constant can be used to initialize other variables and fields, or it can be used as a fixed comparative value when you want to test another field.

Developer		Numeric Constants		numerics				
Numeric Constant Name	record_not_found							
Value	19111							
Edit Code	N (N/S)							
Number of Decimal Places	0							
Variable	Calc. Item	Work Areas	AlphaNum Constant	7 69	System Keys	Commit Data	Help	Previous Menu

Entering the Field Values

Numeric Constant Name. Enter `record_not_found`

When referred to in the application, a numeric constant name is prefixed with *N-*. The full reference to this constant is *N-record_not_found*.

Value. Enter `19111`

The value of this numeric constant is the file error value returned by the KSAM data manager when a requested record is not found. In this application it is used to test the file return status value.

Edit Code. Displayed value `N`

This is a display only field that shows the edit code of the variable. If the variable value includes a `+` or a `-` sign, the display in this field is `S`.

Number of Decimal Places. Displayed value `0`

This is a display only field that shows the number of decimal places in the variable. Numeric variables can have up to 9 decimal places.

Press the `Commit Data` function key to create this numeric constant.

Defining Alphanumeric Constants

Alphanumeric constants are used in the same way as numeric constants, but they contain alphanumeric data.

The screenshot shows a dialog box titled "Alpha-Numeric Constants" within a "Developer" environment. The dialog has a title bar with "Developer", "Alpha-Numeric Constants", and "constants". Inside, there are two input fields: "Constant Name" with the text "record" and "Value" with the text "Y". At the bottom, there is a menu bar with the following items: "Variable", "Calc. Items", "Numeric Constant", "Titles", "10", "7", "System Keys", "Commit Data", "Help", and "Previous Menu".

Menu Path

Dictionary, Storage Items, Alphanumeric Constants

Description

An alphanumeric constant is defined in the same way as an application title. The contents are stored as an alphanumeric string of length equal to the number of characters entered in the string. Any trailing spaces are stripped away when you commit the constant.

Entering the Field Values

Constant Name. Enter `record`

When referred to in an application, an alphanumeric constant is prefixed with *C-*. The full reference to this constant is *C-record*.

Content. Enter `Y`

This is the entire content of this constant. It is a flag value indicating that a record has been found. You will see how it's used later.

Press the **Commit Data** function key to create this alphanumeric constant.

Alphanumeric Constant – *no-record*

Create a second alphanumeric constant called *no-record* containing the value `N`.

Press the **Previous Menu** function key to return to the storage definition menu.

Now you have completed the definition of the required storage items for the enhanced application.

Summary

In this lesson you defined the following storage items:

- An application title.
 - *application.name*.
- a variable.
 - *file.status*.
- a numeric constant.
 - *record.not.found*.
- two alphanumeric constants.
 - *record*.
 - *no.record*.

All storage items are defined using the storage definition menu in the dictionary.

Application Titles

Application titles are literal strings that can be used on screens and reports. Any change to an application title is immediately reflected on screens and reports that use the title.

Variables

All variables have a fixed length and type. The type of data a variable can contain is determined by its edit code.

HP ALLBASE/4GL initializes numeric variables to zero, and alphanumeric variables to spaces, when the application starts. They are not initialized at any other time.

Constants

You can define numeric constants and alphanumeric constants. Numeric constants can contain numeric values and the characters `-` and `+`.

Alphanumeric constants are literal strings, and can contain any printable characters.

Lesson 16 – Function Keys

Objectives

In this lesson you will define a set of terminal function keys. These function keys will be associated with the *customer_scrn* data screen.

You will also define some new functions for use with the function keys, and you will expand the existing *customer_func* function.

Function Keys

A function key set can contain up to eight function keys. Each key can execute an item when the user presses the key. Some of the items that can be called from a function key are:

- A function.
- A process.
- A report.
- A screen.
- A help screen.
- Another function key set.

Any type of screen can use a function key set. A function key set is defined separately from the screen, so you can use one function key set with a number of screens. The function key set associated with a screen is named on the screen header screen.

When you define a function key set you can define up to three items for each function key. These items are an action, the text for the function key label, and optionally, a user switch to be assigned to the function key. The status of the user switch is displayed in the function key label.

Defining Function Keys

The function key set you are about to define allows the user to set the appropriate file update mode of *add*, *modify*, *delete*, or *review*. Each function key calls a function to set the appropriate switch and display a suitable message to the user.

Menu Path

Screens, **Function Keys**

Description

This screen allows you to define the actions, labels, and associated switches for a set of function keys.

Function Keys		Label Text		Switch
Function Key	Action	Line 1	Line 2	Number
f1	F-set.add	Add *		1
f2	F-set.modify	Modify*		2
f3	F-set.delete	Delete*		3
f4	F-set.review	Review*		4
f5	I-HELP	Help		
f6	I-COMMIT	Commit	Data	
f7	K-global-keys-1	System	Keys	
f8	I-PREVMENU	Previous	Menu	

Screen Header	Screen Painter	Field Details	15	21	System Keys	Commit Data	Help	Previous Menu
---------------	----------------	---------------	----	----	-------------	-------------	------	---------------

Entering the Field Values

Function Keys. Enter `customer_keys`

This is the name of the function key set within the application.

When you press **Return**, HP ALLBASE/4GL supplies a default set of function key definitions. These default definitions satisfy the basic needs of all screens.

You can create your own default set of keys by defining a function key set called `default_keys` in your application. Refer to the *HP ALLBASE/4GL Developer Reference Manual* for more information.

The rest of the screen is divided into eight rows, each containing four fields. There is one row for each key. The function key number is displayed to the left of these fields. The following steps show you how to define function key number 1.

Function Key f1

Action. Enter `F-set_add`

This is the action that HP ALLBASE/4GL performs when the user presses function key **f1**. The first character is the action type. In this case *F* indicates a function. When the user presses the key, the function *set_add* is executed. The *HP ALLBASE/4GL Developer Reference Manual* contains a list of the actions that can be called from function keys, and the prefixes for these actions.

Label Text Line 1. Enter `Add`

This is the first line of the function key label that appears on the screen. Try to center the word in the field on the screen. Don't enter the asterisk. HP ALLBASE/4GL enters it when you define the switch for this function key.

Label Text Line 2. Leave blank.

This is the second line of the function key label that appears on the screen.

Switch Number. Enter `1`

This is the user switch associated with the function key. When you enter this number, an asterisk appears at the end of the first line of the label text on the application screen.

When the function key set is displayed on an application screen, the asterisk shows the status of the switch. When user switch 1 is on, the asterisk appears. When the switch is off, the asterisk is cleared. If the second label line is not blank, the asterisk is displayed there instead. The asterisk overwrites any character already in the eighth position on either line.

To Complete the Screen

Define the next three function keys as shown below:

Function Key f2

Action	<code>F-set_modify</code>
Label Text Line 1	<code>Modify</code>
Switch Number	<code>2</code>

Function Key f3

Action	<code>F-set_delete</code>
Label Text Line 1	<code>Delete</code>
Switch Number	<code>3</code>

Function Key f4

Action	F-set_review
Label Text Line 1	Review
Switch Number	4

Leave the default definitions for the remaining keys. Press **Commit Data** to create this function key set.

When you have committed the function key set, press the **Previous Menu** function key to display the screen development menu. Select and activate the item *Header*, and display the header information for the *customer_scrn* screen.

Alter the name *customer_scrn* in the *Function Keys* field to *customer-keys*. Press **Commit Data** to attach the new function keys set to the screen.

Next, create the four functions called by these function keys.

Four New Functions

To complete the procedural aspects of this enhancement, you must create the four simple functions called by the function keys you have just defined.

These functions all have the same format. Each function tests the appropriate file status and sets the user switches accordingly.

Function – set_add

The first function is the *set_add* function which sets the update mode to add. This is what it contains:

```

1 IF V-file.status <> C-no-record THEN MESSAGE no-add ; EXIT
2 OFF *ALL
3 ON 1
4 MESSAGE add
5 EXIT

```

This is how the function operates:

1. This step tests the variable *file_status* to ensure that the record to be added does not already exist. The value in *file_status* is set when the *customer_func* function initially reads the record. (You will soon modify the *customer_func* function to include this logic.) If the record does exist, this function displays a message informing the user that the record already exists. The function then exits with no change to the switches.

2. This step sets all the user switches to *off* to place the user switches into a known state. HP ALLBASE/4GL automatically refreshes the function key labels to show the new state of the switches.
3. User switch 1 is set on. This switch specifies that the user has selected *add* mode. HP ALLBASE/4GL automatically refreshes the function key labels again to show the new switch status.
4. This step displays a message telling the user that the record may be added.
5. The function exits.

Create this function now. Before you can enter the details for the function, you must define the function header. Remember to generate the function after you create it.

The next three functions are very similar. You can create these functions by using the utilities menu catalog copying screen to copy the original *set_add* function to the new functions. You can then modify the new functions so they match the listings shown here. You must remember to modify the function headers as well as the function details if you use this method. Alternatively you may decide to create each function individually. Remember to generate each function as you complete it.

Function – *set_modify*

The *set_modify* function sets the update mode to *modify*. This is what it contains:

```

1 IF V-file_status <> C-record THEN MESSAGE no_modify ; EXIT
2 OFF *ALL
3 ON 2
4 MESSAGE modify
5 EXIT
```

Function – *set_delete*

The *set_delete* function sets the update mode to *delete*. This is what it contains:

```

1 IF V-file_status <> C-record THEN MESSAGE no_delete ; EXIT
2 OFF *ALL
3 ON 3
4 MESSAGE delete
5 EXIT
```

Function – set_review

The *set_review* function sets the update mode to *review*. This is what it contains:

```
1 IF V-file_status <> C-record THEN MESSAGE no_review ; EXIT
2 OFF *ALL
3 ON 4
4 MESSAGE review
5 EXIT
```

The Customer Function

The *customer_func* function must be extended to suit the need of this application.

The *customer_func* function is still called as an after function on the *customer_no* field on the *customer_scrn* screen. The function now includes commands to detect and handle any file access errors that may occur, and it also includes commands to initialize the file and screen buffers before the user adds a new record.

When the user enters a customer number, the function searches the customer file for a matching record. If a record is found, the function sets a status variable, automatically invokes review mode, and displays the record on the screen. If the initial file read is not successful, the function tests the file exit status. If the read failed because a record was not found, the function invokes add mode, and clears the file and record buffers to allow the user to add a record. If the file access failed for any reason other than failure to find a matching record, the function displays a warning message.

The modified *customer_func* function contains the following commands:

```
1 FILE *READ customer *KEY= * ; ENTER 6
2 MOVE C-record V-file_status
3 VISIT set_review
4 SHOW *REFRESH
5 EXIT
6 IF *IOSTATUS <> N-record_not_found
   THEN MESSAGE file_error ; EXIT
7 MOVE C-no_record V-file_status
8 VISIT set_add
9 FILE *BUFFER customer
10 CLEAR *MAP S-customer_name S-zip_code
11 EXIT
```


The original two line function has been significantly expanded from its original format.

The following paragraphs explain the purpose of each of the steps in the modified function:

1. This step reads the customer file using the value in the current screen field as the key to read the file. The error clause passes control to step 6 if the file read fails for any reason.
2. At this stage, the function has successfully read a record. To indicate that a record has been retrieved, this step moves the value of the alphanumeric constant *record* to the variable *file_status*.
3. The VISIT command is a call to a function. In this case it calls the function *set_review*. This function turns the review switch on, and displays a message indicating that the record may be reviewed.
4. This is the only line from the original function which remains unchanged. It displays the current contents of the *customer* file buffer.
5. The function ends if a record was successfully read.
6. This step is executed if an error occurs in the file operation in step 1. It tests the contents of the communication area field *IOSTATUS which contains the file access return code. If it is not equal to the value of the numeric constant *record_not_found*, a message is displayed and the function exits. If the value in *IOSTATUS is equal to the constant *record_not_found*, control passes to the next step in the logic block. You gave the constant *record_not_found* the value of 19111 earlier.
7. This step moves the value of the constant *C-no-record* into the variable *file_status* to indicate that a record was not found.
8. Like step 3, this step visits a function. The function sets the add switch on and displays a message informing the user that a record can be added to the file.
9. The FILE *BUFFER clears the contents of the file buffer preparing it for new data. When this function is executed, the customer number, entered by the user, is held in the screen field buffer. The automatic data movement, from the screen field to the corresponding field in the file buffer, occurs after this function exits. Therefore, the customer number is moved to the customer number field on the customer file after the function finishes. This technique preserves the data entered by the user even though the function clears the file buffer.

10. This command clears all the fields on the screen from *customer.name* to *zip.code* to allow the user to enter data for the new record. The *MAP option indicates that the range of fields to be cleared is specified by screen field name and not by field number.
11. The function exits. The EXIT statement is not required at the end of each logic block. HP ALLBASE/4GL places an implicit EXIT at the end of the logic block if one does not already exist.

Using the logic block modification methods you learned when modifying the *customer_proc* process, you can now modify the *customer_func* function. Call up the original function with the function details screen, modify the function, and then generate it.

If you receive any generate errors, check that the storage items you defined previously are correctly named. This is the most likely source of error.

Summary

In this lesson, you defined a set of function keys.

A function key set can contain up to eight function keys which execute an item when the user presses a key. Some of the items that can be called from a function key are:

- A function.
- A process.
- A report.
- A screen.
- A help screen.
- Another function key set.

Each function key can have the following items defined for it:

- An action.
This is the item that is executed when the user presses the key.
- Label text.
This is the text that appears on the function key label on the application screen.

- A switch number.

You can associate a user switch with a function key. The status of the switch is shown by the presence or absence of an asterisk in the function key label.

You also defined a number of functions to complete the procedural logic for the application.

At this stage all of the new logic is in place. However, the messages cannot be displayed when they are required because you haven't created them yet.

The next lesson describes the procedure to create the messages used by these functions.

Lesson 17 – Messages

Objectives

In this lesson you will create a number of messages. These messages are called by the functions you have already created for this application.

Messages

HP ALLBASE/4GL uses five different types of messages. Each has a different effect on the application. The five message types are:

Type	Description
MESS	Message only.
QUERY	The user must enter a response.
WARN	Warning only, message plus terminal bell.
ERROR	The user must take action to continue.
ABORT	The application terminates when the user presses Return .

When you select the type of message to use in your application, you must decide what should happen in the application after the message is displayed.

A message telling the user that a record may be modified should be a MESS type message. It is for information purposes only.

A message to confirm that the user wants to delete a record should be a QUERY type message. The user must respond to a query message. You can test the user's response to a query message to determine the subsequent action taken by the application.

If the user makes an invalid entry, you should use an ERROR type message. In this case, the user cannot move off the current field without making a valid entry.

If the field is not crucial, the message could be a WARN type message. In this case the message is displayed along with a terminal bell sound. No specific response or action is required from the user.

If the application or the interface to the host operating system detects a fatal error you may choose to specify an ABORT message. In this case, the message text is displayed on the screen, and the application terminates when the user presses **Return**.

This application requires these 12 messages, referred to in the process and functions you have defined so far.

- When an unexpected file error occurs, an **ERROR** message is displayed. The message name is:
 - *file_error*
- These messages tell the user that an appropriate action may be taken. They are **MESS** type messages:
 - *add, modify, delete, review*
- These messages tell the user that an update action has succeeded. They are **MESS** type messages:
 - *add_ok, modify_ok, delete_ok*
- These messages are used to tell the user that a selected action may not be executed. They are **WARN** type messages:
 - *no_add, no_modify, no_delete, no_review*

Creating a Message

The first message you will create is the *add* message.

Menu Path

Dictionary, **Messages**

Description

A message can be constructed from a number of items. Most messages have a literal component. You can also include communication area fields, variables, file record fields, screen fields, and other items. The *HP ALLBASE/AGL Developer Reference Manual* contains a list of all of the items you can use in messages.

Developer	Messages	messages
Name	add	
Type	MESS (MESS/QUERY/WARN/ERROR/ABORT)	
Response Item		
Help Name		
Message Construction		
"Record may be added."		
Field Specs	Ranges	Tables
Help Screens	14	26
System Keys	Commit Data	Help
		Previous Menu

Entering the Field Values.

Name. Enter add

This is the name of the message within the application.

Type. Enter m

This is the type code for this message. Note that HP ALLBASE/4GL converts your entry to uppercase, and expands it to the full message type code. The different available codes are listed to the right of this field. You only need to enter the first character of the message type code.

Response Item. Entry not possible.

This field only applies to QUERY type messages. HP ALLBASE/4GL moves the user's response to the query into the field that you specify here.

Help Name. Entry not possible.

This is the name of the help screen that is displayed if the user presses the **Help** function key while the message is displayed. This option is only valid for QUERY, ERROR, and ABORT messages.

Message Construction. Enter "Record may be added."

This is the content of the message. Note that you must start and finish literal text in messages with double quotes ("). A message can be up to 160 characters long.

If it exceeds this length, it is truncated to fit in the message area at the base of the screen.

Press the **Commit Data** function key to create this message. Messages are automatically generated when committed. Once generated, a message is ready to be used in an application.

Message – *file_error*

Now create the *file_error* message. Locate the cursor on the *Name* field and follow these steps:

Action	Notes
Enter file_error	This is the name of the message.
Enter ERROR	This is an error type message. It requires the user to reenter the data in the current field.
Press Return	This time you have access to the <i>Help</i> field. Leave it blank for the time being. You can define a help screen for this message at a later time.
Enter "File error, code:"	This is the first part of the message.
Enter *IOSTATUS	The current value of the file return code is included in the message when it is displayed.
Enter " has occurred."	This is the final part of the message.
Press Commit Data	The message is generated automatically.

When HP ALLBASE/4GL displays this message, it joins the three components together. You must explicitly enter any spaces that you want to appear between each component. You can define the individual parts of the message on separate lines. If you include more than one item on a line, separate the items with spaces.

Now create the remainder of the messages. Only the essential information for each message is listed here.

Message – *modify*

Name	modify
Type	MESS
Contents	"Record may be modified."

Message – delete

Name	delete
Type	MESS
Contents	"Record may be deleted."

Message – review

Name	review
Type	MESS
Contents	"Record may be reviewed."

Message – add_ok

Name	add_ok
Type	MESS
Contents	"Record added successfully."

Message – modify_ok

Name	modify_ok
Type	MESS
Contents	"Record modified successfully."

Message – delete_ok

Name	delete_ok
Type	MESS
Contents	"Record deleted successfully."

Message – no_add

Name	no_add
Type	WARN
Contents	"Record exists. " "Cannot be added."

Message – no_modify

Name	no_modify
Type	WARN
Contents	"Record does not exist. " "Cannot be modified."

Message – *no_delete*

Name	<code>no_delete</code>
Type	<code>WARN</code>
Contents	"Record does not exist. " "Cannot be deleted."

Message – *no_review*

Name	<code>no_review</code>
Type	<code>WARN</code>
Contents	"Record does not exist. " "Cannot be reviewed."

All the messages necessary at this stage of application development are now complete.

The final step to the expansion of this application is to define a validation table to automatically validate state codes as they are entered by the user. You will do this in the next lesson.

Summary

There are five types of messages. Each has a different effect on the application. The five message types are:

Type	Description
MESS	Message only.
QUERY	The user must enter a response.
WARN	Warning only, message plus terminal bell.
ERROR	The user must take action to continue.
ABORT	The application terminates when the user presses Return .

A message can be constructed from a number of items. Most messages will have a literal component. You can also include communication area fields, variables, file fields, screen fields, and other items.

Lesson 18 – Validation Tables

Objectives

In this lesson you will define a validation table. A validation table is a table of discrete values that can be used to test the validity of user input into a screen field.

Validation Table Definition

A validation table is a table containing up to 51 values. HP ALLBASE/4GL automatically tests the data entered by the user against the values in the table.

Validation tables are implemented in much the same manner as validation ranges. The major difference is that a table contains a number of discrete values, while a range specifies the lower and upper limits to be applied to the field during validation.

When you associate a validation table with a screen field, HP ALLBASE/4GL automatically tests the user's input against the table when data entry is complete. If the value entered exists in the table, processing continues normally. If the entered value does not exist in the table, HP ALLBASE/4GL displays the message associated with the table. Processing then continues in accordance with the type of the message. If you don't define a message in association with the table, HP ALLBASE/4GL displays a system defined ERROR type message by default.

Note

In some cases you may need to validate input data against more than 51 values. You can use the CHECK logic command in an after function on the field to validate the entered data against a number of tables. Refer to the *HP ALLBASE/4GL Developer Reference Manual* for further information on the CHECK command.

The validation tables screen allows you to enter the table name, the message to be displayed if the validation fails, and the table values themselves.

Menu Path

Dictionary, Validation Items, Tables

Description

The table you define here validates the state code entered by the user to ensure that it is a valid US Postal Service abbreviation. When you have defined the table, you must redefine the *state_code* field specification to include the validation table. You will also need to regenerate the *customer_scrn* data screen to update the generated record for the screen.

Developer		Validation Tables		tables	
Table Name	state_code	Secured	N	(Y/N)	
Message Name	state_code_error				
(1) Table Values	(18) KY	(35) ND			
AK	LA	DH			
AZ	ME	DK			
AR	ND	OR			
CA	MA	PA			
CD	MI	RI			
CT	MN	SC			
DE	MS	SD			
DC	MO	TN			
FL	MT	TX			
GA	NE	UT			
HI	NV	VT			
ID	NH	VA			
IL	NJ	WA			
IN	NM	WV			
IA	NY	WI			
(17) KS	(34) NC	(51) WY			

Field Specs. Ranges Messages 22 58 System Keys Commit Data Help Previous Menu

Entering the Field Values

Table Name. Enter *state_code*

This is the name of the table within the application.

Secured. Accept the default value *N*

This designates whether this item is secured against an unauthorized developer changing the entry.

Message Name. Enter *state_code_error*

This is the name of the message that is displayed if the value entered by the user does not match any of the values in the table. You will define this message soon.

Table Values. Enter the following:

Only enter the two character abbreviation. The full name for each abbreviation is merely listed here for reference.

AL	Alabama	KY	Kentucky	ND	North Dakota
AK	Alaska	LA	Louisiana	OH	Ohio
AZ	Arizona	ME	Maine	OK	Oklahoma
AR	Arkansas	MD	Maryland	OR	Oregon
CA	California	MA	Massachusetts	PA	Pennsylvania
CO	Colorado	MI	Michigan	RI	Rhode Island
CT	Connecticut	MN	Minnesota	SC	South Carolina
DE	Delaware	MS	Mississippi	SD	South Dakota
DC	Dist. of Col.	MO	Missouri	TN	Tennessee
FL	Florida	MT	Montana	TX	Texas
GA	Georgia	NE	Nebraska	UT	Utah
HI	Hawaii	NV	Nevada	VT	Vermont
ID	Idaho	NH	New Hampshire	VA	Virginia
IL	Illinois	NJ	New Jersey	WA	Washington
IN	Indiana	NM	New Mexico	WV	West Virginia
IA	Iowa	NY	New York	WI	Wisconsin
KS	Kansas	NC	North Carolina	WY	Wyoming

Press the **Commit Data** function key to create the table.

When you have committed the table, include it in the field specification by performing the following steps.

Action	Notes
Press Field Specs.	This displays the field specification screen.
Enter state_code	This is the name of the field specification that has to be altered
(Tab) to the <i>Table</i> field	This is the field where you enter the name of the validation table.
Enter state_code	This is the name of the validation table.
Press Commit Data	This creates the new field specification.

Now go to the screen fields details screen and call up the *customer_scrn* screen. Call up field number 6 and you will see that the new field specification has been included automatically. Press the **Generate Screen** function key to incorporate the new field specification and table in the application.

Message – *state_code_error*

The last thing you need to do is to create the *state_code_error* message. Go to the message definition screen. If you haven't defined any other messages, tables, or ranges since you defined the validation table, the message name should default to *state_code_error*. If it doesn't, enter the correct name. Assign the message an ERROR type code and enter a message to inform the user that the state code just entered is invalid. Then commit the message to generate it.

Now test your application again and observe how the new items you have defined work together in your application.

Summary

Validation tables are implemented in much the same manner as validation ranges. The major difference is that a table contains a number of discrete values while a range specifies the lower and upper values to be applied to the field during validation.

When you associate a validation table with a screen field, HP ALLBASE/4GL automatically validates the user's input against the table when data entry is complete. If the value entered exists in the table, processing continues normally. If the entered value does not exist in the table, HP ALLBASE/4GL displays the message that you have associated with the table. Processing then continues in accordance with the type of the message. If you don't define a message in association with the table, HP ALLBASE/4GL displays a system defined ERROR type message by default.

Conclusion

This part of the *training* application is now complete.

The lessons in the next chapter show you how to develop the items needed to use multiple files in conjunction with an HP ALLBASE/4GL application. These lessons also introduce some more advanced screen handling techniques.

Intentionally Blank

Contents

Chapter 10: Further HP ALLBASE/4GL Facilities

The Applications	10 - 1
Product/Option Application	10 - 1
The Application Screens	10 - 2
Function Keys	10 - 3
Scrolling Options	10 - 3
Reporting	10 - 3
Processing Rules	10 - 4
Supplier Application	10 - 4
Module Building	10 - 4
The Application Screen	10 - 5
Function Keys	10 - 5
Application Structure	10 - 5
Product/Option Application	10 - 6
Supplier Application	10 - 7



Lesson 19 – Further Dictionary Items

Objectives	10 - 8
Dictionary Requirements	10 - 8
Field Specifications	10 - 9
Record Layouts	10 - 10
Data Files	10 - 11
Validation Ranges	10 - 11
Variables	10 - 13
Alphanumeric Constants	10 - 14
Numeric Constants	10 - 15
Summary	10 - 15

Lesson 20 – More Screen Techniques

Objectives	10 - 17
Application Screens	10 - 17
Main Menu	10 - 18
Product Menu	10 - 18

Painting the Screen	10 - 18
Product Data Screen	10 - 19
Painting the Screen	10 - 20
Screen Field Details	10 - 21
Option Window	10 - 22
Painting the Screen	10 - 23
Screen Field Details	10 - 24
The New Option Window	10 - 25
The Product Window	10 - 25
Summary	10 - 26

Lesson 21 – Extending the Application Logic

Objectives	10 - 27
Function Keys	10 - 27
The New Logic Blocks	10 - 28
The Product Process	10 - 29
Screen Field Logic	10 - 31
Function Key Logic	10 - 37
Summary	10 - 39

Lesson 22 – Decision Tables

Objectives	10 - 41
Decision Tables	10 - 41
Decision Table Header	10 - 41
Decision Table Questions	10 - 43
Decision Table Actions	10 - 45
Decision Table Relationships	10 - 47
Messages	10 - 52
Add Functions	10 - 54
Summary	10 - 57

Lesson 23 – Scrolling Data on the Screen

Objectives	10 - 58
The Scrolling System	10 - 58
File Record Layouts	10 - 58
The Scrolling Functions	10 - 59
Summary	10 - 64

Lesson 24 – The Modify and Delete Functions

Objectives	10 - 65
The Modify Functions	10 - 65
The Delete Functions	10 - 66
Deleting Options	10 - 73
Summary	10 - 74

Lesson 25 – Further Reporting Techniques

Objectives	10 - 75
The Product Report	10 - 75
Report Header	10 - 76
Report Sorting	10 - 78
Report Selection Criteria	10 - 80
Line Headers	10 - 82
Report File Linkages	10 - 83
Painting the Report	10 - 87
Summary	10 - 90

Lesson 26 – Calculated Items

Objectives	10 - 92
Calculated Items	10 - 92
Generating the Report	10 - 95
Summary	10 - 95

Lesson 27 – User Help Screens

Objectives	10 - 96
Help Screens	10 - 96
Message Level Help	10 - 97
Screen Level Help	10 - 98
Summary	10 - 98

Lesson 28 – The Module Builder

Objectives	10 - 99
Introduction to Module Builder	10 - 99
Necessary Preparations	10 - 100
Field Specification	10 - 100
Record Layout	10 - 100
File Specification	10 - 101
Module Builder	10 - 101

Entering the Field Values	10 - 102
Main Access: File	10 - 102
Record	10 - 103
Index	10 - 103
Include All Fields	10 - 103
Module Details	10 - 103
Validate	10 - 106
Link To: File	10 - 106
Final Steps	10 - 107
Module Generation	10 - 107
The Module Building Process	10 - 108
Joining the Module to the Main Menu	10 - 108
Further Options	10 - 108
Summary	10 - 109
Necessary Preparations	10 - 109
Module Building	10 - 109
Joining the Module to the Application	10 - 110
Where You Are Now	10 - 110
From Here On	10 - 110

10 Further HP ALLBASE/4GL Facilities

This chapter contains the final lessons for the KSAM based application in this training guide. As well as using methods you have already seen, this chapter introduces a number of new techniques and new items.

The style of these lessons differs from the earlier lessons. Each keystroke is no longer fully defined for you. In most cases you're only given a description or diagram of the completed item. It's up to you to use the appropriate facilities to create the item.

This chapter draws your attention to any new techniques that are introduced. In addition, it discusses the purpose of the items you create. We recommend that you take your time going through these lessons. Look back over what you have already done, and if necessary, use the *HP ALLBASE/4GL Developer Reference Manual* for more detailed information on the items being used.

Throughout these lessons, we suggest that you create your own entries for the description fields on screens that have documentation fields.

The Applications

This chapter describes two substantial extensions to the existing *training* application. The first is a product/option file maintenance system. This system involves two files, a *product* file and an *option* file. The second is a supplier file maintenance system, which involves a single file, the *supplier* file.

Product/Option Application

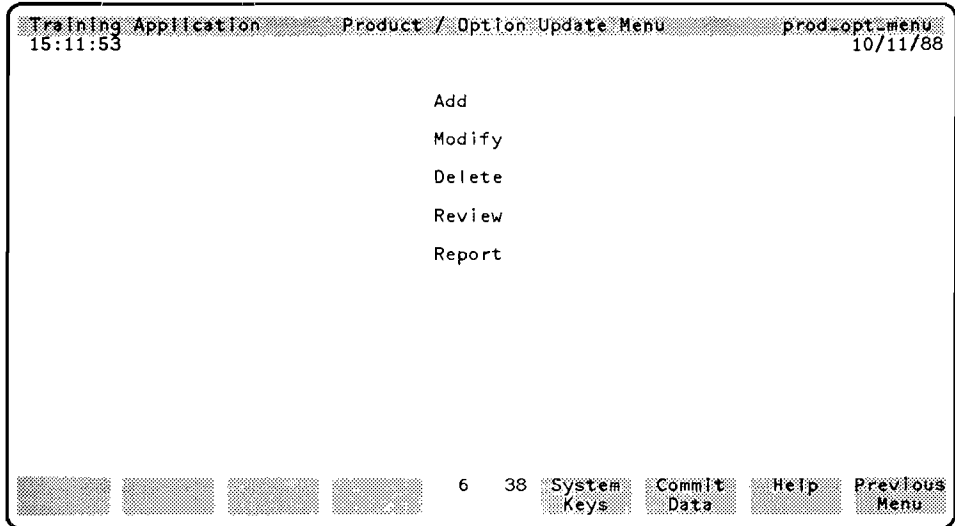
The first application involves a product/option relationship where each product must have at least one option, and can have any number of options related to it.

To add the new features to this application, you must add a new action item to the current *main* menu. You must also create some new screens using windows, scroll areas, and function keys. These additions require a number of new logic blocks as well as a decision table to support them. Finally you will create a new report that uses sort levels and file linkages. You perform these steps in lessons 19 to 27 of this chapter.

The logic used in this enhancement differs from the logic you have used so far. From the main menu, the user selects the *Product/Option Update* menu.

This new menu will offer five choices *Add*, *Modify*, *Delete*, *Review*, and *Report*. This allows the system administrator to assign menu item security to each menu item, limiting access to selected items for particular users. The menu item security system is described in the *HP ALLBASE/4GL Developer Administration Manual*.

The *Product/Option* menu is shown here.



This menu allows the user to select the following options.

Operation	Description
Add	Add a product or option.
Modify	Modify an existing product or option.
Delete	Delete a product or option.
Review	Review a product or option.
Report	Print a report on products and options.

The Application Screens

When the user selects an option from the *Product/Option* menu, the appropriate update mode is set. The same data screen is then displayed for each mode.

When necessary, this screen uses a window to allow the user to add, modify, or delete option records.

The screen and window appear as shown here.

Training Application		Add Product / Option		product	
14:08:53				10/11/88	
Product Number		Description			
		Supplier			
		Lead Time			
Option Number		Description			
		Cost			
		Qty on Hand			
Option #	Description	Cost	On Hand		
Product		Scroll Options	10	43	System Keys
					Commit Data
					Help
					Previous Menu

Function Keys

The data screen and windows used in this part of the application have sets of function keys that allow the user to swap between product maintenance and option maintenance.

Scrolling Options

At any time when a valid product record has been retrieved, the user can press a **Scroll Options** function key to display the details of the options for the current product. The option details are displayed in a scroll area at the bottom of the screen.

Reporting

The extensions to the application include a report that lists the details of products and options.

The user can specify a range of products for reporting. For all products in the specified range, the report lists the details of the products and the options. It also

calculates the cost of stock for each option, the cost of stock for each product, and the total cost of stock for all products listed.

Processing Rules

The processing rules for the product/option maintenance system are as follows:

- A product may be added or modified any time. When a product is added to the system, a basic option must also be added. This option must have the number "000".
- A product may only be deleted if there are no options except option number 000 associated with it.
- When a product is deleted, option 000 must also be deleted.
- An option, other than option 000, can be added to an existing product.
- An option may be modified at any time.
- An option other than option 000 may be deleted at any time.

Supplier Application

In lesson 28, the final lesson of this chapter, you will use the module builder feature of HP ALLBASE/4GL to create the *supplier* application. This application allows you to create and modify records in the *supplier* file, and is similar to the customer database application you created earlier. The *supplier* file contains one record per supplier, containing the supplier number, name and address of the supplier, along with the product number of the product supplied.

In the *supplier* application, or module, the product number is validated against the *product* file using a link between the screen field and the product file. Validation ensures that the product being supplied first exists in the *product* file. This link is also used to display information from the *product* file on the screen accessing the *supplier* file. As mentioned earlier, this application is not a definitive example of system design, instead it demonstrates the features of HP ALLBASE/4GL.

Module Building

When you use the module builder feature, HP ALLBASE/4GL automatically creates the new screen, and its associated function keys. HP ALLBASE/4GL also creates logic blocks and any other required application items. Together all these items comprise the *supplier* module. So the user can access the *supplier* module, you must add an action item to the *main* menu.

The Application Screen

When the user selects *Supplier* from the *main* menu the *mb_supplier* screen is displayed. With this data screen the user views, adds, modifies, and deletes supplier records.

Module builder creates the *mb_supplier* screen as shown below.

```

training                               Supplier Maintenance Screen                               mb_supplier

Supplier Number      0
Supplier Name
Address
State Code
Zip Code
Product Number
Product Description

Mode (Add)  Previous Record  Next Record  More Keys  7  40  System Keys  Commit Data  Help  Previous Menu
  
```

Function Keys

Module builder automatically adds the function keys sets required to perform view, add, modify, and delete operations. Included in these function key sets are function keys to scan all the records in the *supplier* file.

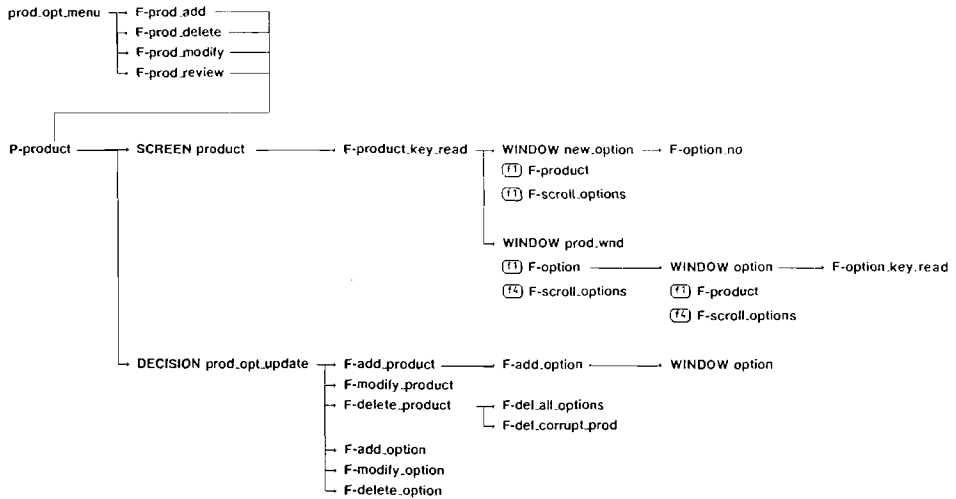
Application Structure

Once again, both applications use the concept of a process calling a screen. Functions associated with fields on the screen perform the necessary file look-up operations. When the user terminates the screen, control returns to the original process which then performs the appropriate file update operations.

Product/Option Application

In the product/option application the functions associated with screen fields, or screen function keys, can also display one of three different windows on the screen. Other functions called by function keys display data in a scroll area on the screen.

The structure of the application can be shown diagrammatically as follows.



This diagram shows the structure for the *Add*, *Modify*, *Delete*, and *Review* options on the Product/Option menu. It does not show the product report logic or the details of the option scrolling system.

Each menu option calls a function. These functions set the appropriate mode by initializing a variable. Each function then calls the *product* process.

The *product* process displays the *product* screen, which in turn has a function associated with the first field on the screen. The functions associated with this screen can display three different windows on the screen.

On exit from the *product* screen, control returns to the *product* process. The process calls a decision table to determine which function is executed to update the product and option files data.

You will probably want to refer back to this diagram as you define the various logic blocks for this application.

Supplier Application

The supplier application has one screen with two sets of function keys. This screen has three modes of operation: add, modify, or delete. HP ALLBASE/4GL automatically sets the mode, basing its choice on the previous user action. However, the user can alter the mode with a function key.

The structure of the *supplier* module is defined by templates, which are stored in library applications within the developer environment. Each item in the module is copied from a template. Module builder specifies the required templates when it builds the module.

Lesson 19 – Further Dictionary Items

Objectives

In this lesson, you define a number of dictionary items required for this part of the application. You will be using procedures that have been described earlier in this guide, so you should be familiar with the appropriate developer facilities.

This lesson also introduces a new dictionary validation item—a validation range.

Dictionary Requirements

The first task is to create the files used by this system. This involves creating the field specifications, record layouts, and data files. The files and fields you need are shown below:

- A product file with the following fields:
 - product_no
 - description
 - supplier_no
 - lead_time

- An option file with the following fields:
 - option_key
 - product_no
 - option_no
 - description
 - cost
 - qty_on_hand

Field Specifications

The first items you must create are the dictionary field specifications. The required specifications are listed below. Only the field entries that differ from the defaults are listed. Create these items now.

Product Number. The unique product number key for the product file. It is in the range *XY1000 - XY9999*.

Field Spec Name	<code>product_no</code>
Field Length	<code>6</code>
Minimum Entry Length	<code>6</code>
Edit Code	<code>U</code>
Range	<code>product_no</code>

Description. A 30 character item description field used by both files.

Field Spec Name	<code>description</code>
Field Length	<code>30</code>
Edit Code	<code>X</code>

Supplier Number. A six digit supplier identifier which can be used as a secondary key into the product file.

Field Spec Name	<code>supplier_no</code>
Field Length	<code>6</code>
Edit Code	<code>N</code>

Lead Time. A two digit value indicating the number of days required to obtain the product from the supplier.

Field Spec Name	<code>lead_time</code>
Field Length	<code>2</code>
Edit Code	<code>N</code>

Option Key. A concatenation of the product number and option number forming a unique key to the option data file.

Field Spec Name	<code>option_key</code>
Field Length	<code>9</code>
Minimum Entry Length	<code>9</code>
Edit Code	<code>U</code>

Option Number. A three digit option specifier. A unique value within any given product.

Field Spec Name	option_no
Field Length	3
Edit Code	N
Pad Character	0

Cost. A dollar value field indicating the cost per unit from the supplier of the particular option.

Field Spec Name	cost
Field Length	12
Edit Code	N
Decimal Places	2

Quantity on Hand. The current stock level of the particular option.

Field Spec Name	qty_on_hand
Field Length	5
Edit Code	N

When you have created these dictionary field specifications you can create the record layouts for these files.

Record Layouts

Create the following record layouts. Remember to generate the layouts once you have defined them.

Product record. This record contains the four fields associated with each product item.

Record Layout Name. product

Field	Field Spec. Name	Key #	Duplicates
1	product_no	1	N
2	description		
3	supplier_no	2	Y
4	lead-time		

Option record. This record contains the six fields needed to store and retrieve each option item.

Record Layout Name. option

	Field	Field Spec.	Name	Key #	Duplicates
1	option_key			1	N
2	product_no			2	Y
3	option_no			3	Y
4	description				
5	cost				
6	qty_on_hand				

Now you have created and generated the record layouts you can create the data files.

Data Files

You can now create the *product* and *option* data files. Create the file definitions, and then use the file creation screen to create the physical files.

Product File. The product file contains all the product records.

File Name	product
File Type	I
External Name	product
Default Record	1
Record Layout	product

Option File. The option file contains all the option records.

File Name	option
File Type	I
External Name	option
Default Record	1
Record Layout	option

Validation Ranges

The last dictionary item needed to support the field specifications and files is the *product_no* validation range. You haven't used validation range yet so it's described in full.

Menu Path

Dictionary, Validation Items, Ranges

Description

A validation range is used in similar manner to a validation table. A range defines lower and upper limits for the acceptable values in a field. When a range is assigned to a field specification used on a screen, HP ALLBASE/4GL automatically tests user input data against the limits specified by the range. For any given range, the acceptable values include the upper and lower limits, and all values between the limits.

You can also use validation ranges to select records for reporting.

Developer		Validation Ranges		ranges	
Range Name	product_no	Secured	N	(Y/N)	
Edit Code	U	(X/A/U/K/N/S/Q/D/T)			
Value - From	XY1000				
- To	XY9999				
Number of Decimal Places					
Message Name	product_no_error				

Field Specs. Tables Messages 14 54 System Keys Commit Data Help Previous Menu

Entering the Field Values

Range Name. Enter `product_no`

This is the name of the range within the Application.

Secured. Enter `N`

This indicates whether this item is secured against modification by an unauthorized developer.

Edit Code. Enter `U`

This is the standard edit code field indicating the type of range checking to be performed. This code is usually the same as the field to which the range is assigned.

If the edit code is *N* or *S* then a numeric check is performed. Otherwise the system collating sequence is used. Refer to the *HP ALLBASE/4GL Developer Reference Manual* for more details.

Value - From. Enter *XY1000*

This is the lower limit of the range.

Value - To. Enter *XY9999*

This is the upper limit of the range.

Number of Decimal Places. Leave blank.

For numeric range checking this entry specifies the precision to which the values should be compared.

Message Name. Enter *product_no_error*

Like a validation table, HP ALLBASE/4GL displays this message if the validation fails.

Press the **Commit Data** function key to create this validation range.

This completes the validation range definition. The next task is to create the variables and constants used by this application.

Variables

This part of the application requires five additional variables. They are described below.

confirm. This variable is used to receive confirmation from the user for a deletion request. The *Q* edit code is a special *Query* code which allows the entry of *Y*, *y*, *N*, or *n* only.

Name	<i>confirm</i>
Length	<i>1</i>
Edit Code	<i>Q</i>

current_record. Indicates the current record type. Either *product* or *option*.

Name	<i>current_record</i>
Length	<i>3</i>
Edit Code	<i>X</i>

mode. Indicates the current update mode. One of Add, Modify, Delete, or Review.

Name	mode
Length	6
Edit Code	X

option_status. Indicates whether the current option record exists.

Name	option_status
Length	1
Edit Code	X

product_status. Indicates whether the current product record exists.

Name	product_status
Length	1
Edit Code	X

Alphanumeric Constants

Six alphanumeric constants are required for the application. They are described below.

add. An indicator of the current update mode.

Name	add
Value	Add

delete. An indicator of the current update mode.

Name	delete
Value	Delete

modify. An indicator of the current update mode.

Name	modify
Value	Modify

option. An indicator of the current record.

Name	option
Value	opt

product. An indicator of the current record.

Name	product
Value	prd

review. An indicator of the current update mode.

Name	review
Value	Review

Numeric Constants

This application requires two additional numeric constants.

Name	zero
Value	000

Name	end_of_file
Value	19110

You have now completed defining all of the constants and variables required by the application. All of the additional dictionary items for this part of the application are complete.

Summary

In this lesson you defined a number of dictionary items for this application. In particular, you defined a validation range.

Validation ranges have the following properties:

- Validation ranges can be used to test whether user input data in a screen field is within a specified range of values. Validation ranges can also be used for selecting records for reporting.
- A validation range specification requires an edit code. These are the same as the edit codes that are applied to dictionary field specifications.
- You must specify the starting and finishing values for a validation range. The acceptable values for a validation range include the starting and finishing values.
- If the edit code for a validation range is N or S, HP ALLBASE/4GL performs the validation check according to numeric values. For other

edit code types, HP ALLBASE/4GL performs the validation checking on a character by character basis using the system collating sequence.

- The validation range definition allows you to specify the name of a message to be displayed to the user if input data fails the validation check. If you don't specify a message name, HP ALLBASE/4GL displays a system defined default message on validation failure.

Lesson 20 – More Screen Techniques

Objectives

In this lesson you will create a number of screens for the application. The screens you create will include a scroll area, and you will also create some window screens.

This lesson also introduces the screen painter facilities that allow you to use an existing screen as a template for a new screen, and to include line drawing characters on your screens.

Application Screens

This application requires five additional screens—one menu, a data screen, and three windows.

The general application flow involved with the new screens is as follows. The main menu passes control to the new product menu. From this menu, the user selects one of four possible file update actions. These actions are functions that set the appropriate update mode. All four functions then call the same process. The process displays the product data screen.

The product data screen contains only fields that are relevant to the product records. A function on the first field of this screen displays a window with headings for a scroll area as well as some line drawing characters which draw a box on the product screen.

When the user presses the `Option` function key an *option* window is displayed. This window contains the same scroll area headings as the data screen. It also contains the fields relevant to the option record as well as its own arrangement of line drawing characters to complete the box on the product screen as well as providing a box around its own fields.

The exact line and column positioning of the items you create is not critical. Although you do have some freedom as far as the position of the items is concerned, you must keep the fields in the same order as shown on the example screens.

Main Menu

First, you need to modify the main menu. This menu needs an additional menu item to call up the product menu.

Use the screen painter to call up the *main* menu. Add the following action item below the items already defined.

Text	Type	Name
Products	D screen	prod-opt-menu

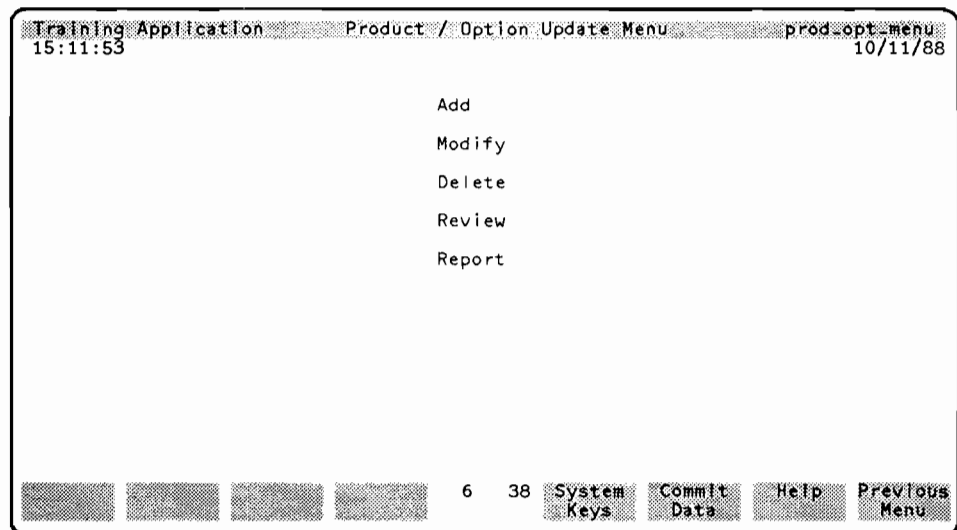
When you have created the new action item, exit from the painter. This automatically saves the new menu. You can now create the other new screens.

Product Menu

Create a screen header for the product menu called *prod_opt_menu*. Enter **M** in the *Screen Type* field to specify that this screen is a menu. Accept the default values for the other fields, and enter a suitable description.

Commit this screen header and call up the screen painter.

Painting the Screen



Paint the image shown here. The following notes describe the items on the screen.

The top two lines are similar to screens you have already defined. First, paint the application title *application_name*, and then paint the screen title as a text item. In the top right corner is a communications area field *SCREEN. The system date and time are displayed on the second line.

Then create the following menu action items.

Text	Type	Name
Add	F (function)	prod_add
Modify	F (function)	prod_modify
Delete	F (function)	prod_delete
Review	F (function)	prod_review
Report	R (Report)	product

When you have defined these items exit from the painter. This saves the menu as you exit. Next, create the *product* data screen.

Product Data Screen

The product data screen contains all of the fields necessary to maintain product records. It has an area for the option window, and it also has an area for scrolling details for the options for an existing product.

First define the screen header. Enter these values, and accept the defaults for the other fields.

Name	product
Screen Type	D
Scroll: Top Line	17
Scroll: Bottom Line	22
Scroll: Direction	D
Window Starting Line	9

This header specifies a scroll area from line 17 to 22. Whenever a new line of data is displayed in the area with a SCROLL command, the lines already displayed move down as the new line is displayed at the top of the area.

The header also specifies the starting line for a window. Whenever a window is displayed on this screen, the current contents of the screen from line 9 downwards are cleared and the window contents are displayed from line 9 to the bottom of the screen.

When you have created the screen header, call up the screen painter and paint the *product* data screen.

Painting the Screen

Training Application	V-mode	Product / Option	Product
16:18:12			10/11/88
Product Number	Description		
	Supplier		
	Lead Time		
Move	Copy	End Function	Delete
		1	1
	Column Mode	Begin Block	End Block
			Main Keys

The two lines at the top of the screen are similar to those you painted for the first training application except for the entry in the center of the first line.

Note the entry *V-mode*, just before the literal *Product/Option*. This is a system item referring to the variable *mode*. Since the same screen is used for adding, modifying, deleting, and reviewing data, this screen item displays the current update mode next to the screen title text. The maximum length of the *mode* variable is 6 characters, so make sure you leave a space between it and the literal text *Product/Option*.

Use the dictionary field specifications to create the fields on the screen. The prompt text to the left of each field shows you which field specification to use.

The next task is to create the frame of line drawing characters that surround the items on this screen. Position the cursor at line 3 column 2 and use the following steps to create the box on the screen.

Action**Notes**

Press

Special Text

This displays the special text function keys. An asterisk appears in the function key label corresponding to the enhancement modes that are currently on. The default is line drawing mode only.

Press **Return**

The painter enters text input mode.

Press **Q**

The **r** character is echoed to the screen. (Refer to your terminal reference manual for a full mapping of line drawing characters to the keyboard characters.)

Press **:**

76 times

This extends the line drawing characters to the right giving a line above the prompts and data fields.

Press **W**

The **w** character is echoed to the screen.

Press **Return**

This finishes the special text item creation.

Here are the line drawing characters you will use in this lesson and their equivalent keyboard characters.

r	=	Q	w	=	W
L	=	A	J	=	S
┆	=	1	┆	=	2
-	=	:	 	=	:

Create the vertical lines on each side of the box. Each line is made from single character items. You can use the copy facility to speed up the process of creating the lines.

When you have finished painting the screen, exit from the screen painter to save the screen.

Screen Field Details

The next step is to define the attributes for each of the screen fields on the *product* screen. Only those values requiring an entry are shown. Accept the defaults for all other values.

Field 1

Primary Data Movement	F-product_no.product
Function	product_key_read
Prior/After/Both	B

Field 2

Primary Data Movement	F-description.product
-----------------------	-----------------------

Field 3

Primary Data Movement	F-supplier_no.product
-----------------------	-----------------------

Field 4

Primary Data Movement	F-lead_time.product
-----------------------	---------------------

This screen is now complete. Generate it, and return to the screen header screen to create the option window.

Option Window

The option window screen contains all the fields necessary to maintain records for the options associated with particular products. It is overlaid on the product data screen when the user presses the **Option** function key.

First, define the screen header. These are the values you must enter. Accept the defaults for the other values.

Screen	option
Screen Type	W
Function Keys	option

When HP ALLBASE/4GL displays a window on a data screen, the function keys defined for the window are also displayed. If you have not defined a set of function keys for the window, the function keys from the original screen are not changed.

When you have created the screen header, use the screen painter to create the option window.

Painting the Screen

Option Number	Description	Cost	Qty on Hand
Option #	Description	Cost	On Hand

Input Field	Output Field	System Item	Special Text	1	1	Layout Keys	Sequence Numbers	More keys	Exit
-------------	--------------	-------------	--------------	---	---	-------------	------------------	-----------	------

Notice that the *option* window is painted at the very top of the painter screen. When it is displayed, the window is overlaid starting from the window start line of the underlying data screen. This means that the line at the top of this screen is aligned with the bottom of the open box on the underlying data screen. To simplify the task of aligning the fields on the window, there is a screen painter facility that you haven't used yet. This is the *Overlay Screen* capability. With just the blank screen painter screen displayed, follow these steps.

Action

Press

More Keys

Press

Overlay Screen

Notes

This displays a further set of function keys.

A prompt for the overlay screen name appears.

This is the overlay prompt that is displayed on the screen painter screen.

Please enter the name of the Screen to overlay...									
product				24	8				Abort

Action	Notes
Enter product	HP ALLBASE/4GL prompts you to move the cursor to the starting position of the overlay.
Press ⏏ , then Return	This moves the cursor to the top left hand corner of the screen. This where you want the overlaid screen to be inserted.
Press Return	The <i>product</i> data screen is overlaid onto this screen.

You can now paint the window with the new items in correct alignment with the existing items on the product screen. When you've finished, you can delete the unwanted items from the top of the overlaid screen and then move the window up to the top of the screen where it should be.

First copy the horizontal special text item at line 3 to line 9. Then, with the cursor located at the ends of the item at line 3 where **r** and **⏏** currently appear, press **1** and then **2** respectively. The correct characters **⏏** and **⏏** are displayed. To correct the line drawing characters at the ends of the item at line 9, use the keys **A** and **S**.

Now complete the screen with all of the prompts and input items. The last line of text on the window is a heading for the scroll area.

Then delete the unwanted fields from the top of the *product* screen and use the block move facility to move all of the window fields up to the correct position.

When you have completed the screen, leave the screen painter and define the screen field details for this screen.

Screen Field Details

Define the attributes for each of the screen fields. Only those values requiring a specific entry are shown here. Accept the defaults for the other fields.

Field 1

Primary Data Movement	F-option_no.option
Function	option_key_read
Prior/After/Both	A

Field 2

Primary Data Movement	F-description.option
-----------------------	----------------------

Field 3

Primary Data Movement	F-cost.option
-----------------------	---------------

Field 4

Primary Data Movement	F-qty_on_hand.option
-----------------------	----------------------

When you have completed the *option* window, remember to generate it. You can then go on to create the *new_option* window, and then return to the screen header screen to create the *product* window screen.

The New Option Window

This application requires a further window to allow the user to add option number 000 for a new product. This window is almost identical to the *option* window, except for the *option_no* field. On the *new_option* window, the *option_no* field is a display only field.

Use the utilities menu copying screen to create a copy of the *option* window called *new_option*.

Use the screen field details screen to modify the details for field number 1 on the *new_option* window to read as shown below.

Field 1

Default Data Movement	N-zero
Input/Display	D
Function Name	option.no
Prior/After/Both	P

Generate the screen when you have made these changes. You can now return to the screen header to create the *product* window.

The Product Window

The last screen you need to create is the product window. This window is displayed when the *option* window is not required. In effect it clears the option window from the screen. Define the screen header with the following details.

Name	prod_wnd
Screen Type	W
Function Keys	product

Painting the Screen

Option #	Description	Cost	On Hand
Input Field	Output Field	System Item	Special Text
1	1	Layout Keys	Sequence Numbers
		More keys	Exit

This window has no data fields on it. The easiest way to create it is to overlay the *option* window that you have just created, and then delete all the fields relating to any unwanted option details. Then you can modify the existing line drawing characters to end up with the desired screen image.

When you have completed this screen, exit from the screen painter and generate the *product* window. Now that you have created all of the screens required by the application, you can define the function key sets used by the screens.

Summary

In this lesson you created a number of new screens for the application. These screens include a data screen, a menu, and three windows. The data screen uses a scroll area, and also has a defined area for displaying the windows.

You also used some new screen painter techniques. These included:

- The overlay screen feature that allows you to use an existing screen as a template for a new screen.
- The special text feature that allows you to include line drawing characters on screens. The special text feature also allows you to specify display enhancements such as color and inverse video for particular screen items.

Lesson 21 – Extending the Application Logic

Objectives

In this lesson you will create a number of logic blocks for the application. These logic blocks include the *product* process, the functions associated with fields on the screen and windows, and some functions associated with function keys.

You will also create two sets of function keys for the windows used in this application.

Function Keys

This application requires two sets of terminal function keys. They are the *product* function keys, and the *option* function keys.

Both sets of keys display a function key that allows the user to scroll the options for the current product.

The *product* key set also has a key that allows the user to select option updating. The *option* screen function key set has a key that allows the user to return to product updating.

Only the function keys required for this application are listed here. Accept the default entries for the other keys.

Function Key Set Name: product

Key #	Action	Label Line 1	Label Line 2
1	F-option	Option	
4	F-scroll_options	Scroll	Options

Function Key Set name: option

Key #	Action	Label Line 1	Label Line 2
1	F-product	Product	
4	F-scroll_options	Scroll	Options

When you have defined the two function key sets, you can define the logic blocks that drive the application.

The New Logic Blocks

This system requires a number of new logic blocks. First there are four functions which set the appropriate update mode. These are the four functions called from the menu.

These functions all call the same process, which then controls the display of the product screen and updating of the product and option files. The process calls a number of functions which perform the appropriate update depending on the update mode and current record type.

The screens themselves also need some functions. One is associated with the product number field on the product screen. The second is associated with the option number field on the option window.

The function key sets you defined earlier require some functions as well. These are the functions that allow the user to swap between product and option maintenance tasks.

The final functions are the scrolling functions which are also called from the function keys. These enable the user to scroll through the existing options for the current product.

The first task is to create the four initial functions.

The Mode Functions

Four of the actions called from the new menu are functions. These functions move the appropriate update mode to the *mode* variable and then call the *product* process.

Create and generate the functions shown below:

Function – *prod_add*

- 1 MOVE C-add V-mode
- 2 PROCEED product

Function – *prod_delete*

- 1 MOVE C-delete V-mode
- 2 PROCEED product

Function – *prod_modify*

- 1 MOVE C-modify V-mbde
- 2 PROCEED product

Function – *prod_review*

- 1 MOVE C-review V-mode
- 2 PROCEED product

The PROCEED statements in these function invoke the *product* process. Since any current function terminates when a process is started, these functions don't need an EXIT command.

The Product Process

You can now create the *product* process as shown:

```

1 MODE *WRITE product option
2 MOVE C-product V-current_record
3 MOVE C-no_record V-product_status
4 TRANSACT *BEGIN
5 SCREEN product
6 DECISION prod_opt_update
7 IF *IOSTATUS = "00000" | *IOSTATUS = N-end-of-file
   THEN ZIP ELSE ENTER 10
8 TRANSACT *END
9 ENTER 4
10 MESSAGE no_transact
11 TRANSACT *UNDO
12 SCREEN prod_opt_menu

```

The process is described below.

1. This step enables write access to both the product *and* option files.
2. This step ensures that the current record type is a product record when the screen is first displayed. Options can only be accessed *after* an existing product has been retrieved from the product file.
3. This step initializes the value of the variable *V-product_status* when the screen is first displayed.
4. This step introduces the TRANSACT command. The TRANSACT command *defines* groups of file operations that make up one logical transaction.

In this application, adding a new product, or deleting an existing product are transactions that involve changes to two data files. To ensure logical consistency of the data files, both files must be updated successfully during the transaction. If a system failure occurs in the time between the two files being updated, the data files could be left in a mutually inconsistent state. In this type of situation, you can use the TRANSACT command to mark the beginning and end of the logical transaction. The TRANSACT *BEGIN command marks the beginning of the transaction, and the TRANSACT *END command marks the end. The TRANSACT *UNDO command reverses all file transactions that have occurred since the last TRANSACT *BEGIN command.

5. This step displays the *product* data screen. The same screen is used for each mode. The setting of the *mode* variable determines how the process updates the files when the user commits the screen.
6. The DECISION command executes a decision table. The decision table tests the update mode and record type, and then invokes the appropriate file update function.
7. If the KSAM file manager returns either 00000 or 19110 (the value of the constant *end_of_file*) this step terminates the transaction. Values of 00000 or 19110 in *IOSTATUS indicate that all file transactions occurred without error, or that an end-of-file condition occurred. An end-of-file condition may occur when options for a product are deleted. The ZIP command passes control to step 8 to terminate the current transaction if these conditions occur.

If the file manager return status is not 00000 or 19110, a file error has occurred during the update functions called from the decision table. In this case, the data files may be mutually inconsistent, and control passes to the error handling logic starting at step 10.

8. This step marks the end of the TRANSACT command that started in step 4.
9. This step returns control to step 4 to display the *product* screen again.
10. Control only passes to this step if an error is detected at step 7. This step displays a message to warn the user that an error has occurred and the current product information may be corrupted.
11. This step reverses the file operations that have occurred since the TRANSACT *BEGIN command.

12. This step returns the user to the *product_opt_menu* screen. Displaying a menu screen is one technique you can use to recover from serious, but non-fatal error conditions. When HP ALLBASE/4GL displays a menu, all current activity in the application ceases. This means that the application returns to a known condition. To continue from the *prod_opt_menu* the user must select a menu action. Any actions that can update data files initiate a process, so all file record and screen buffers are cleared automatically.

The *product* process requires the following message:

Message – *no_transact*

Name	no_transact
Type	WARN
Contents	"A file error has occurred." " This transaction will be reversed."

Screen Field Logic

You can now define the functions attached to the *Product Number* and *Option Number* fields on the screen and window.

Function – *product_key_read*

This function is called as both a prior and after function on the *product_no* field on the *product* screen.

The function is effectively divided into two parts. One part is executed before the user enters data into the field. This part of the function tests the current record type, and performs some initialization tasks.

The other part of the function is the “after” function. This part of the function reads the *product* file to retrieve a record matching the product number entered by the user.

Create and generate this function now. The details of the function are explained after the listing.

```

1 IF *ENTERED *ON THEN ENTER 8
2 IF V-current_record = C-product
   THEN WINDOW prod_wnd ; EXIT
3 SHOW *REFRESH S-product_no S-lead_time
4 MATH F-option_no.option + 1 = F-option_no.option
5 SHOW *REFRESH S-option_no

```

```

6 MOVE "5" *FIELDNO
7 EXIT
8 FILE *READ product *KEY= * ; ENTER 13
9 SHOW *REFRESH S-product.no S-lead.time
10 MOVE C-record V-product.status
11 ON 1
12 EXIT
13 IF *IOSTATUS <> N-record.not_found
    THEN MESSAGE file_error ; SERIES 14 16 ; EXIT
14 FILE *BUFFER product
15 CLEAR *MAP S-description S-lead.time
16 MOVE C-no_record V-product.status
17 IF V-mode <> C-add THEN MESSAGE prod_not_exist ELSE
    MESSAGE add-prod; VISIT new-option; TIE 2
18 EXIT

```

Steps 1 through 7 of the function form the *before* part of the function. Steps 8 through 18 form the *after* part of the function.

The following notes describe the operation of the function.

1. This step tests the current state of the switch *ENTERED. This indicates whether the function is being executed before or after data entry to the field. If the function is being performed after data entry, control passes to step 8. Steps 2 to 7 are only executed when the function is called before data entry in the field.
2. This step tests the current record type. If it is a *product* record, the current window area is cleared by overlaying the *prod_wnd* window on the current screen. Then the function exits allowing normal input processing of the field to continue.

This is always done the very first time the screen is displayed. The *prod_wnd* window completes the line drawing character framing around the product fields and displays the heading lines for the scroll area. To the user, the screen and the window appear to be a single screen.

3. This step, and the next four, are only executed if the current record type is an *option*. In this step the product screen fields are redisplayed from the current product record buffer. Since the user can only access an option when a current record exists, the current product record must contain valid data. If the current record type is an option record, the option window must already be displayed on the current screen, since the user has accessed an option.

4. This step increments the current option number by 1.
5. This step refreshes the option number field on the option window.
6. This step moves the number of the *Option Number* field to the communications area field *FIELDNO. When this function exits, the screen processor immediately passes control to that field. Refer to the screen processing rules in Chapter 7 if you want to review this.

The *Option Number* field is field number 1 of the window on which it was painted. However, when HP ALLBASE/4GL displays a window, the window field numbering is adjusted to start after the number of the last field on the original data screen. In this case field number 1 on the window becomes field number 5 on the base screen.

7. The function exits at this point, completing the *before* function processing. Processing of the current field finishes, and the cursor moves to the *Option Number* field on the window.
8. Control passes to this step when the function is executed after data entry in the field. This step reads the *product* file for a record with a key equal to the value in the current screen field. If an error occurs during the file access, control passes to step 13.
9. A record has been found, so the product screen data fields are refreshed with the contents of the product file record.
10. A product record has been found so the *product_status* variable is set to reflect this. Other parts of the application use this variable to determine if the current product exists.
11. User switch 1 is set *on*. This switch is used by the scrolling system to determine if any scrolling has already been done for this product.
12. The function exits.
13. This test is performed if the file read operation in step 8 fails. If the error is not the standard *record_not_found*, then a number of actions occur. First the *file_error* message is displayed. Then the next command is executed.

The *SERIES* command performs steps 14 to 16 and then returns to the next step. This is the *EXIT* command that follows the *SERIES* command.

14. This command clears the product file record buffer.
15. This command clears the screen fields from the product *Description* field to the product *Lead Time* field.
16. A product record has not been found so the *product_status* variable is set to reflect this.

17. This step displays a message that depends on the current update mode. If the current mode is *add*, this step displays the message *add_prod* and then calls the function *new_option*. Finally, the TIE command specifies that field number 2 is the next field to be processed.

These commands are executed if the user is adding a new product. The *new_option* function displays the *new_option* window to ensure that the user specified the details for option number 000 for the new product.

If the user has not selected *add* mode, and a product record has not been found, this step displays the message *prod_not_exist*.

18. The function exits at this point.

Function – *new_option*

This function is executed by a VISIT command in the *product_key_read* function. It is called when the user is adding a new product. It displays the window *new_option* to allow the user to enter the details of option 000 for a new product. Since the *new_option* window contains required fields, the user cannot commit the current screen without completing the option details.

The *new_option* function contains the following commands.

- 1 **MOVE** C-no_record V-option_status
- 2 **FILE** *BUFFER option
- 3 **WINDOW** new_option
- 4 **EXIT**

The *new_option* function is described below.

1. This step sets the variable *V-option_status* to indicate that there is no current record.
2. The **FILE** *BUFFER command clears the *option* file record buffer.
3. This step displays the *new_option* window on the *product* screen.
4. The function exits.

Function – *option_key_read*

This function is called as an after function on the *option_no* field on the option window. The function performs some preliminary checking and initialization, and then reads the *option* file for a record matching the option number entered by the user.

```

1 IF V-current_record = C-product THEN EXIT
2 IF V-mode = C-delete & * = N-zero
   THEN MESSAGE del.prod.opt; EXIT
3 LINK 2 *S01 * F-option.key.option
4 FILE *READ option ; ENTER 8
5 SHOW *REFRESH S-option_no S-qty-on-hand
6 MOVE C-record V-option_status
7 EXIT
8 IF *IOSTATUS <> N-record_not_found
   THEN MESSAGE file_error ; SERIES 9 11 ; EXIT
9 FILE *BUFFER option
10 CLEAR *S 6 8
11 MOVE C-no_record V-option_status
12 IF V-mode <> C-add THEN MESSAGE opt_not_exist
   ELSE MESSAGE add_option
13 EXIT

```



The function operates as described below.

1. This ensures that the function is not performed if the current record being handled is a *product* record. There is only one situation where this may occur. You will encounter this situation when you define the functions associated with the function keys.
2. This step tests the current mode, and the contents of the *option_no* field. If the mode is *delete* and the option number is 000, the user is attempting to delete option number zero. This is not permitted, so this step displays an error message.
3. This step introduces a number of new items. The LINK command concatenates a number of fields into another field. In this example, the LINK command concatenates the fields **S01* and *** into the *option_key* field on the *option* file buffer. This step builds a unique primary key to access the option file.

In this command, *** is a reference to the contents of the current screen field. Similarly, **S01* is a reference to the contents of the first field on the current screen. On this particular screen it is the *product_no* field.

The window for the LINK command does not prompt you for the number of items to be linked together. HP ALLBASE/4GL calculates the number of link fields you enter, and then automatically inserts the correct link count value. The LINK command also has an optional joiner character which is

inserted between each field as it is linked. This example does not need a joiner character so you can leave the field blank.

4. This step reads the option file. Since the previous step moved the key value to the appropriate field in the file buffer, there is no need to specify the key for this file read operation. In all other respects this command is identical to the file read statements that you have already used.
5. If the specified record is found, the `SHOW` command displays the record buffer contents in the appropriate fields on the screen.
6. An option record has been found, so the `option_status` variable is set to reflect this.
7. The function exits.
8. This step starts the file read error section of the function. If the file error is caused by any condition other than a record not found condition, it displays an error message, clears the screen and file buffers, sets the status variable, and then exits.
9. This step clears the option file buffer.
10. This step clears the option window fields.
11. The `option_status` variable is set to show that a record was not successfully read.
12. The mode is checked and an appropriate message is displayed.
13. The function exits.

The `option_key_read` function requires the following message.

Message – `del_prod_opt`

Name	<code>del_prod_opt</code>
Type	<code>ERROR</code>
Contents	<code>"To delete option 000 " "you must delete the product."</code>

You have now completed the screen field functions. The next step is to create the functions that allow the user to swap between the option window and the product screen using the function keys.

Function Key Logic

Two functions called from the function keys allow the user to switch between updating products and updating options.

Function – *option*

This function allows the user to move to the *option* window when a valid product record has been retrieved. The *option* function contains the following steps:

```

1 IF V-product_status <> C-record THEN MESSAGE
   no_product; EXIT
2 IF V-mode = C-review THEN MESSAGE no_option_review; EXIT
3 MOVE C-option V-current_record
4 MOVE C-no_record V-option_status
5 FILE *BUFFER option
6 WINDOW option
7 MOVE "5" *FIELDNO
8 EXIT

```

The function operates as described below.

1. Since options can only be accessed after a product record has been retrieved, this step checks that a product record is current. If there is no current product record, it displays an error message and then exits.
2. The options can be reviewed by using the *Scroll Options* function key. If the current mode is *review*, this step displays a warning message, and the function exits.
3. This step starts setting up for the option window. It specifies that the current record type is now an *option* record.
4. An option record has not been accessed yet, so the *option_status* variable is set to *no_record*.
5. This step clears the option file buffer. Data may otherwise remain in the option file buffer from a previous update.
6. This step displays the *option* window. The current contents of the data screen from line 9 downwards is cleared and the window is overlaid on the screen from that line downwards.
7. This step changes the value of the current field number. The cursor moves to the option number field, number 5, on the screen when the function finishes.

Note

Since the function changes the value of *FIELDNO, HP ALLBASE/4GL automatically commits the current field when this function finishes. If the user presses the **Option** function key while the cursor is on the *product_no* field, the *product_key_read* function is executed as an after function.

8. The function exits.

Function – *product*

This function allows the user to return to the product screen. The *product* function contains the following steps:

- 1 MOVE C-product V-current_record
- 2 ON 1
- 3 MOVE "1" *FIELDNO
- 4 EXIT

This function is very simple. The current record mode is first set to *product*. The initial scroll indicator, user switch 1, is set to *on*. Then the number of the *product number* screen field is moved to *FIELDNO. The function then exits.

Note

As in the *option* function, this action causes a field commit for the current field. This is the reason for checking the current record type in line 1 of the *option_key_read* function.

When the cursor moves to the first field of the screen the *product_key_read* function on that field is performed in its prior state. If you refer back to that function, you will see that it clears the window area by displaying the *prod_wnd* window.

Function – *option_no*

This function is called as a prior function from the *option_no* field on a *new_option* window. This window is executed when the user enters the details for option

number 000 for a new product. The *option-no* function contains the following commands.

- 1 **FILE *BUFFER** option
- 2 **CLEAR *S 6 8**
- 3 **MOVE N-zero F-option-no.option**

The *option-no* function operates like this:

1. This step clears the *option* file buffer of any data from a previous update.
2. This step clears the screen fields from the *description* field to the *qty-on-hand* field.
3. This step moves the value 000 to the *option-no* field on the *option* file buffer.

Now all of the screen and window handling for this application is complete. You now need to provide the ability to update the application data files. The first item to define is a decision table to invoke the correct update function.

Summary

In this lesson you created a number of logic blocks for the application.

Some of these logic blocks introduced logic commands that you haven't used before. The commands are:

- The **TRANSACTION** command.

This command allows you to define groups of file transactions that must be performed together to ensure logical consistency of the files. The command exists in three forms. The ***BEGIN** and ***END** forms of the command mark the start and finish of the group of file commands. The ***UNDO** form of the command reverses all file transactions that have occurred since the last **TRANSACTION *BEGIN** command.

- The **LINK** command.

The **LINK** command concatenates a number of fields, and places the result in a specified field. The **LINK** command optionally allows you to specify a “joiner” that is inserted between the fields as they are linked. In this application, you used the **LINK** command to concatenate the product number and the option number to create a unique *option-key* to read the option file.

The *HP ALLBASE/4GL Developer Reference Manual* describes all the logic commands.

This lesson also introduced an important feature of the screen processing system. The *product* function and the *option* function are both called from function keys. These functions change the value in the communication area field *FIELDNO. When a function called from a function key changes the value in *FIELDNO, HP ALLBASE/4GL executes the function, and then immediately performs the input processing for the current screen field when the function exits.

In this application, pressing the **Option** function key while the cursor is on the *product.no* field executes the *option* function immediately. HP ALLBASE/4GL then completes the input processing for the *product.no* field. This processing executes the *product.key-read* function to retrieve a product record.

Lesson 22 – Decision Tables

Objectives

In this lesson you will create a decision table. The *product* process executes this decision table when the user terminates the *product* screen. The decision table determines which update function is executed to update the product and option files after data entry.

You will also create the functions that add new records to the product and option files.

Decision Tables

Decision tables allow you to define a series of actions to be performed as the result of the outcome of a series of conditional tests or questions. A decision table can test up to eight questions and execute up to eight actions such as functions, processes and screens.

Having defined the questions and actions, you can then define up to 31 relationships. Each relationship defines a combination of results for the questions, the actions to be performed, and the order in which the actions are performed.

You can use a decision table to perform the same functions as complex IF-THEN-ELSE structures.

A decision table is defined with four screens:

- The decision table header screen.
- The decision table questions screen.
- The decision table actions screen.
- The decision table relationships screen.

In this application, the decision table tests the status of a number of variables that are set while the *product* screen is active. The results of these tests determine the appropriate file update actions.

Decision Table Header

You must define a decision table header before you can access any other parts of the decision table.

Menu Path

Logic, Decision Tables, Header

Description

This screen allows you to enter brief details about the decision table.

Developer	Decision Table Header	dtable-header
Decision Table Name	prod_opt_update	Secured # (Y/N)
Description prod_opt_update AUTHOR: rmjones This table determines which function is called to update the product and option data files. Last Modification: Date Time		
Question	Actions	R'ships
Generate	21	57
D. Table	System	Commit
	Keys	Data
	Help	Previous
		Menu

Entering the Field Values

Decision Table Name. Enter `prod_opt_update`

This is the name of the decision table.

Secured. Accept the default value `N`

This indicates whether the decision table is secured against modification by an unauthorized developer.

Description. Enter a suitable description.

Press the `Commit Data` function key to create the header for the decision table.

Decision Table Questions

Menu Path

Logic, Decision Tables, Questions

Description

This screen allows you to enter up to eight separate IF statements. These are the questions to be resolved when the decision table is executed.

Decision Table		prod_opt_update	Secured	N (Y/N)
Questions				
IF	Data name 1:	V-mode	IF Test	#
	Data name 2:	C-add		
IF	Data name 1:	V-mode	IF Test	#
	Data name 2:	C-modify		
IF	Data name 1:	V-mode	IF Test	#
	Data name 2:	C-delete		
IF	Data name 1:	V-current_record	IF Test	#
	Data name 2:	C-product		
IF	Data name 1:	V-current_record	IF Test	#
	Data name 2:	C-option		
IF	Data name 1:		IF Test	
	Data name 2:			
IF	Data name 1:		IF Test	
	Data name 2:			
IF	Data name 1:		IF Test	
	Data name 2:			

Header Actions R'ships Generate D.Table 17 25 System Keys Commit Data Help Previous Menu

Entering the Field Values

Decision Table Name. Press

This field defaults to the name used in the last decision table screen. In this case, it defaults to *prod_opt_update* from the decision table header screen.

Questions

The remainder of this screen contains eight sets of three fields. The fields are *IF Data name 1*, *IF Test*, and *Data name 2*. Each set of three fields specifies a question to be resolved.

Data name 1. Enter *V-mode*

This is the name of the first data item that is to be tested. The rules for the construction of an IF statement in a decision table are the same as those for the construction of an IF statement in a logic block.

IF Test. Enter =

This is the relationship that must exist between the two named fields. (A decision table question can also test the status of a single item. For example, the test can be *BLANK.)

Data name 2. Enter *C-add*

This is the name of the data item that the first data item is tested against. In this case it is the alphanumeric constant *add*.

The first question is now complete. Enter the remaining four questions as shown below:

Question 2

Data Name 1	<i>V-mode</i>
IF Test	=
Data Name 2	<i>C-modify</i>

Question 3

Data Name 1	<i>V-mode</i>
IF Test	=
Data Name 2	<i>C-delete</i>

Question 4

Data Name 1	<i>V-current_record</i>
IF Test	=
Data Name 2	<i>C-product</i>

Question 5

Data Name 1	<i>V-current_record</i>
IF Test	=
Data Name 2	<i>C-option</i>

Press the **Commit Data** function key when you have completed these entries.

Decision Table Actions

Menu Path

Logic, Decision Tables, **Actions**

Description

This screen allows you to specify up to eight actions to be performed as a result of the resolution of the decision table questions.

Developer		Decision Table Actions		dtablactions					
Decision Table			<code>prod_opt_update</code>	Secured N (Y/N)					
Action Number	Action (Prefixes: B-/D-/F-/H-/P-/R-/X-)								
1st	F-add_product								
2nd	F-modify_product								
3rd	F-delete_product								
4th	F-add_option								
5th	F-modify_option								
6th	F-delete_option								
7th									
8th									
Header	Question	R'ships	Generate D.Table	17	45	System Keys	Commit Data	Help	Previous Menu

Entering the Field Values

Decision Table Name. Press

This field defaults to the name used in the last decision table screen. In this case, it defaults to `prod_opt_update` from the decision table questions screen.

Actions

The next eight fields contain the type and name of the action to be performed for the particular action number.

1st. Enter F-add_product

This indicates that action number 1 is the *add_product* function. The *F-* prefix indicates that the action is a function.

2nd. Enter F-modify_product

3rd. Enter F-delete_product

4th. Enter F-add_option

5th. Enter F-modify_option

6th. Enter F-delete_option

Press the **Commit Data** function key when you have finished these entries.

Decision Table Relationships

Menu Path

Logic, Decision Tables, Relationships

Description

This screen allows you to connect the various decision table components together. It allows you to define the combinations of questions that result in certain actions being executed.

It contains a series of *Yes*, *No*, or *Don't Care* responses to the questions and an ordered list of the actions to be performed if those results occur.

Developer		Decision Table Relationships		dtablerrships	
Decision Table	prod_opt_update	Secured	N (Y/N)		
Column Number	1	Amendment Mode	A (A/D)		
----- Question -----					
Number	Response (Y/N/-)	Possible Action	Order of Execution (1-8 or 0 to ignore)		
1	Y	F-add_product	1		
2	-	F-modify_product	0		
3	-	F-delete_product	0		
4	Y	F-add_option	0		
5	-	F-modify_option	0		
6	-	F-delete_option	0		
7	-		0		
8	-		0		
Column Definition		Add prod/base option			
Header	Question	Actions	Generate D.Table	21	52
			System Keys	Commit Data	Help
					Previous Menu

Entering the Field Values

Decision Table Name. Press

This field defaults to the name used in the last decision table screen.

Column Number. Accept the default value 1

This is the number of the column or set of relationships that you are about to define. You can define up to 31 columns.

Amendment Mode. Accept the default value **A**

The entry for this field allows you to choose what you want to do with this particular column, **A** = add or amend, and **D** = delete.

Questions

The first column of fields (*Number* column) refers to the questions defined on the decision tables questions screen. They are numbered in the same order as they were defined. The second column of fields (*Response* column) refers to the answers to the questions. The responses default to -. This means that the outcome of the question has no bearing on the column. If you enter **Y** the question must be resolved as *true* for the column to be valid. If you enter **N** the question must be resolved as *false* for the column to be valid. All eight criteria must be met for the column to be valid. Complete the question columns as shown:

Number	Response
1	Y
2	-
3	-
4	Y
5	-
6	-
7	-
8	-

This column is resolved as true when both *V-mode* is equal to *C-add* and *V-current_record* is equal to *C-product*. This is effectively stating that a product record must be added.

Actions

The next column specifies the actions that are to be performed, and the order in which they are performed. A set of display fields show the names of the actions you have already defined. The column to the right (Order of Execution) initially defaults to all 0s. The default (0) means don't do this action. The numbers specified next to the action indicate which actions are executed if the column resolves as true, and also indicate the order in which the actions are performed if more than action one is executed.

The numbers you specify must be contiguous and one number cannot occur more than once. For example, 10204000 is not valid as the 3 is missing. It should read 10203000. Similarly, the entry 00220104 is not valid as the 2 appears twice. It should be either 00230104 or 00320104.

Complete the order of execution column as shown below.

Action	Order of Exec.
F-add_product	1
F-modify_product	0
F-delete_product	0
F-add_option	0
F-modify_option	0
F-delete_option	0
	0
	0

This relationship executes the *add_product* function if the column is resolved as true.

Column Definition. Enter Add prod/base option

This is a comment field that allows you to enter a description of what the column does. It has no operational effect.

When you have entered all of the data, press the **Commit Data** function key.

To Complete the Exercise

There are five more columns to be created for this decision table. Complete the columns as shown.

Column 2

Question	Response	Action	Order
1	Y	F-add_product	0
2	-	F-modify_product	0
3	-	F-delete_product	0
4	-	F-add_option	1
5	Y	F-modify_option	0
6	-	F-delete_option	0
7	-		0
8	-		0

Column Definition: Add the Option

Column 3

Question	Response	Action	Order
1	-	F-add_product	0
2	Y	F-modify_product	1
3	-	F-delete_product	0
4	Y	F-add_option	0
5	-	F-modify_option	0
6	-	F-delete_option	0
7	-		0
8	-		0

Column Definition: Modify the Product

column 4

Question	Response	Action	Order
1	-	F-add_product	0
2	Y	F-modify_product	0
3	-	F-delete_product	0
4	-	F-add_option	0
5	Y	F-modify_option	1
6	-	F-delete_option	0
7	-		0
8	-		0

Column Definition Modify the Option

Column 5

Question	Response	Action	Order
1	-	F-add_product	0
2	-	F-modify_product	0
3	Y	F-delete_product	1
4	Y	F-add_option	0
5	-	F-modify_option	0
6	-	F-delete_option	0
7	-		0
8	-		0

Column Definition **Delete the Product**

Column 6

Question	Response	Action	Order
1	-	F-add_product	0
2	-	F-modify_product	0
3	Y	F-delete_product	0
4	-	F-add_option	0
5	Y	F-modify_option	0
6	-	F-delete_option	1
7	-		0
8	-		0

Column Definition **Delete the Option**

This completes the definition of the decision table. Generate it by pressing the **Generate D Table** function key on the decision table relationships screen.

You have now created most of the necessary dictionary items, data screens, functions and processes to drive the application.

There are a number of further items that you need to create before the application is complete. The remaining items are the messages, file update functions (the logic blocks that actually add, modify, and delete records in the appropriate files), the scrolling functions, and a report.

Messages

The following pages describe the messages and some of the remaining logic blocks required by the application.

Use the *Messages* selection on the dictionary menu to create these messages now.

Product Number Error. This message is displayed when the user enters a product number that does not fall within the range specified by the *product_no* range.

Remember that the quotes (") are required to delimit the literal contents of the message.

Name	<code>product_no_error</code>
Type	<code>ERROR</code>
Contents	<code>"Product number must be in " "the range XY1000 - XY9999"</code>

Pressing the `Commit Data` function key generates the message automatically.

Add Product. The *product_key_read* function displays this message to tell the user that the details for a new product may be entered.

Name	<code>add_prod</code>
Type	<code>MESS</code>
Contents	<code>"New product, # " * ", may be added."</code>

The * is a reference to the current screen field. Since the *product_key_read* function is called when the product number field is active, the current product number is displayed in the message.

Product Does Not Exist. This message tells the user that a product record has not been found. It is called by the *product_key_read* function. The function displays the message if the user has selected *delete*, *modify*, or *review* mode. It is an `ERROR` type message that requires the user to re-enter a valid product number.

Name	<code>prod_not_exist</code>
Type	<code>ERROR</code>
Contents	<code>"Record does not exist for product number " *</code>

Add Option. The *option_key_read* function displays this message to tell the user that the details for a new option may be added.

Name	add_option
Type	MESS
Contents	"New option, # " * ", may be added."

Option Doesn't Exist. The *option-key-read* function displays this message to indicate that a record for the option number does not exist.

Name	opt_not_exist
Type	ERROR
Contents	"Record does not exist for option number " *

No Product. The *option* function (which is called from the **Option** function key) displays this message. It tells the user that the *option* mode cannot be accessed until a valid product has been retrieved.

Name	no-product
Type	WARN
Contents	"You must retrieve a product before " "accessing any options."

Options Not For Review. The *option* function displays this message. In review mode, options can only be reviewed by pressing the **Scroll Options** function key.

Name	no_option_review
Type	WARN
Contents	"Press the [Scroll Options] function key " "to review options."

You can now test part of the application if you wish. You can select any of the modes on the main menu and display the product screen. Since you haven't created the file updating functions yet, you won't be able to perform any valid operations on the files.

The next task is to create the functions that allow you to add new records.

Add Functions

The functions that add new records to the *product* and *option* files are called by the *prod_opt_update* decision table.

Function – *add-product*

Create the *add-product* function as follows:

```

1 IF V-product_status = C-record THEN
    MESSAGE product_warn ; EXIT
2 FILE *WRITE product ; ENTER 8
3 MOVE C-record V-product_status
4 ON 1
5 MESSAGE prod_add_ok
6 VISIT add_option
7 EXIT
8 MESSAGE file_error
9 EXIT

```

Step 1 checks to see if the record to be added already exists. If it does exist, a warning message is displayed and the function exits. If the record does not exist, control passes to step 2 to add the record to the file.

Steps 3 to 5 are executed after a successful write operation. They set the current status values, turn switch 1 on and then display a message to confirm the successful addition of the product record. Then the function calls the *add_option* function to add option number 000 for the product.

If the write operation in step 2 fails, an error message is displayed by step 8 before the function exits.

The function requires two additional messages.

Product Exist Warning.

Name	product_warn
Type	WARN
Contents	"This product already exists. " "It cannot be added to the file."

Product Add OK.

Name	prod_add_ok
Type	MESS
Contents	"New product, #" F-product_no.product , has been created."

This message uses a file field reference *F-product_no.product* to include the product number in the message.

The next step is to create the function to add records to the *option* file.

Function – add_option

The *add_option* function contains the following steps:

```

1 IF V-option_status = C-record THEN
    MESSAGE option_warn ; EXIT
2 MOVE S-product_no F-product_no.option
3 LINKLOOP 2 F-product_no.option 1 F-option_key.option
4 FILE *WRITE option; ENTER 8
5 MESSAGE opt_add_ok
6 IF F-option_no.option = N-zero THEN MOVE C-option V-current_record;
    WINDOW option; MOVE "1" *NEWTIE
7 EXIT
8 MESSAGE file_error
9 EXIT

```

The *add_option* function operates as follows:

1. Like the first step of the *add_product* function, this step makes sure that the option does not already exist. If the option does exist, this step displays a warning message and the function exits.
2. This step moves the current product number into the *product_no* field on the *option* file buffer. This is necessary because the buffer was cleared before the new data was entered.
3. This step builds the unique option file key. There are several ways that this could have been achieved. This particular method uses the LINKLOOP command to build the key.

Each ... *LOOP* command performs a repeated operation on successive fields in a file record, screen, or the scratch pad. You must specify the number of

times the command is performed, the field that the operation starts on, and the step factor to indicate the next field or fields to be operated on.

In this step, the LINKLOOP command is performed twice, as shown by the first number after the LINKLOOP command. The first field is *F-product_no.option*. The step factor of 1 indicates that the next field is the next operand. The result is placed in the field *F-option_key.option*.

The same result could also be achieved with the command:

```
LINKLOOP 2 *S01 4 F-option_key.option
```

This command takes the first screen field as the first operand and links it to a screen field four fields further on. On this particular screen, this is the *option number* field.

4. The new option is written to the option file. If a file access error is detected, control passes to step 8.
5. If the file write operation in step 4 is successful, the user is informed that the file was updated.
6. If the option number written to the file was 000 then the user has just added a new product. The current record type is set to *option*. This step then displays the *option* window and sets the value in the communication area field *NEWTIE to 1. This is equivalent to the command TIE 1, and passes control to field number 1 on the *product* screen when it is next displayed.
7. The function exits.
8. If a file access error was detected in step 4 the user is informed by a message.
9. The function exits.

The *add_option* function needs the following messages.

Message – *option_warn*

Name	<i>option_warn</i>
Type	WARN
Contents	"This option already exists. " "It cannot be added."

Message – *opt_add_ok*

Name	<i>opt_add_ok</i>
Type	MESS
Contents	"New option, number " <i>F-option_no.option</i> ", has been created."

Summary

In this lesson you defined a decision table. A decision table is similar to a set of IF-THEN-ELSE statements. You can use a decision table to test a number of conditions and then execute a specified set of actions in a specified sequence depending on the results of the test.

Decision tables have the following features:

- To define a decision table, you must complete four screens. These are the decision table header screen, the decision table questions screen, the decision table actions screen, and the decision table relationships screen.
- The questions for a decision table are in the same format as logic IF statements.
- A decision table can execute any action that can be called from a menu. These include functions, processes, screens, and further decision tables.
- For any given set of outcomes for the decision table questions, the decision table relationships specify which actions are executed, and the sequence in which they are executed.
- A decision table must be generated before it can be used in the application.

In this lesson, you also created a number of functions to add records to the product and option files.

At this stage, you can test the application by adding some new records to see how the various components work together. You cannot delete or modify records, and the logic for scrolling option records does not exist yet. If you try to use any of these features, you will receive an error message.

Lesson 23 – Scrolling Data on the Screen

Objectives

In this lesson you will use the SCROLL command to display data on the scroll area of a screen.

This lesson also introduces you to some of the techniques for using multiple record layouts in conjunction with a data file.

The Scrolling System

The *product* data screen has a scroll area on it. This lesson shows you how to scroll data in this area.

The scrolling system for this application operates as follows. The first time the user presses the *Scroll Options* function key, the scroll area is cleared and the first option record for the current product is read and displayed in the scroll area. As the user continues to press the *Scroll Options* function key, subsequent option records for the current product are scrolled onto the screen. To do this, the application must maintain a file pointer to the current scroll record so the next record to be scrolled can be easily obtained.

File Record Layouts

The application scrolls records from a file which is also being used for the creation or modification of specific records. This requires the maintenance of a work buffer as well as the buffer being used to read records from the file. This is achieved by defining a second record layout for the file.

Using two record layouts for the one data file allows you to use two separate file record buffers. This means the application can read and use two different records from the same data file. Using a second record layout with a file does not make any change to the structure of the records in the physical file. The structure of the records in the physical file is always determined by the structure of the default record layout for the file. Secondary record layouts simply provide a means of reading or writing records through different file record buffers.

The structure of a secondary record layout for a file does not need to be identical to the default record layout. If required, you can obtain a different “view” of a file

record by reading it through a secondary record layout defined with different field specifications.

In this application, both record layouts for the *option* file have exactly the same structure.

Record Layout – *option_scroll*

Use the developer *copy* utility to copy the existing *option* record layout to a new layout called *option_scroll*. Remember to generate the record layout after you have copied it.

Modify the existing *option* file definition by adding the new *option_scroll* record layout to the list of record layouts as the second layout. The default record layout is still the *option* record layout.

All references to the *option* file so far have named only the file, with no record layout specifier. In this form the default record layout is always used, so the existing code remains unchanged.

The Scrolling Functions

You can now create the functions that scroll data on the screen.

Function – *scroll_options*

Pressing the **Scroll Options** function key executes the *scroll_options* function. This function calls two further functions to retrieve the first and subsequent option records for the current product.

```

1 DEFINE %FILE% option.option_scroll
2 IF V-product_status <> C-record THEN
   MESSAGE no_product; EXIT
3 IF 1 *ON THEN VISIT get_1st_option; ENTER 7
   ELSE VISIT get_next_option
4 IF 1 *OFF THEN ENTER 7
5 SCROLL 9 "***** End of Options for Product # "
   S-product_no.product " ***** "
6 EXIT
7 SCROLL 6 F-option_no.%FILE% 4 F-description.%FILE%
   5 F-cost.%FILE% 8 F-qty_on_hand.%FILE%
8 EXIT
```

The *scroll_options* function operates as follows.

1. This step introduces the **DEFINE** command. The **DEFINE** command creates an identifier that is expanded whenever it is referenced in the logic block. The identifier in this case is `%FILE%`. It is expanded to `option.option_scroll` every time it is referenced. As the file specifier is quite long, using the identifier simplifies the entry of commands. Any **DEFINE** commands must precede the other commands in a logic block. You can have more than one **DEFINE** command at the start of a logic block.
2. This step checks to ensure that a product record is current. If there is no current product record, this step displays a message and the function exits.
3. An earlier function (for example, `add_product`) may set user switch number 1 *on*. This step checks the status of the switch. If the switch is *on*, no previous option has been read for this product so it is necessary to retrieve the first option for the product. The function `get_1st_option` reads the first option record. When the `get_1st_option` function exits, control passes to step 7 of this function. Steps 4 to 6 are not executed if an option has not yet been retrieved.

If user switch 1 is *off*, an option has already been read for this product and the next one must be retrieved. The function `get_next_option` reads the next option record.

4. This step is only executed after a previous option has been read for this product. The function `get_next_option` sets user switch 1 *on* if no more options for the product are found. If the switch is *off*, a record has been found and control passes to step 7 where the data is scrolled on the screen.
5. This **SCROLL** command is executed when the last of the options has been found. The **SCROLL** command takes the specified parameters and concatenates them to form a line of text that is displayed in the scroll area. Any lines already displayed in the scroll area are moved one line in the defined scroll direction and the new line of text is displayed on the first line of the scroll area.

The first **SCROLL** parameter is 9. This is displayed as a string of nine spaces so the second **SCROLL** parameter, a text literal, is preceded by nine spaces. The third **SCROLL** parameter is a screen field and the last **SCROLL** parameter is a concluding literal. Separate the parameters with spaces when you enter the command.

7. This is the step that actually scrolls the option data on the screen. This step shows the use of the identifier %FILE% that was defined in step 1. For example, the second parameter:

F-option_no.%FILE%

is expanded to read:

F-option_no.option.option_scroll

In the SCROLL command, each of the file fields is separated by a number of spaces so they are aligned with the column headings painted on the screen.

Function – *get_1st_option*

The next function that you need to create is the *get_1st_option* function. This function is called by the *scroll_options* function to retrieve the first option record for the current product. Since the processing rules for this application require that each product must have at least one option (number 000) this function should always retrieve an option record. Failure to find a record indicates a serious file problem. The steps in this function are:

- 1 **DEFINE** %FILE% option.option_scroll
- 2 **DISPLAY** *RESET=S ""
- 3 **FILE** *FIND %FILE% *KEY= F-product_no.product ; **ENTER** 9
- 4 **FILE** *NEXT %FILE%
- 5 **IF** F-product_no.%FILE% <> F-product_no.product
 THEN ENTER 9
- 6 **SCROLL** 9 "***** Start of Options for Product # "
 S-product_no.product " *****"
- 7 **OFF** 1
- 8 **EXIT**
- 9 **MESSAGE** prod_corrupt
- 10 **PROCEED** prod_opt; **NOTE** Product/option record corrupted
 since option 000 does not exist. Clear buffers and display
 the prod/opt menu so the user can delete the product.

The *get_1st_option* function operates as follows:

1. The **DISPLAY** command is another command used to manipulate the scroll area on a screen. It displays a line of data on a specific line in the scroll area. The ***RESET=S** option clears the scroll area. The empty parameter ("") specifies that nothing is displayed on the first line of the scroll area.

2. The FILE *FIND command searches a file for a record with a key greater than or equal to the value specified. As the *option_key* field in this application is the concatenation of the *product_no* and the *option_no* fields, a search with the key equal to the product number finds the first option for that product. If no suitable record is found, control passes to step 9.
3. The FIND command does not read the record contents into the file buffer. It simply positions a file pointer to the record. You must use a FILE *NEXT command to read the record into the file record buffer.
4. Since the FILE *FIND command searches for a key value equal to or greater than the specified key, it may find a record with a greater product number than the one specified in step 3. This step checks to see if the record retrieved by step 4 has the correct value in the product number field. If the value in the product number field does not match the current product number, control passes to step 9.
5. The SCROLL command scrolls a heading line in the scroll area.
6. This command sets user switch number 1 to off. The calling function (*scroll_options*) tests the status of this switch when the user next presses the **Scroll Options** function key. If this switch is off, the next option is retrieved.
7. The function exits.
8. Control only passes to this step if an option record for the product has not been found. This condition is detected if an error occurs in step 3, or the product number in the record retrieved from the file is not the current product number. Failure to find an option record means that the record for option 000 does not exist, suggesting that the product or option file is corrupted. This step displays a message telling the user that records may be corrupted.
9. Control only passes to this step if the record for option number 000 cannot be found. It demonstrates a method for recovering from a potentially serious problem. The PROCEED command in this step immediately terminates any current activity in the application, clears the file and screen buffers, and closes all data files.

The NOTE command allows you to describe individual lines within a logic block. HP ALLBASE/4GL ignores all the text following a NOTE command.

The process *prod_opt* displays the menu *prod_opt_menu* to allow the user to delete or correct the corrupted record.

Function – *get_next_option*

The last function to create is the *get_next_option* function. This function is called by the *scroll_options* function to retrieve the second and subsequent options for the current product. It contains the following commands:

```

1 DEFINE %FILE% option.option_scroll
2 FILE *READ %FILE% ; ENTER 7
3 FILE *NEXT %FILE% ; ENTER 5
4 IF F-product_no.%FILE% = F-product_no.product THEN EXIT
5 ON 1
6 EXIT
7 ON 1
8 VISIT get_1st_option
9 EXIT

```

This function operates as shown below.

1. This defines an abbreviation for the *option.option_scroll* reference.
2. Between the time of the scrolling of the last record and the current scroll request, the user may have read the option file via the default buffer. This will not directly affect the data in the second buffer but it may affect the file pointer. This step relocates the file pointer by reading the file for the contents of the current scroll record buffer. However, this step will fail if the last scrolled record has been deleted. Control passes to step 7 if this situation occurs.
3. This next record is retrieved from the file.
4. This step tests the new record to ensure that it refers to the correct product by testing the value in the *product_no* field. If the retrieved value is correct, the function exits with the next option details in the correct buffer.
5. This step sets user switch number 1 *on* if the end of the file is reached in step 3, or the product number changes as determined in step 4. This indicates to the calling function that the last option for the product has been found.
6. The function exits.
7. This step and the next step are only executed if the original record is not found again in step 2. This step sets switch number 1 *on*.
8. This step attempts to locate the first option for the product by visiting the function *get_1st_option* again. The return from this step is the same as if the calling function visited *get_1st_option* directly.
9. The function exits.

Message – *prod_corrupt*

The *get_1st_option* function requires the following message:

Name	prod_corrupt
Type	WARN
Contents	"Product record may be corrupted. " "Delete and enter again."

Process – *prod_opt*

The *get_1st_option* function calls the *prod_opt* process if the product record is corrupted. This process clears all file and screen buffers and then displays the *prod_opt_menu* screen. The user can then select another operation from the menu to delete the product or correct the problem. The process is a simple one. The SCREEN command calls the *prod_opt_menu*.

```
1 SCREEN prod_opt_menu
```

These functions complete the processing required for a fail safe scrolling mechanism.

When you have created and generated all three of these functions you can test them through the add or review options on the main menu.

In the next lesson, you create the functions that allow the user to modify and delete records.

Summary

In this lesson you used the SCROLL command to display data in the scroll area of a screen.

You also associated a secondary record layout with a data file. This enables the application to read and use two different records from the same data file.

This lesson also introduced the FILE *FIND command, and the FILE *NEXT command. The FILE *FIND command searches a file for a record with a key equal to or greater than a specified value. You can then use the FILE *NEXT command to retrieve the record.

Lesson 24 – The Modify and Delete Functions

Objectives

In this lesson you will create the functions that modify and delete records in the product and option files.

The Modify Functions

The two functions that modify the product and option records are similar to the functions that add records to the files. These functions are quite straightforward, so the following paragraphs simply list the functions and their associated messages.

Function – *modify-product*

```
1 IF V-product_status = C-no-record THEN
    MESSAGE no_prod_warn ; EXIT
2 FILE *WRITE product ; ENTER 5
3 MESSAGE prod_modify_ok
4 EXIT
5 MESSAGE file_error
6 EXIT
```

The *modify-product* function requires the following messages:

Message – *no_prod_warn*

Name	no_prod_warn
Type	WARN
Contents	"No product for modification."

Message – *prod_modify_ok*

Name	prod_modify_ok
Type	MESS
Contents	"Product # " F-product_no.product ", has been modified."

Function – *modify_option*

```

1 IF V-option_status = C-no_record THEN
    MESSAGE no_option_warn ; EXIT
2 FILE *WRITE option ; ENTER 5
3 MESSAGE opt_modify_ok
4 EXIT
5 MESSAGE file_error
6 EXIT

```

The *modify_option* function requires the following two messages:

Message – *no_option_warn*

Name	no_option_warn
Type	WARN
Contents	"No option for modification."

Message – *opt_modify_ok*

Name	opt_modify_ok
Type	MESS
Contents	"Option # " F-option_no_option ", has been modified."

The Delete Functions

You must create four functions to delete product and option records. The processing for this operation is a little different from earlier parts of the application. The delete functions require the user to verify that the selected option or product is the correct item to be deleted. You will use a query message to do this.

When the user selects a product for deletion, the application first checks to see if there are any options other than option number 000 associated with the product. If options do exist, the user is alerted and asked to confirm the deletion of the options.

The user must delete the options before the product can be deleted. When a product is deleted, option number 000 for the product is also deleted.

If option number 000 for a product does not exist when it is first accessed, the user is alerted and given the option of deleting the product.

Function – *delete_product*

The *delete_product* function contains the following steps:

```

1 DEFINE %PROD% .product
2 IF V-product.status = C-no-record THEN
    MESSAGE no-prod.warn ; EXIT
3 LINK 2 F-product.no%PROD% "000" F-option.key.option
4 FILE *READ option ; ENTER 17
5 VISIT del.all.options
6 IF 8 *ON THEN ENTER 15
7 MESSAGE delete.product
8 IF V-confirm <> "Y" THEN ENTER 15
9 SERIES 3 3
10 FILE *DELETE option ; EXIT
11 FILE *DELETE product ; EXIT
12 MESSAGE prod.delete.ok
13 MOVE C-no-record V-product.status
14 EXIT
15 MESSAGE prod.delete.fail
16 EXIT
17 VISIT del.corrupt.prod
18 EXIT

```

The *delete_product* function operates as follows:

1. This defines an identifier for the *product* file.
2. This step ensures that a product record is current.
3. This step links the current product number and the number *000* into the key field on the *option* file record buffer.
4. This step reads the option file for a record with the key value set up in step 3. If the record is not found, the file may be corrupted. Every product must have an option number 000. Control passes to step 17 if an error is detected.
5. The *del.all.options* function deletes all of the options for the current product. The *del.all.options* uses switch number 8 to indicate its exit status. If switch number 8 is *off* when the function exits, all options other than option number 000 for the current product have been deleted successfully.
6. This step tests the status of switch number 8. If this switch is *on*, the options for the product have not been deleted. Control passes to step 15 to display a message to the user.

7. The message *delete_product* is a query message that asks the user to confirm the deletion of the product. The user's response is returned to the variable *confirm*.
8. If the user's response to the query message is "Y", control passes to step 9. If the user's response is anything else, control passes to step 15.
9. This step uses the **SERIES** command to execute step 3 again to rebuild the key for option number 000 for the current product. This is necessary since the *del_all_options* function may change the *option* file buffer contents.
10. The record for option 000 is deleted.
11. The product record is deleted.
If a file error is detected in either of these steps, the function exits immediately. In this case, control returns to the *product* process, and any currently open transaction is reversed by the **TRANSACTION *UNDO** command. You may want to look back at the *product* process to see how this is done.
- 12, 13, 14. These steps display a confirmation message for the user and set the *product_status* variable. The function then exits.
- 15, 16. Control passes to these steps if the user does not confirm the deletion of the product, or the options for the product are not deleted successfully. These steps display a message, and then exit.
17. Control passes to this step if the first read of the *option* file does not succeed. This can only occur if option number 000 for the current product does not exist, indicating corruption of the *option* file or the *product* file. This step visits the function *del_corrupt_prod* to delete the current product record, and any option records for the product.
18. The function exits.

The *delete_product* function requires the following messages:

Message – *delete_options*

Name	<i>delete_options</i>
Type	QUERY
Response Item	V-confirm
Contents	"This product has options. " "Enter 'Y' to confirm their deletion."

Message – *delete_product*

Name	delete_product
Type	QUERY
Response Item	V-confirm
Contents	"Enter 'Y' to confirm " "deletion of this Product."

Message – *prod_delete_ok*

Name	prod_delete_ok
Type	MESS
Contents	"Product has been deleted."

Message – *prod_delete_fail*

Name	prod_delete_fail
Type	WARN
Contents	"Product has NOT been deleted."

Function – *del_all_options*

The *del_all_options* function deletes all options for a product except option number 000. It is called from the *delete_product* function.

The *del_all_options* function uses switch number 8 to indicate its exit status. If the options for a product are deleted successfully, switch number 8 is set *off* when the function exits. Otherwise, switch number 8 is set *on*.

The *del_all_options* function contains the following commands.

```

1 ON 8
2 FILE *NEXT option ; ENTER 11
3 IF F-product_no.option <> F-product_no.product
  THEN OFF 8 ; EXIT
4 MESSAGE delete_options
5 IF V-confirm <> "Y" THEN EXIT
6 SCROLL 6 F-option_no.option 4 F-description.option 8 "DELETED"
7 FILE *DELETE option
8 FILE *NEXT option ; ENTER 11
9 IF F-product_no.option <> F-product_no.product THEN ENTER 12
10 ENTER 6
11 IF *IOSTATUS <> N-end_of_file THEN MESSAGE file_error ; EXIT
12 SCROLL 9 "***** All options deleted for product # "
```

```
F-product_no.product " *****"  
13 OFF 8  
14 EXIT
```

The *delete_all_options* function operates as follows:

1. This step turns switch number 8 *on*. If the function deletes the option records successfully, this switch is set *off* before the function exits.
2. The FILE *NEXT command reads the next record from the *option* file. Since this function is called after the FILE *READ command in the *delete_product* function, the FILE *NEXT command attempts to retrieve the record for option number 001. If the FILE *NEXT command encounters an error, control passes to step 11.
3. This step tests the value in the *product_no* field on the record just read. If this value is not equal to the product number of the current product, the record just read is an option record for a different product. In this case, the current product does not have any options other than option number 000, so no options require deletion. In this case, switch number 8 is set *off* and the function exits.

If the record just read is an option record for the current product (that is, the value in the *product_no* field matches the current product number), the product must have at least one option other than option number 000.

4. This step displays a query message that asks the user to confirm the deletion of the options for this product.
5. If the user enters "Y" in response to the message, control passes to step 6. Otherwise, the function exits.
6. Steps 6 through 10 form a loop that deletes all the option records for the current product.

This command displays a line of data containing six spaces, the current option number, four further spaces, the description of the option, eight further spaces, and the word "DELETED".

7. This step deletes the current *option* file record.
8. This step reads the next record in the *option* file. If an error is encountered, control passes to step 11.
9. If the value in the *product_no* field in the record just read does not match the number for the current product, the record just read is an option record for a different product. This means all options for the current product have been

deleted. In this case, control passes to step 12. Otherwise, control passes to step 10 which returns control to step 6 to continue the option deletion loop.

10. Control passes to step 6. The option deletion loop continues until there are no options for the current product left in the *option* data file. Control passes to step 12 when the last option record for the current product has been deleted.
11. Control passes to this step if an error occurs in either of the FILE *NEXT commands. This step tests the value in *IOSTATUS. If this value is not equal to 19110 (the value of the constant *end_of_file*), indicating the FILE *NEXT command encountered an end of file condition, control passes to step 12. An end of file condition may occur after the last record in the file has been deleted. This situation does not indicate an incorrect condition.

If the file error is caused by any condition other than encountering an end of file condition, then a message is displayed the function exits.

12. The user is informed that all of the options for the current product have been deleted.
13. This step sets switch number 8 *off* to indicate that all options for the current product have been deleted successfully.
14. The function exits.

Function – *del_corrupt_prod*

The *del_corrupt_prod* function is called from the *delete_product* function if the first read of the *option* file fails. This can only occur if the record for option number 000 does not exist, indicating that the record for the product or option is corrupted.

The *del_corrupt_prod* function deletes the product and option records for a corrupted product record.

The *del_corrupt_prod* function contains the following commands.

- ```

1 MESSAGE corrupt_prod_del
2 IF V-confirm <> "Y" THEN MESSAGE prod_delete.fail ; EXIT
3 FILE *FIND option *KEY= F-product.no.product ; ENTER 14
4 FILE *NEXT option ; ENTER 14
5 IF F-product.no.option <> F-product.no.product
 THEN ENTER 9
6 SCROLL 2 F-option.no.option 4 F-description.option
 8 "DELETED"
7 FILE *DELETE option

```

```
8 ENTER 4
9 FILE *DELETE product ; ENTER 12
10 MOVE C-no_record V-product_status
11 EXIT
12 MESSAGE file_error
13 EXIT
14 IF *IOSTATUS = N-record_not_found | *IOSTATUS = N-end_of_file
 THEN ENTER 9 ELSE MESSAGE file_error ; EXIT
```

The *del\_corrupt\_prod* function operates as follows.

1. This step displays a query message asking the user to confirm the deletion of the product and options.
2. If the user enters “Y” in response to the message, control passes to step 3. Otherwise, the *prod\_delete\_fail* message is displayed and the function exits.
3. This step uses the FILE \*FIND command to search the file for a record with a key value equal to or greater than the key specified. In this case, the command uses the current product number as the key for the search. This command finds the first existing option record for the current product since the option key is a concatenation of the product number and the option number.  
If the FILE \*FIND command does not find a record, control passes to step 14.
4. The FILE \*NEXT command in this step retrieves the record into the file buffer. Control passes to step 14 if the record is not retrieved successfully. Steps 4 through 8 form a loop that deletes all the existing options for the current product.
5. This step tests the value in the *product\_no* field for the record just read. If the record just read is an option record for a different product, no option records exist for the current product or all option records for the product have been deleted. In this case, control passes to step 9 to delete the product record.
6. This step scrolls the current option details, then the word “DELETED” in the scroll area of the current screen.
7. This step deletes the current record from the *option* file.
8. Control is passed to step 4 to continue the option deletion loop.
9. Control passes to this step after the last option record for the current product has been deleted. This step deletes the current product record for the *product* file. If an error occurs during this operation control passes to step 12.

10. This step sets the value of the variable *product\_status* to indicate that there is no current product record.
11. The function exits at this point.
12. Control passes to this step if a file access error is detected in step 9. This step displays the file error message.
13. The function exits.
14. Control passes to this step if a file access error is detected in step 3 or step 4. If the file error is caused by failure to find a record or reaching the end of the file, control passes to step 9 to delete the current product record. This situation can occur if the first FILE \*FIND command fails to find any options for the current product, or the FILE \*NEXT command in step 4 reaches the end of file after the last record in the file has been deleted.

If the file access has failed for any other reason the *file\_error* message is displayed and the function exits.

You must now create the *corrupt\_prod-del* message. This message is displayed by the *del\_corrupt\_prod* function.

#### Message – *corrupt\_prod-del*

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| Name          | <i>corrupt_prod-del</i>                                                        |
| Type          | QUERY                                                                          |
| Response Item | V-confirm                                                                      |
| Contents      | "Product/option records corrupted. "<br>"Enter 'Y' to confirm their deletion." |

This completes the items for the product deletion part of this application.

## Deleting Options

### Function – *delete\_option*

The option deletion operation is more straightforward and should not require any explanation. The *delete\_option* function contains the following steps:

- 1 IF V-option\_status = C-no-record THEN  
    MESSAGE no\_option\_warn ; EXIT
- 2 MESSAGE delete\_option
- 3 IF V-confirm <> "Y" THEN MESSAGE opt\_delete\_fail ; EXIT
- 4 FILE \*DELETE option

```
5 MESSAGE opt_delete_ok
6 MOVE C-no-record V-option_status
7 EXIT
```

### Message – *delete\_option*

The *delete\_option* function requires the following messages:

|               |                                                       |
|---------------|-------------------------------------------------------|
| Name          | delete_option                                         |
| Type          | QUERY                                                 |
| Response Item | V-confirm                                             |
| Contents      | "Enter 'Y' to confirm "<br>"deletion of this option." |

### Message – *opt\_delete\_ok*

|          |                            |
|----------|----------------------------|
| Name     | opt_delete_ok              |
| Type     | MESS                       |
| Contents | "Option has been deleted." |

### Message – *opt\_delete\_fail*

|          |                                |
|----------|--------------------------------|
| Name     | opt_delete_fail                |
| Type     | WARN                           |
| Contents | "Option has NOT been deleted." |

---

## Summary

In this lesson you created the functions to modify and delete records in the product and option file.

The data manipulation components of this application are now complete. You should be able to use the application to add, delete, and modify product and option records according to processing rules for the application.

## Lesson 25 – Further Reporting Techniques

---

### Objectives

In this lesson you will create a report that uses data from both the product file and the option file. In developing this report, you will use the following reporting features.

- A start of report function.
- Sort field definitions and control breaks.
- Automatic totalling.
- Record selection criteria.
- File linkages and a post link read function.
- A before print function associated with a report line group.

You will also use a calculated item in this report. A calculated item is a special type of variable that is evaluated by executing a function every time the item is referenced.

These facilities allow you to develop reports that are considerably more flexible and powerful than the simple report in the earlier part of this training course.

---

### The Product Report

The product/option report that you are about to develop uses a start of report function to display a screen. This screen allows the user to enter the starting and finishing values for a range of product numbers for the report.

The report then lists each of these products and the options for each product. The report calculates the cost of stock for each option by multiplying the quantity on hand by the cost price. When all options for a product have been listed, the total cost of stock for that product is calculated and printed. Finally, the report calculates the total cost of stock for all products listed.

## Report Header

Call up the report header screen and enter the data shown below. Accept the default entries for any fields not listed here.

|                               |                        |
|-------------------------------|------------------------|
| Report Name                   | <b>product</b>         |
| Report File Designator        | <b>prod</b>            |
| Primary File[.Record]         | <b>product</b>         |
| Index                         | <b>1</b>               |
| Characters per line           | <b>100</b>             |
| Start of Report Function Name | <b>select_products</b> |

The last field is one that you haven't used yet. It is the name of a function that is executed when the report starts. In this application, the function displays a screen that asks the user for the range of product numbers to be included in the report. The report is produced when this function finishes.

Create the *select\_products* function and its associated data screen now.

### Screen – *select\_products*

The screenshot shows a terminal window with a header bar containing the text: "Training Application", "Product Report Selection", and "select\_products". Below the header, on the left, is the time "06:43:41" and on the right, the date "10/11/88". The main area of the screen is mostly blank, with a central box containing two input fields: "Lower Limit" and "Upper Limit", each followed by a shaded rectangular area representing a text input field. At the bottom of the screen, there is a row of buttons: "System Keys", "Commit Data", "Help", and "Previous Menu".

This data screen requires two data input fields. Neither field uses any data movement or logic. To create the screen, define the screen header and paint the image. Create both fields using the dictionary field specification *product\_no* to ensure

that automatic product number range checking is done. You don't need to enter any screen field details. Remember to generate the screen.

### **Function – *select\_products***

This function is called as the start of report function for the product report. It executes the *select\_products* screen, retrieves data from the screen, and initializes some items for use later in the report. The function contains the following steps:

```

1 SCREEN select_products
2 IF *S01 >= *S02 THEN MOVELOOP 2 *S01 1
 *P02 -1 ELSE ENTER 5
3 MOVELOOP 2 *P01 1 *S01 1
4 SHOW
5 MOVE " " *P03
6 OFF *ALL
7 EXIT

```

The function operates as follows.

1. This step executes the *select\_products* data screen. This screen allows the user to enter the range of products to be included in the report. When the user commits the screen, this step terminates and the next step is executed. The *start of report* function is the only function in a report that can display a screen.
2. This step checks whether the lower limit value is less than the upper limit value. If it is, control passes to step 5.

If the lower limit value is greater than the upper limit value, this step swaps the two values. The MOVELOOP command moves the two screen fields to two scratch-pad fields.

Scratch-pad fields are general purpose working variables that assume the attributes and value of the data moved into them. If the source is numeric, the scratch-pad field becomes numeric. If the source is alphanumeric, the scratch-pad field is treated as an alphanumeric variable. You can reference scratch-pad fields by number in the form \*P01 to \*P99. You can also assign names to scratch-pad fields using the *Scratch-Pad Fields* screen in the dictionary menu. Refer to the *HP ALLBASE/4GL Developer Reference Manual* for more details on naming and accessing scratch-pad fields.

In this case the MOVELOOP command is performed twice. The first screen field, \*S01, is moved to the second scratch-pad field \*P02. Then the second screen field, \*S02, is moved to the first scratch-pad field \*P01. Note that

the step factor for the screen fields is positive, while the step factor for the scratch-pad fields is negative.

3. This step moves the contents of the scratch-pad fields back to the screen fields in reverse order. This swaps both the values. Note that both step factors are positive.
4. This step redisplay the screen field buffers. Step 3 moves values directly into the internal screen field buffers so this step displays the new values. The swap is complete and the user has seen the change on the screen.
- 5, 6, 7. These steps initialize scratch-pad field \*P03 with a single space and initialize all the user switches for later use in the report. The function then exits.

Create and generate this function now.

---

## Report Sorting

The report sorting levels determine the order in which the report items are listed, and also define control break levels.

Control breaks allow you to perform a variety of sub-heading and sub-totalling tasks. A control break for a given sort level occurs each time the value in a sort field value changes from one record to the next. This report uses one control break to print the subtotal line after each product.

### Menu Path

Reports, **Sorting**

### Description

This screen allows you to define up to eight sort fields, giving you a total of eight control break levels. The sort fields must be data fields on the primary file that you specified in the report header. For each sort field, you must state whether the sort is to be in ascending or descending order. You can also specify whether the file is already in sequence.



| Developer                            |               | Report Sorting |                         | report_sort |               |
|--------------------------------------|---------------|----------------|-------------------------|-------------|---------------|
| Report name                          | product       | Secured        | N                       | (Y/N)       |               |
|                                      | Field Name    | ORDER          | Ascending or Descending |             |               |
| 1st                                  | product_no    | A              | (A/D)                   |             |               |
| 2nd                                  |               |                | (A/D)                   |             |               |
| 3rd                                  |               |                | (A/D)                   |             |               |
| 4th                                  |               |                | (A/D)                   |             |               |
| 5th                                  |               |                | (A/D)                   |             |               |
| 6th                                  |               |                | (A/D)                   |             |               |
| 7th                                  |               |                | (A/D)                   |             |               |
| 8th                                  |               |                | (A/D)                   |             |               |
| Is File already in sequence? Y (Y/N) |               |                |                         |             |               |
| Header                               | Record Select | Line Header    | Generate Report         | 20 38       | System Keys   |
|                                      |               |                |                         |             | Commit Data   |
|                                      |               |                |                         |             | Help          |
|                                      |               |                |                         |             | Previous Menu |

## Entering the Field Values

**Report Name.** Enter product

This is the name of the report that you are defining.

### Sort Fields

The next 16 fields define the eight sort fields and their sorting order. The sort field numbers *1st* through to *8th* specify the control break level and relate directly to subheading line groups *H1* – *H8* and subtotal *T1* – *T8* line groups.

The sorting order is either ascending or descending. The collating sequence is listed in the *HP ALLBASE/4GL Developer Reference Manual*.

This report only uses the *1st* sort field.

**1st Field Name.** Enter product\_no

This is the name of the field on the primary data file which forms the report's first sort level and control break.

For this report, a control break occurs every time a product record is read from the primary report file.

**ORDER.** Enter A

This specifies that the records are to be sorted in ascending order.



## Entering the Field Values

**Report Name.** Enter product

This is the name of the report that you are defining.

**After Selection Function.** Leave this field blank.

This is the name of the function that is to be executed after each record is read.

HP ALLBASE/4GL sets the switch *\*BYPASS* *on* if the current record read from the file fails selection, or *off* if the record passes selection. You can use this function to perform your own manual selection by changing the status of the *\*BYPASS* switch. You can use this screen to specify just an after selection function and carry out record selection entirely from within this function.

**Selection Number.** Enter 1

This number identifies this set of selection criteria.

**Action.** Accept the default of A

This is the standard action code indicating add, change, insert, or delete.

**Link.** Leave this field blank.

If the selection criteria is to be applied to a link file, you must specify the link number in this field. You will learn about defining links to other files soon.

When you leave this field blank, the link file displayed in the next field is the primary data file *product*. By default this set of selection criteria is applied to the primary report file.

**Field Spec. Name.** Enter *product\_no*

This is the name of the field that is tested by this set of selection criteria. The field must exist on the link file displayed in the previous field.

**Values FROM.** Enter *\*S01*

This is the lower limit value of the field. In this case it is the first screen field of the *select\_products* data screen. The entry can be a number, a literal, a validation table, or range. The *HP ALLBASE/4GL Developer Reference Manual* lists the possible field types that may be used.

**Values TO.** Enter *\*S02*

This is the upper limit value of the field. In this case it is the second screen field of the *select\_products* data screen. The range specified by the upper and lower limits includes the *FROM* and *TO* values.

Press the **Commit Data** function key to create the selection criteria.

Then press the **Line Header** function key to start defining the line headers for the report.

## Line Headers

You have already defined the header details for each print line in the previous report. Only those fields which require specific entries are shown here. Accept the default entry for any fields that are not listed here.

**P1.01** The first heading line.

|             |    |
|-------------|----|
| Line Group  | P1 |
| Line Number | 01 |
| Action      | A  |

**P1.02** The second heading line.

|             |    |
|-------------|----|
| Line Group  | P1 |
| Line Number | 02 |
| Action      | A  |

**D1.01** This is the first detail line. It contains column headings for the product details.

|                   |    |
|-------------------|----|
| Line Group        | D1 |
| Line Number       | 01 |
| Action            | A  |
| Skip Lines Before | 2  |

**D1.02** This is the second detail line. It contains the details for a single product.

|                  |    |
|------------------|----|
| Line Group       | D1 |
| Line Number      | 02 |
| Action           | A  |
| Skip Lines After | 2  |

**D1.03** This is the third detail line. It contains column headings for the option details.

|             |    |
|-------------|----|
| Line Group  | D1 |
| Line Number | 03 |
| Action      | A  |

**D2.01** This is the first line of the second detail group. This line contains the option details and is printed as a result of a link to the *option* file. You will learn more about file linkages shortly.

|                       |              |
|-----------------------|--------------|
| Line Group            | D2           |
| Line Number           | 01           |
| Action                | A            |
| Before Print Function | print_option |

The before print function *print\_option* tests to see if any options were found and it prints a message if none were found. You will create the function and message soon.

**T1.01** This is the first subtotal line. It is printed when a change is detected in the first sort level value, in this case the *product\_no* field. As each product number is unique, this line is printed after the details of each product have been printed.

|                   |    |
|-------------------|----|
| Line Group        | T1 |
| Line Number       | 01 |
| Action            | A  |
| Underline Numbers | B  |

**TF.01** This is the final total line.

|                   |    |
|-------------------|----|
| Line Group        | TF |
| Line Number       | 01 |
| Action            | A  |
| Skip Lines Before | 2  |
| Underline Numbers | B  |

**E1.01** This extra line is printed from the *print\_option* function mentioned above. It tells the user that no options were found for the particular product just printed.

|             |    |
|-------------|----|
| Line Group  | E1 |
| Line Number | 01 |
| Action      | A  |

You have now completed all the line headers required for this report.

---

## Report File Linkages

This report uses a file linkage to read records from the *option* file whenever a product record is printed. After the product details have been printed, the link is completed by printing all the matching records from the *option* file.

## Menu Path

Reports, File Linkages

## Description

This screen allows you to define a link to another file from a particular line group. Each time the specified line group is printed the link is performed. You can define up to 999 links for a report.

| Developer                  |                      | File Linkages          |                    |              |                | report-link        |             |      |               |
|----------------------------|----------------------|------------------------|--------------------|--------------|----------------|--------------------|-------------|------|---------------|
| Report Name                | product              |                        |                    | Secured      | N (Y/N)        |                    |             |      |               |
| Link Number                | 1                    |                        |                    |              |                |                    |             |      |               |
| Action                     | A (A/C/I/D)          |                        |                    |              |                |                    |             |      |               |
| Link From Line Group       | Print Line Group     | End of Link Line Group | Post-Read Function |              |                |                    |             |      |               |
| D1                         | D2                   |                        | option-present     |              |                |                    |             |      |               |
| Link File[.Record]         | option               |                        |                    | Index Number | 2              |                    |             |      |               |
| *KEY=                      | F-product.no.product |                        |                    |              |                |                    |             |      |               |
| Must the Record be Present | N (Y/N)              |                        |                    |              |                |                    |             |      |               |
| Number                     | Act                  | Link-From              | Print              | End          | Function       | Link File[.Record] |             |      |               |
| 1                          | A                    | D1                     | D2                 |              | option-present | option             |             |      |               |
| Header                     | Record Select        | Line Header            | Generate Report    | 15           | 35             | System Keys        | Commit Data | Help | Previous Menu |

## Entering the Field Values

**Report Name.** Enter product

This is the name of the report that you are creating.

**Link Number.** Enter 1

This screen allows you to define the links one at a time so this number becomes the identifier for the linkage. This identifier is also used on the selection criteria screen if you specify selection criteria for a link.

**Action.** Enter A

This is the standard action code that allows you to add, change, insert, or delete a linkage specification.

**Link From Line Group.** Enter D1

A link is always initiated by a particular line group. Any links defined for a line group are initiated before the line is printed. Therefore, the details from the link file record are available for inclusion when the line group is printed.

**Print Line Group.** Enter D2

A linkage allows you to print another line group after the *Link From* line group has been printed. This line group is printed for each record selected from the link file.

**End of Link Line Group.** Leave blank.

Once the link is completed and all of the relevant records have been accessed and printed, you can print a further link line group. For example, you could use this line to print some additional summary data. This additional line group is printed at the end of the link.

**Post-Read Function.** Enter option-present

This function is executed each time a record is read from the link file. This allows you to perform a number of actions. For example, you could manually terminate the link by setting the \*ENDLINE switch *on*, or you could manipulate the data retrieved for use in the report. In this report, the function determines if any options have been found for the current product.

**Link File[.Record].** Enter option

This is the name of the file that you want to access for the link.

**Index Number.** Enter 2

This is the index number that is used to read the link file. A link file can only be read through a key field. The *product\_no* field is defined as key field number 2 for the *option* file.

**\*KEY=** Enter F-product\_no.product

This is the key value that is used to access the link file for the first link read. The link ends when the next record read no longer matches the key value, the end of the file is reached, or the switch \*ENDLINE is set *on*.

**Must the Record be Present.** Enter N

This setting takes effect when an unsuccessful attempt is made to read the first link record. When this field is set to *N*, the file record buffer is cleared and the report continues normally. If this field is set to *Y*, the buffer is cleared, the line group that initiated the link isn't printed, the link terminates, and the report

continues. If the file error was not *record not found* or *end of file* then the report aborts. Refer to the *HP ALLBASE/4GL Developer Reference Manual* for further details about specifying critical links.

Press the **Commit Data** function key to create the file linkage definition.

You can now define the functions that are called after the link file is read, and before the D2.01 line is printed.

### Function – *option\_present*

This function is executed each time an attempt is made to link to the *option* file to determine if any option has been found. It contains the following steps:

- 1 **IF** F-product.no.product = \*P03 **THEN EXIT**
- 2 **IF** \*IOSTATUS <> "00000" **THEN ON 1; EXIT**
- 3 **MOVE** F-product.no.product \*P03
- 4 **EXIT**

The *option\_present* function operates as described below.

1. This step determines if this is the first time an option has been read for this product. \*P03 was initialized to spaces in the *start\_of\_report* function to set this condition for the first time the function is executed. If the current product number is equal to the scratch-pad field, this function has already been executed and an option for the product has been read. In this case, the function exits immediately. Otherwise, this function tests if the first option has been found.
2. This step tests the status of the \*IOSTATUS communication area field. HP ALLBASE/4GL sets \*IOSTATUS whenever the link file is read to allow you to determine the success or otherwise of the file access. If it is not zero, user switch number 1 is set on and the function exits. The use of this switch will be demonstrated shortly.
3. This step moves the current product number into scratch-pad \*P03 so the test in step 1 can be performed the next time this function is executed.

Create and generate this function now.

### Function – *print\_option*

This function is executed immediately before the D2.01 line (the option details line) is printed. If no option for the product has been found, it prints the E1 line group instead. The function contains the following steps:

- 1 **IF** 1 \*ON **THEN PRINT** E1 ; **ON** \*ENDLINE ; **OFF** 1
- 2 **EXIT**



1. The *option-present* function sets the user switch 1 *on* if the file access for the first record fails. This step tests the switch and performs a number of tasks if it is *on*. First the E1 line group is printed. This is a single line that states that no options were found for the product and the product or option files may be corrupted. Then the \*ENDLINE switch is set *on*. This terminates the current line group (D2) so the line is not printed on the report. Then user switch 1 is reset to the *off* state. This is a standard technique to substitute another print line for the current print line.
2. The function then exits.

You can now create the image of the report.

## Painting the Report

The following illustration shows the layout of the line groups and fields for this report.

|         |                                                                                              |                               |   |   |                 |            |           |                    |                |                    |
|---------|----------------------------------------------------------------------------------------------|-------------------------------|---|---|-----------------|------------|-----------|--------------------|----------------|--------------------|
|         | 1                                                                                            | 2                             | 3 | 4 | 5               | 6          | 7         | 8                  | 9              | 10                 |
|         | 1234567890123456789012345678901234567890123456789012345678901234567890                       |                               |   |   |                 |            |           |                    |                |                    |
| P1.01   | Product and Options Stock Report as at                                                       |                               |   |   |                 |            |           |                    | AAAAA          | Page: NNNN         |
| P1.02   | on the                                                                                       |                               |   |   |                 |            |           |                    | AAAAA          |                    |
| D1.01   | Product #                                                                                    | Description                   |   |   |                 | Supplier   | Lead Time |                    |                |                    |
| D1.02   | AAAAA                                                                                        | AAAAAAAAAAAAAAAAAAAAAAAAAAAAA |   |   |                 | NNNNN      | NN        |                    |                |                    |
| D1.03   | Option #                                                                                     | Description                   |   |   | Cost per Unit   | Qty        |           |                    |                |                    |
| on Hand | Cost of Stock                                                                                |                               |   |   |                 |            |           |                    |                |                    |
| D2.01   | NNN                                                                                          | AAAAAAAAAAAAAAAAAAAAAAAAAAAAA |   |   | \$NNN,NNN,NN.NN | NNNNN AAAA |           | \$NN,NNN,NNN,NN.NN |                |                    |
| T1.01   | Total cost of Stock for Product                                                              |                               |   |   |                 |            |           |                    | AAAAA          | \$NN,NNN,NNN,NN.NN |
| TF.01   | Total Cost of Stock for Products                                                             |                               |   |   |                 |            |           |                    | AAAAA to AAAAA | \$NN,NNN,NNN,NN.NN |
| E1.01   | **** No option records found for this product. Product or Option file may be corrupted. **** |                               |   |   |                 |            |           |                    |                |                    |

You should be able to paint the report with very little assistance. However, some of the fields do require some explanation. The following paragraphs deal with these a line group at a time.

### Page Heading Group

The P1.01 and P1.02 lines contain three data fields. The first field at P1.01 column 69 is the report time. Create this as a standard alphanumeric field. When you enter the field details, specify the contents as \*TIME[1,5]. This is a substring

of the time that only prints the hours and minutes. The field at column 97 is \*PAGENO. The field on the next line at column 69 is \*DATE.

### Primary Detail Group

The D1.01 to D1.03 lines are printed for each product record found by the report. The D1.01 and D1.03 lines are individual headings for the product details and the option details.

The D1.02 line contains all four fields from the product record. These are *product\_no* at column 2, *description* at column 17, *supplier\_no* at column 59, and *lead\_time* at column 75. Create them using the appropriate field specifications.

### Secondary Detail Group

The D2.01 line is printed for each option record that is found by the linkage to the option file. The first four fields are from the option record. These are *option\_no*, *description*, *cost*, and *qty\_on\_hand*. Create these using the appropriate field specifications.

You must modify the cost field image to include numeric editing codes for numeric punctuation and the currency symbol. The numeric editing codes control the format of the field data on the report. Position the cursor on the cost field image and press the **Modify Field** function key. Enter \$NNN,NNN,NNN.NN. This overwrites the field specification definition to one that includes the currency symbol and numeric punctuation. It means that data in the cost field image is printed with a leading currency symbol and commas in the correct locations.

The last field on the line is the cost of stock. Create this field image manually. The source for the field is a *calculated item* that is totalled into \*TOTALS(1). Enter U-option\_cost in the *Field Name/Literal* field and then 1 in the *Total Number* field.

Although you haven't created this item, you can include it in this report. You will create the calculated item shortly.

### Subtotal Group

The T1.01 line is printed whenever the value of the first sort field changes. In this case it is the product number field in the product file. The first data field is the product number read from the product file. Create it using the *product\_no* field specification name. The field is created automatically as required.

The next few paragraphs provide an outline description of the subtotal lines, sub-heading lines and the data that they use when they are printed.

A control break occurs when the value of the sort field on the record read from the primary record changes. Therefore, a control break occurs when the next record is

read from the file. This may lead you to think that the new details would be used in the subtotal line group. They are not. HP ALLBASE/4GL uses the previous record details for the subtotal line group and then uses the new record details for any subheading line groups. Therefore the subtotal line prints the product number for the product just listed.

The second data field on this line contains the total cost of the options for the product just listed. The field image is identical to the field above it. Enter **\*TOTALS(1)** in the *Field Name/Literal* field as the name of the data source. If you have used the copy function key to copy the image from the field above it, check to make sure that the other detail fields are satisfactory. In particular, make sure that the automatic totalling field, *Total Number*, is blank.

You may be thinking that using **\*TOTALS(1)** is correct for the first product listed, but from then on it will give a running total for all options listed. This doesn't occur. HP ALLBASE/4GL keeps a matrix of all 16 **\*TOTALS** fields. The number of rows in this matrix is always one greater than the number of sort fields you have specified. There is a separate level of **\*TOTALS** for each control break level, and one for the final value.

In this case **\*TOTALS(1)** for the level one control break is being accrued for each option listed. When the level one control break occurs, the value of level one control break **\*TOTALS(1)** is added to **\*TOTALS(1)** for the final total level. The level one control break **\*TOTALS(1)** is then reset to zero. The T1 line group refers to the values of the level one control break **\*TOTALS** fields.

The *HPALLBASE/4GL Developer Reference Manual* describes the report totalling facilities for reports that use more than one control break.

### Final Total Group

The TF.01 line is printed at the end of the report. It contains the total value of all the stock included in the report. It also lists the range of products that have been included in the report.

The first two fields at column 65 and 75 are almost identical. Create their images by using the *product\_no* field specification. Modify their respective field details to change the name of their data sources.

The first field refers to **\*S01** and the second field refers to **\*S02**. You can include the screen fields on the current screen in a report.

The third field at column 83 is identical to the field above it on line T1.01. Make a direct copy of it to speed up the painting procedure. Remember that this time the value of **\*TOTALS(1)** is the final total value accrued over the whole report.

### Extra Line Group

The E1.01 line is only printed by the *print\_option* function when no options have been found for a particular product. It only contains one literal field.

Once you have painted each of these lines, exit from the report painter. The report painter automatically saves the image of the report. Before you generate the report, you must define the calculated item used in line D2.01. This is described in the next lesson.

---

## Summary

In this lesson you created a report that accessed two different data files. In creating this report, you used the following report facilities.

- A start of report function.
- Sort field definitions.
- Record selection criteria.
- File linkages and a post link read function.
- A before print function associated with a report line.

### Start of Report Functions

A start of report function allows you to display a screen to obtain values for reporting from the user. The start of report function is the only report function that can display a screen.

### Sort Field Definitions

The sort field definitions for a report define the report control breaks. A control break occurs whenever the value in a sort field changes from one record to the next.

At a control break, HP ALLBASE/4GL prints subtotal and subheading line groups. These line groups can contain subtotal values that have been accumulated in one of the \*TOTALS(n) communication area fields.

You can define sorting fields even if the primary report file is in the correct order. HP ALLBASE/4GL still uses the sort fields to define the control breaks for the report.

### Record Selection Criteria

Report record selection criteria allow you to select records that are printed on the report. The selection criteria specify a range of values for a field on the record read from the report primary file or a link file.

If the value in the specified field does not fall within the specified range, the record is not included in the report.

### File Linkages

File linkages allow you to retrieve data from a file other than the primary report file. File linkages are defined on a line group basis.

Before HP ALLBASE/4GL prints the line group, it performs the initial read of the link files defined for the line group. The data retrieved by the link read can be included in the initiating line group. The data can also be included in link print line groups.

You can specify that a function is executed after the link file read.

### Print Line Functions

HP ALLBASE/4GL allows you to execute a function before and after each physical report line is printed. In this report, you used a before print function to substitute a different print line if an option file read fails to find any options for a product.

## Calculated Items

This lesson also introduced calculated items. You will complete the definition of the *option-cost* calculated item in the next lesson.

---

---

## Lesson 26 – Calculated Items

---

---

### Objectives

In this lesson you will define and generate a calculated item. This calculated item completes the requirements of the report you defined in the previous lesson.

---

### Calculated Items

A calculated item is a special type of variable. A calculated item implicitly refers to a function. When HP ALLBASE/4GL encounters a reference to a calculated item, it automatically executes the function to determine the value of the calculated item.

A calculated item can be defined as a single CALC command, or it may be extended to use as many lines of logic in the function as you require.

### Menu Path

Dictionary, Storage Items, Calculated Items.

### Description

This screen allows you to define a calculated item. You will define the item's basic characteristics and a single CALC statement to derive the value of the calculated item. Defining a calculated item automatically creates a function of the same name. This function is automatically executed each time the item is accessed.

If the item you are defining requires more than a single CALC statement, you can subsequently call up the function in the function details screen. A calculated item defined this way is known as an extended calculated item.

| Developer                | Calculated Items                                                                               | calculated_items |
|--------------------------|------------------------------------------------------------------------------------------------|------------------|
| Item Name                | option_cost                                                                                    |                  |
| Length                   | 12                                                                                             |                  |
| Edit Code                | N (X/A/U/K/N/S/Q/D/T)                                                                          |                  |
| Number of Decimal Places | 2                                                                                              |                  |
| Pad Character            |                                                                                                |                  |
| CALC                     | F=cost.option x F=qty_on_hand.option = U=option_cost                                           |                  |
| Description              | option_cost<br>AUTHOR: rmjones<br>This calculation determines the cost of stock for one option |                  |
| Last modification:       | Date                                                                                           | Time             |
| Variable                 | Function                                                                                       | Numeric          |
| Details                  | Constant                                                                                       | Constant         |
| AlphaNum                 | 21                                                                                             | 46               |
| System                   | Keys                                                                                           | Commit           |
| Help                     | Data                                                                                           | Previous         |
|                          |                                                                                                | Menu             |

## Entering the Field Values

**Item Name.** Enter `option_cost`

This is the name of the calculated item that you are defining. HP ALLBASE/4GL automatically creates a function with the same name for the necessary logic that is needed behind the item. Even if the calculated item only requires one line of logic to calculate its value, HP ALLBASE/4GL still creates the function.

A calculated item cannot have the same name as an existing function in the same application.

**Length.** Enter `12`

The length of the calculated item can be up to 99 characters.

**Edit Code.** Enter `N`

This is a standard edit code.

**Number of Decimal Places.** Enter `2`

This specifies the maximum number of decimal places assigned to the item. It is only relevant for numeric fields.

**Pad Character.** Leave blank.

As for variables, alphanumeric calculated items are left justified and numeric items are right justified. The pad character entered here replaces any leading or trailing spaces remaining as a result of the justification.

### The CALC command

When you leave the pad character field you will see that the word *CALC* is displayed to the left of the input area occupied by the cursor. This input area allows you to enter the statement to calculate the value of the item. Refer to the *HP ALLBASE/4GL Developer Reference Manual* for further details about the CALC command. Enter:

```
F-cost.option x F-qty-on-hand.option = U-option-cost
```

Note that the destination of the CALC command is the calculated item itself. This is how a value is assigned to a calculated item.

---

#### Note

If you extend a calculated item, you can use any logic block commands including a visit to another function. However, the only function that can assign a value to a calculated item is the function with the same name as the calculated item. If you attempt to assign a value to a calculated item from any other source, the application will abort at run-time.

---

If you don't want to use a CALC statement, or wish to extend this function you can leave the CALC statement blank. You can then define the logic for the item by calling up the function with the function details screen.

**Description.** Enter a suitable description.

Press the **Commit Data** function key when you have finished creating the calculated item.

A calculated item automatically creates a function of the same name. Pressing the **Commit Data** function key generates the function automatically. If any errors occur during the generation operation, they are displayed in the normal way. Correct any errors using either the calculated item screen or the function details screen.

You may wish to examine the logic block created by the CALC item by looking at it using the function details screen.



## Generating the Report

Once you have defined the calculated item, generate the *product* report. The *training* application is now functionally complete except for some help screens and the *supplier* module.

---

## Summary

In this lesson you created a calculated item. Calculated items have the following properties.

- A calculated item is similar to a variable, except that its value is determined by executing a function every time the calculated item is referenced.
- Calculated items can be simple calculated items, or extended calculated items. In the simple form, the value of the item is determined by a single CALC command. In the extended form, the value of the calculated item is determined by a function containing more than one command, or containing a command other than a CALC command.
- Creating a calculated item automatically creates a function with the same name. HP ALLBASE/4GL generates the function automatically when you commit the calculated item.
- The value of a calculated item can only be assigned by the function with the same name as the calculated item. If you attempt to move a value into a calculated item from any other source, the application will abort.

## Lesson 27 – User Help Screens

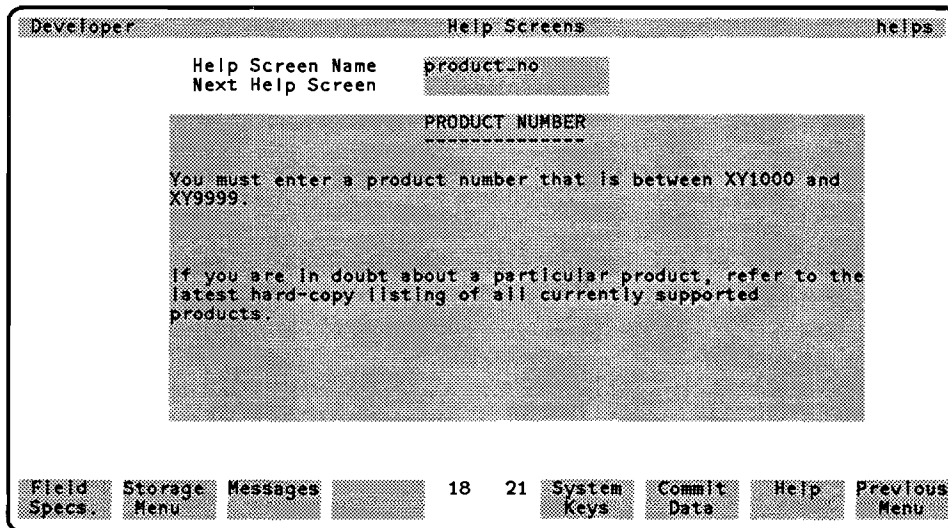
### Objectives

In this lesson, you will create a number of end user help screens.

### Help Screens

HP ALLBASE/4GL allows you to define three levels of help. These are *message help*, *field help*, and *screen help*.

The procedures for defining the different types of help screens are identical. The way these screens are linked to the application determines the way HP ALLBASE/4GL displays them when the user presses the **Help** function key.



### Menu Path

Dictionary, **Help Screens**.

## Description

A help screen is a free format area of 16 lines that can each contain 60 characters. When displayed, it is centered on the screen.

## Entering the Field Values

**Help Screen Name.** Enter `product_no`

This is the name of the help screen. In this case, the help screen is associated with the `product_no` field.

**Next Help Screen.** Leave blank.

You are not limited to just one help screen. Entering the name of a further help screen in this field links a further help screen to this screen. When the user presses the **More Help** function key on this help screen, the next help screen named here is displayed. This example only uses one screen.

**Help Screen Contents.** Enter some text for the help screen.

This is a free form area where you can insert lines, delete lines, insert characters and delete characters as required. You can use any printable characters.

Press the **Commit Data** function key when you have finished creating the help screen.

When you created the `product` data screen using `product_no` field specification, you accepted the default help name of `product_no` for the field. Next time you run the application and the cursor is on the product number field, press the **Help** function key. The help screen you have just defined will be displayed. This is an example of field level help.

## Message Level Help

Modify the `product_no_error` message so it specifies a help screen name of `product_no_error`. This message is invoked when the `product_no` range check fails.

Create a help screen called `product_no_error`, to tell the user which values are valid for the field.

Run the application again and enter an invalid product number in the `product_no` field on the `product` screen. The error message is displayed and the field is error highlighted. Press the **Help** function key. The help message for the error is displayed.

This is an example of a message level help. The same process applies for ABORT and QUERY type messages.

## Screen Level Help

Now create a help screen called *product* to give the user general assistance in using the *product* screen. Go to the *product* screen header and ensure that the help screen name is *product*.

Run the application again and press the **Screen Help** function key after you have called the field help from the product number field. Screen help will then be displayed.

If you want to allow the user to call up screen help directly from a field, you must ensure that its field help name is blank.

---

## Summary

HP ALLBASE/4GL allows you to define help screens associated with messages, screen fields, and screens. If you need to display more information than you can fit on one screen, you can link further help screens to any help screen.

---

---

## Lesson 28 – The Module Builder

---

---

### Objectives

In this lesson, the final lesson of chapter 10, you are introduced to the features of the module builder. While working through this lesson you will create the *supplier* module. This completes the *training* application.

First you will create the dictionary items required for the module, then you will create the module using the module builder screen and the module details screen, and finally you will generate the complete module and attach it to the *main* menu.

This lesson shows you what control you have over the module that HP ALLBASE/4GL creates.

---

### Introduction to Module Builder

HP ALLBASE/4GL module builder allows you to quickly create a working application module. This application, or module, is made up of several application items, Application items are individual components of the application or module, such as screens, logic blocks, field specifications, messages, and other named components. All the module's application items revolve around a main file and a single screen. Once you have specified the main file, the associated record layout, and the module type (inquiry or maintenance), the module builder automatically creates the resultant screen and logic required to perform maintenance or inquiry tasks on the main file.

When using the module builder you do not need to create the items individually. Consequently you do not need to paint the data screen and complete the field details screen, nor do you need to create each required logic block, message or any other application item. All this is performed by the module builder.

Another feature of the module builder allows you to link data entered into a field in the resultant screen with existing data in another, or secondary, file. You can use the link feature to validate the entered data with data in the secondary file or to display data from a single record in the secondary file on the resultant screen.

---

## Necessary Preparations

Before you can use the module builder you must first define and create the *supplier\_name* field specification and the *supplier* record layout. Then you need to create the *supplier* file.

The *supplier* file contains the following fields:

- **supplier\_no**
- **supplier\_name**
- **address**
- **state\_code**
- **zip\_code**
- **product\_no**

### Field Specification

You previously specified all but one of the fields required for this lesson. The one dictionary field specification you need to create is *supplier\_name*. Only the field entries that differ from the defaults are listed. Create the *supplier\_name* field specification now.

**Supplier Name.** A field for the supplier company name.

|                  |                      |
|------------------|----------------------|
| Field Spec. Name | <b>supplier_name</b> |
| Field Length     | <b>30</b>            |
| Edit Code        | <b>X</b>             |

The module builder uses the text in the short description field, the first line in the documentation area of the field specification screen, as the prompt for the field in the resultant screen. Enter a suitable title in the short description field.

### Record Layout

One record layout is required, named *supplier*. Remember to generate the record layout once you have defined it. Create the record layout as specified below:

**Record Layout Name. supplier**

| Field | Field Spec. Name | Key # | Duplicates |
|-------|------------------|-------|------------|
| 1     | supplier_no      | 1     | N          |
| 2     | supplier_name    | 2     | Y          |
| 3     | address          |       |            |
| 4     | state_code       | 3     | Y          |
| 5     | zip_code         | 4     | Y          |
| 6     | product          | 5     | Y          |

**File Specification**

Now that you have generated the record *supplier* layout, you are ready to create the *supplier* file. Complete the file definition, and then use the file creation screen to create the physical *supplier* file.

**Supplier File.** This file contains all the supplier records.

|               |                 |
|---------------|-----------------|
| File Name     | <b>supplier</b> |
| File Type     | <b>I</b>        |
| External Name | <b>supplr</b>   |
| Record Layout | <b>supplier</b> |

---

## Module Builder

You are now ready to use the module builder to build the resultant screen and associated logic needed to allow users access the *supplier* file. To access the module builder screen, choose *Module Builder* from the main menu. HP ALLBASE/4GL displays the module builder screen shown below.

| Developer          |              | Module Builder |              | module_builder |                                            |
|--------------------|--------------|----------------|--------------|----------------|--------------------------------------------|
| Module Name        | supplier     | Type           | M (I/M)      |                |                                            |
| Main Access: File  | supplier     | Record         | supplier     | Index          | supplier.no                                |
| Include All Fields | Y (Y/N)      |                |              |                |                                            |
| Dict. Menu         | Screens Menu | Logic Menu     | Reports Menu | 3 21           | Help Commit Data System Keys Previous Menu |

## Description

You use the module builder screen to specify the essential information required by the module builder to create the *supplier* module. This screen requires you to specify the main file name, along with a record layout name and a index field name. The module uses the specified record layout and index field to access the main file.

## Entering the Field Values

**Module Name.** Enter *supplier*

This name is used by the module builder in the names it gives most application items it creates for the module. The naming conventions used by the module builder are described in the *HP ALLBASE/4GL Developer Reference Manual*.

**Type.** Accept the default value of **M**

There are two possible module types: maintenance and inquiry. A maintenance type module allows the end user to view and modify the information in the file. An inquiry type module only allows the end user to view the file.

**Main Access: File.** Enter the file name *supplier*



This field specifies the main file to be accessed by the module for viewing or maintenance.

**Record.** Accept the default value of `supplier`

This is the record layout used to access the file. HP ALLBASE/4GL automatically enters the default record layout.

**Index.** Accept the default value of `supplier_no`

This is the name of the field used to locate an individual record within a file. This value defaults to the primary index for the file.

**Include All Fields.** Accept the default value of `Y`

This tells the module builder to include in the resultant screen all fields listed in the record layout. If you enter `N`, then you must use the module details screen to define which fields from the record layout you want in the resultant screen. The module builder always includes the index field in the resultant screen.

Press the `Commit Data` function key when you have completed the module builder screen. HP ALLBASE/4GL displays a message asking you either to move to the module details screen or to generate the module.

---

### Note

At this point you could generate the module, provided you specified `Y` at the *Include All Fields* prompt. Module builder would attempt to place all the fields in the record layout on the resultant screen, and there would be no links to secondary files. However, in this lesson you will access the module details screen to further refine the module you are building.

---

Press the `Specify Details` function key to display the module details screen.

---

## Module Details

You use the module details screen to specify which fields from the record layout are included in the resultant screen. You can also specify which fields in the resultant screen are linked with secondary files. Access to the module details screen, which is shown below, is only possible after you have committed the module builder screen.

| Developer                    | Module Builder  | module-details  |                 |
|------------------------------|-----------------|-----------------|-----------------|
| Module Name                  | supplier        | Type            | M (I/M)         |
| Main Access:                 | File supplier   | Record supplier | Index           |
| Include All Fields           | (Y/N)           |                 |                 |
| --- Screen Field Details --- |                 |                 |                 |
| Sequence Number              | 1               | Action          | C (A/C/D/I/L)   |
| File Spec. Name              | supplier        |                 | ( )             |
| On Screen Label              | Supplier Number |                 |                 |
| Type                         | I (I/D)         |                 |                 |
| Required                     | Y (Y/N)         |                 |                 |
| Link                         | N (Y/N)         |                 |                 |
| Validate                     | (Y/N)           |                 |                 |
| Link To:                     | File            | Record          | Index           |
| List Record                  | Previous Field  | Next Field      | Generate Module |
| 11                           | 22              | Help            | Commit Data     |
|                              |                 | System Keys     | Previous Menu   |

## Description

To create a screen, the module builder takes the fields from the record layout and creates corresponding screen fields. With the module details screen, you move through each field in the screen field sequence, defining its field specification name and its on screen label. You can change, delete, insert, and add fields to the resultant screen. You can also define whether a field is for display or input, and whether it is a field that requires an entry.

You also use the module details screen to link a field in the resultant screen with a secondary file. To build this feature into your application you first specify which file the field is linked to. Then if necessary, you specify the record layout and index field for this file. In the resultant module, there is an after function associated with the link field. This after function looks through the index field of the secondary file for a record with data matching that in the link field. This is the matching record.

You can then use this match to perform one or both of two options. The first option is to display data from other fields of the matching record on the resultant screen. The second option is to validate the entered data against data in the index field of the secondary file. This acts in the same way as a validation range or validation table.

You can only create one-to-one links between a screen field and a secondary file. This means you can link each secondary file to only one screen field, and that you

can only access one record at a time in the secondary file. However you can link up to twenty fields with secondary files.

## Entering the Field Values

When you commit the *Sequence Number* field, HP ALLBASE/4GL displays the details for a screen field.

### Supplier Number

**Sequence Number.** Accept the default value of 1

**Action.** Accept the default value C

This action code works in the same manner you encountered when creating logic blocks.

**On Screen Label.** Enter *Supplier Number*

This is the label text displayed next to the *supplier\_no* field on the resultant screen.

The module builder always includes the index field you specified as the first field in the resultant screen. As this field is used for record retrieval, its function within the resultant screen cannot be modified. Consequently you can only edit the *On Screen Label* field.

Press the **Commit Data** function key.

### Supplier Name

**Sequence Number.** Accept the default value 2

**Action.** Accept the default value C

**File.** Accept the default value *supplier*

The screen field's primary movement field is in the file buffer specified in this field.

**Field Spec. Name.** Accept the default value *supplier\_name*

The data in this field completes specification of the primary movement field for the screen field.

**On Screen Label.** Enter *Supplier Name*

Press the **Commit Data** function key complete the specification of details for the *supplier\_name* screen field.

Notice that after you commit the *Sequence Number* field HP ALLBASE/4GL fills the *File*, *Field Spec. Name*, and *On Screen Label* fields with default values. This

screen sequence is defined by the record layout *supplier*. HP ALLBASE/4GL uses the record layout for the screen field sequence if you specify **Y** at the *Include All Fields* prompt in the module builder screen.

### Other Field Labels

Use the same procedure to enter the details of the screen fields listed below. The occurrence field is unlabeled and enclosed in parenthesis. It is on the same line as the *Field Spec. Name* field. The cursor only moves to the occurrence field when necessary.

| Sequence No. | Field Spec. Name | Occurrence | On Screen Label   |
|--------------|------------------|------------|-------------------|
| 3            | address          | 1          | <b>Address</b>    |
| 4            | address          | 2          |                   |
| 5            | address          | 3          |                   |
| 6            | state_code       |            | <b>State Code</b> |
| 7            | zip_code         |            | <b>Zip Code</b>   |

Note that the address field from the record layout requires three fields on the generated screen, but only the first field is labeled.

### Linking a Field to a Secondary File

Now link the *product\_no* field to a secondary file.

**Sequence Number.** Accept the default value of **8**

Accept the default values for the *File* and *Field Spec. Name* fields.

**On Screen Label.** Enter **Product Number**

**Link.** Enter **Y**

This tells the module builder to create an after function to link the data entered by the user into the *Product Number* field with data in the secondary file named below.

**Validate.** Enter **Y**

This tells the module builder to include in the after function, logic to validate the data entered into the screen field with the index field of the secondary file. It means that the user cannot enter into the *supplier* file a product number that doesn't exist in the *product* file.

**Link To: File.** Enter **product**

This is the name of the secondary file linked to the *product\_no* field.

**Link To: Record.** Accept the default value **product**

**Link To: Index.** Accept the default value **product\_no**

The link after function searches through the index field of the secondary file. It retrieves the record where data in the index field matches that entered in the link field.

Press the **Commit Data** function key.

## Adding a Display Field

You will now add a display field to the resultant screen. This field shows the *description* field from a *product* file record where the data in the *product\_no* field matches the data entered in the link field.

**Sequence Number.** Enter **9**

Nine is the number of the next field in the resultant screen field sequence.

**File.** Accept the default value of **product**

**Field Spec. Name.** Accept the default value of **description**

**On Screen Label.** Enter **Product Description**

Press the **Commit Data** function key.

---

## Final Steps

You are now ready to perform the final steps required to create the *supplier* module; generating the module and joining it to the *main* menu. Then you can run the *training* application to investigate the capabilities of the *supplier* module.

## Module Generation

Press the **Generate Module** function key to create the *supplier* module.

HP ALLBASE/4GL now works through the process required to build and generate a working module. When complete HP ALLBASE/4GL displays the message “*Generation Successful*” and the displays the module builder screen.

### The Module Building Process

The process HP ALLBASE/4GL follows to build the *supplier* module can be summarized into three main steps:

1. Creating the resultant screen and the process that calls it.
2. Creating all other application items.
3. Generating all the module's items.

For more information refer to the *HP ALLBASE/4GL Developer Reference Manual*.

### Joining the Module to the Main Menu

Finally you add an action item to the *main* menu which accesses the *supplier* module. Do this now. Use the screen painter to add an action item to the *main* menu which calls the process *mb\_supplier*. Label the action item *Supplier Details*.

You can now test the *supplier* module with *Application Testing*. Explore the module fully – add records, retrieve records, and delete records. The help screens contain detailed instructions on using the module.

---

## Further Options

Once the *supplier* module has been created, you have several further options. You can modify the items created to alter the module's functionality, or you can use items created for the module in other parts of an application.

You treat all the items created by the module builder in the same way as those you create yourself. All module builder items are listed in the catalog display. Modification is quite simple because the templates the module builder uses to create items contain the same HP ALLBASE/4GL features available to you.

Copying items created by module builder to other parts of your application speeds up the creation of an application. An example of where you can use items elsewhere is the after function for the *Product Number* field in the *mb\_supplier* screen, called *mb\_supplier-1-08*. In the *mb\_supplier* screen this after function validates the product number entered against the secondary file *product*. Thus it ensures the user only enters a valid product number into the *mb\_supplier* screen. You can copy and then modify this function to become an after function to validate the *Supplier Number* field in the *product* screen. In this context, the after function would validate data entered into the *Supplier Number* field against the secondary file *supplier*.

---

## Summary

In this lesson you created the *supplier* module using the HP ALLBASE/4GL module builder.

### Necessary Preparations

Before you used the module builder, you had to define and create the field specification names, record layout and the physical file required for the module you created.

### Module Building

To create the module you accessed the module builder screen to define the following:

- The module name, which is used in the naming of application items created by the module builder.
- The module type. You can create maintenance modules, which allow the end user to view, modify, delete, or add information in a file, or inquiry modules, which only allow the user to view information in a file.
- The main file accessed by the module. The screen and associated logic created can only modify one file; the main file. When you specify the main file you must also specify the record layout and index field you are using to access it.
- Whether the module builder automatically places all the fields in the record layout in the resultant screen.

Once you have committed the module builder screen, you can either generate the module or move to the module details screen to refine the module. If you want to generate the module from the module builder screen you must specify *Y* at the *Include All Fields* prompt.

In the module details screen you performed the following:

- Define which fields from the main file record layout are placed in the resultant screen.
- Define a link between a field in the resultant screen and a secondary file.
- Use this link to display information from a record in the secondary file on the resultant screen.
- Use this link to validate the data entered into the link field.

When this screen was complete, you generated the module.

### Joining the Module to the Application

After the module had been generated you joined it to the rest of your application by adding an action item to the *main* menu. This action item calls the main process generated by the module builder.

---

## Where You Are Now

Congratulations! You have completed the KSAM section of the HP ALLBASE/4GL Developer Self-Paced Training Guide.

While you have been developing the *training* application in this guide you have seen how to define the majority of items in HP ALLBASE/4GL. You have also seen how HP ALLBASE/4GL runs applications, and how HP ALLBASE/4GL can do a lot of work for you during the application development phase.

Obviously there are many more ways to use HP ALLBASE/4GL than those outlined in this training guide. You have probably started referring to the *HP ALLBASE/4GL Developer Reference Manual*. By now you should be familiar with its layout and content. We suggest that you now start experimenting and solving real life problems that are specific to your data processing needs.

### From Here On

This guide contains two further chapters. These chapters describe the interface between HP ALLBASE/4GL and HP ALLBASE/SQL.



# Contents

---

## Chapter 11: Using HP ALLBASE/SQL with HP ALLBASE/4GL

|                                    |        |
|------------------------------------|--------|
| Introduction . . . . .             | 11 - 1 |
| What Happens Next . . . . .        | 11 - 1 |
| What We Assume . . . . .           | 11 - 1 |
| HP ALLBASE/SQL Interface . . . . . | 11 - 2 |
| Setting Up . . . . .               | 11 - 2 |
| The Application . . . . .          | 11 - 3 |



### Lesson 29 – Creating an HP ALLBASE/SQL Table

|                                                        |        |
|--------------------------------------------------------|--------|
| Objectives . . . . .                                   | 11 - 4 |
| Some Preparations . . . . .                            | 11 - 4 |
| Field Specifications . . . . .                         | 11 - 4 |
| Record Layouts . . . . .                               | 11 - 5 |
| HP ALLBASE/SQL Table Definition and Creation . . . . . | 11 - 6 |
| HP ALLBASE/SQL Table Ownership . . . . .               | 11 - 8 |
| Table Format . . . . .                                 | 11 - 9 |
| Summary . . . . .                                      | 11 - 9 |

### Lesson 30 – SQL Logic Blocks

|                                         |         |
|-----------------------------------------|---------|
| Objectives . . . . .                    | 11 - 10 |
| Some Preparations . . . . .             | 11 - 10 |
| Working With SQL Data . . . . .         | 11 - 11 |
| The Customer Process . . . . .          | 11 - 12 |
| The Customer Function . . . . .         | 11 - 14 |
| SQL Logic Blocks . . . . .              | 11 - 16 |
| Defining an SQL Logic Block . . . . .   | 11 - 16 |
| The Customer SQL Logic Block . . . . .  | 11 - 16 |
| SQL Logic Block Details . . . . .       | 11 - 16 |
| Generating an SQL Logic Block . . . . . | 11 - 17 |
| Generation Errors . . . . .             | 11 - 18 |
| SQL Logic Block Limitations . . . . .   | 11 - 18 |
| The SELECT Command . . . . .            | 11 - 19 |
| Host Variables . . . . .                | 11 - 19 |

|                                      |         |
|--------------------------------------|---------|
| Two More SQL Logic Blocks . . . . .  | 11 - 20 |
| SQL Application Generation . . . . . | 11 - 21 |
| SQL Data Manipulation . . . . .      | 11 - 21 |
| Retrieving SQL Data . . . . .        | 11 - 22 |
| Adding New Records . . . . .         | 11 - 23 |
| Modifying a Record . . . . .         | 11 - 24 |
| Deleting Records . . . . .           | 11 - 25 |
| Committing Transactions . . . . .    | 11 - 26 |
| Summary . . . . .                    | 11 - 27 |
| SQL Logic Blocks . . . . .           | 11 - 27 |
| The SELECT Command . . . . .         | 11 - 27 |
| SQL Data Manipulation . . . . .      | 11 - 28 |

**Lesson 31 – Reporting From an HP ALLBASE/SQL Database**

|                                    |         |
|------------------------------------|---------|
| Objectives . . . . .               | 11 - 30 |
| Preparations . . . . .             | 11 - 30 |
| Report Header . . . . .            | 11 - 30 |
| Start of Report Function . . . . . | 11 - 31 |
| Report SQL Logic Block . . . . .   | 11 - 31 |
| Completing the Report . . . . .    | 11 - 31 |
| Summary . . . . .                  | 11 - 32 |

# 11 Using HP ALLBASE/SQL with HP ALLBASE/4GL

---

## Introduction

This chapter and the following chapter are only relevant to you if you intend to use HP ALLBASE/SQL in conjunction with HP ALLBASE/4GL. If you don't intend to use HP ALLBASE/SQL with HP ALLBASE/4GL, you can relax—you have finished this training course.

## What Happens Next

This chapter and the next chapter assume that you have worked through the first 10 chapters of this guide. These next chapters concentrate on the additional features in HP ALLBASE/4GL that allow you to access and manipulate data stored in HP ALLBASE/SQL databases. They assume that you are familiar with the concepts and techniques described in the earlier chapters. These two chapters concentrate on the techniques for using the interface between HP ALLBASE/4GL and HP ALLBASE/SQL to achieve the same results.

As you work through the next lessons, you will use the existing *training* application as a basis for developing a new application called *sqltrain*. To avoid duplication of effort, you can copy a many items from the *training* application. Many of these can be used without any modification. Others require some changes before they can be used. You will also develop a number of new items for this application.

The application you develop in the following lessons performs the same data processing functions as the earlier *training* application. From the end-user's viewpoint, the two applications are equivalent.

## What We Assume

The descriptions in this chapter and the next chapter assume that you are familiar with the basic concepts of SQL. They also assume that you are familiar with the format and syntax of SQL commands. If you haven't used SQL before, we suggest that you read the introductory chapters of the *HP ALLBASE/SQL Database Administration Guide*, and the *HP ALLBASE/SQL Reference Manual* before you start developing this application.

## HP ALLBASE/SQL Interface

The interface between HP ALLBASE/4GL and HP ALLBASE/SQL consists of the following items:

- **Base Table Definitions**

You may define and create HP ALLBASE/SQL base tables from within HP ALLBASE/4GL. You will create a number of HP ALLBASE/SQL tables as you develop this application.

- **SQL Logic Blocks**

SQL logic blocks are the mechanism for passing SQL commands from HP ALLBASE/4GL to HP ALLBASE/SQL. An SQL logic block contains one or more SQL commands that are executed when the SQL logic block is invoked by a command (the SQL command) in a logic block. This application uses a number of SQL logic blocks to access and update SQL data.

- **Select Lists**

You can think of a select list as a “virtual” table. The columns in a select list can be derived directly from an existing table or view, or can be derived using SQL expressions. Select lists provide a means of accessing SQL data in existing tables or views, or combining data in a number of tables or views for a particular query. You will use two select lists in this application.

These items operate in conjunction with the facilities you have already used. The techniques for using screens, logic and reports in conjunction with data retrieved from an HP ALLBASE/SQL database are exactly the same as the techniques you have already used.

---

## Setting Up

To work through the following lessons, you must have a suitable application defined in the HP ALLBASE/4GL system, and you must have a suitable HP ALLBASE/SQL database environment defined.

Appendix A contains a description of the application definition. It also contains a suggested database environment configuration for this application. The instructions in the following lessons assume that the application definition and database environment are the same as those described in Appendix A.

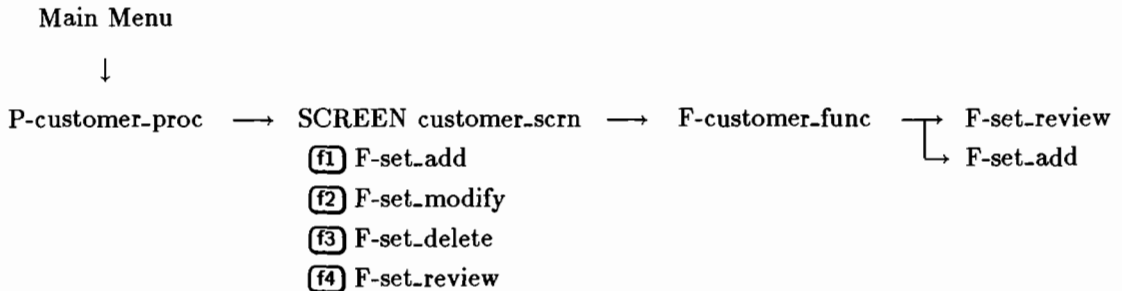
## The Application

The remainder of this chapter describes the procedures to create a simple application using an HP ALLBASE/SQL database. This application provides similar functionality to the *customer\_proc* process in the KSAM-based *training* application.

This chapter contains three lessons that show you how to create an HP ALLBASE/SQL table from HP ALLBASE/4GL, define the logic to access the table, and create a report that uses the table.

To the user, this application appears to be identical to the *customer\_scrn* screen in the earlier application. Structurally, the application is also very similar to the KSAM-based application.

The following diagram shows the structure of this application.



Once again, this application uses the technique of calling a process from a menu item. This process displays a screen, which in turn, executes a function that is associated with a field on the screen. The screen also uses a set of function keys. Each function key calls a function.

---

---

## Lesson 29 – Creating an HP ALLBASE/SQL Table

---

---

### Objectives

In this lesson you will:

- Define a record layout for an HP ALLBASE/SQL base table.
- Complete the definition for the base table.
- Create the table.

---

### Some Preparations

Most of the dictionary items for this application can be copied directly from the existing *training* application. You can also copy a number of items from the *training* application, and then modify them slightly to use them in this application.

To get started with the *sqltrain* application, sign on to the developer and use the utilities menu copying screen to copy the following dictionary items from the *training* application.

- **Field Specifications**
  - customer\_name
  - customer\_no
  - state\_code
  - zip\_code
- **Validation Table**
  - state\_code

---

### Field Specifications

Unlike the KSAM data manager, HP ALLBASE/SQL does not allow dictionary field specifications with multiple occurrences. In the KSAM-based *training* application, the *address* field specification is defined as occurring three times.

Instead of using one field with three occurrences, you must create three new field specifications. Create three field specifications *address*, *address1* and *address2*. Each field is 30 characters long, left justified, and has an X edit code.

## Record Layouts

HP ALLBASE/4GL uses record layouts to define the columns of HP ALLBASE/SQL tables. In general, record layouts for HP ALLBASE/SQL tables are similar to the record layouts for KSAM files, with the following exceptions:

- HP ALLBASE/SQL is **not** case sensitive. When you use a record layout to define an HP ALLBASE/SQL table, the field specification names are shifted to uppercase. This means that you must make sure that the record layout does not contain two field specifications that have the same name when they are shifted to uppercase. (For example, the field specification names *Account\_No* and *account\_no* are distinct in HP ALLBASE/4GL, but are duplicates in HP ALLBASE/SQL)
- HP ALLBASE/SQL table column names defined from HP ALLBASE/4GL can use the characters A to Z, 0 to 9, and - (underscore), but cannot use extended (eight-bit) characters.
- An HP ALLBASE/SQL table can only have one record layout.
- The record layout for an HP ALLBASE/SQL table does not need to have any fields defined as key fields. If you do define a field on the record layout as a key field, HP ALLBASE/4GL uses the field to create an index for the table.

The *sqltrain* application uses a record layout *customer\_rcrd* to define the columns for the *customer* table. The record layout is similar to the record layout for the KSAM *customer* file.

### Record Layout – *customer\_rcrd*

Use the record layout header screen to define a header for the *customer\_rcrd* record layout, and then use the record layout details screen to build the record. The structure of the record is:

| Field No. | Field Spec. Name | Key # | Duplicate Keys |
|-----------|------------------|-------|----------------|
| 1         | customer_no      | 1     | N              |
| 2         | customer_name    |       |                |
| 3         | address          |       |                |
| 4         | address1         |       |                |
| 5         | address2         |       |                |
| 6         | state_code       |       |                |
| 7         | zip_code         |       |                |

Generate the record layout after you have defined it.

## HP ALLBASE/SQL Table Definition and Creation

The procedure for defining and creating an HP ALLBASE/SQL table is similar to the procedure for creating KSAM files. First, you must complete the definition for the table, and then physically create the table.

### Menu Path

Dictionary, Database Items, **Data File/SQL Table Definition**

### Description

HP ALLBASE/4GL uses the same screen to define HP ALLBASE/SQL tables and KSAM data files. The screen uses two different windows to allow you to specify the details for SQL tables, or KSAM files.

For an SQL table, this screen allows you to specify the name of the table as it is known to HP ALLBASE/4GL. This screen also allows you to specify the SQL DBEFileSet to hold the table, the SQL access class for the table, and the name of the record layout for the table.

| Developer          |                                                                             | File/SQL Table Definition |             | file_defn |    |             |             |      |               |
|--------------------|-----------------------------------------------------------------------------|---------------------------|-------------|-----------|----|-------------|-------------|------|---------------|
| File Name          | customer                                                                    | File Type                 | S (I/S/F/V) |           |    |             |             |      |               |
| Description        | customer                                                                    |                           |             |           |    |             |             |      |               |
| AUTHOR:            | rmjones                                                                     |                           |             |           |    |             |             |      |               |
|                    | This base table contains customer records for the SQL training application. |                           |             |           |    |             |             |      |               |
| Last Modification: | Date                                                                        | Time                      |             |           |    |             |             |      |               |
| External Name      | CUSTOMER                                                                    |                           |             |           |    |             |             |      |               |
| SQL DBEfileset     | CUSTFS                                                                      |                           |             |           |    |             |             |      |               |
| SQL Access Class   | 1 (1 - Public, 2 - Publicread, 3 - Private)                                 |                           |             |           |    |             |             |      |               |
| Record Layout      | customer_rcrd                                                               |                           |             |           |    |             |             |      |               |
| Field Specs        | Record Header                                                               | SQL Sel Details           | Create File | 3         | 31 | System Keys | Commit Data | Help | Previous Menu |



## Entering the Field Values

**FileName.** Enter `customer`

This is the HP ALLBASE/4GL name for the SQL table. This is the name you use in references to the table in HP ALLBASE/4GL logic.

**File Type.** Enter `S`

This entry indicates the type of file interface that is to be used in this application. `S` specifies that this is an HP ALLBASE/SQL table. When you enter `S` in this field, HP ALLBASE/4GL automatically displays the SQL table definition window.

**Description.** Enter some text that describes the purpose of the SQL table.

**External Name.** Accept the default value of `CUSTOMER`

This is the name of the table as it is known in the database environment for the application.

**SQL DBEfileset.** Enter `CUSTFS`

This is the name of the DBEfileset for the table. The DBEfileset must exist in the database environment before you can create the file. The entry in this field defaults to `SYSTEM`, but you can type over the default.

**SQL Access Class.** Accept the default entry value of `1`

This entry specifies the table locking mode as *public*, *publicread*, or *private*. All transactions on this table are subject to the normal HP ALLBASE/SQL locking provision. Refer to the *HP ALLBASE/SQL Reference Manual* for more information about table locking.

**Record Layout.** Enter `customer_rcrd`

Press the **Commit Data** function key to create the definition of the SQL table when you have entered all of the values required by this screen.

Once you have completed the SQL table definitions, press the **Create File** function key to access the file/SQL table creation screen. This is the same screen that you used to create KSAM files.

| Developer                                                              |               | File/SQL Table Creation |        | create_file    |               |
|------------------------------------------------------------------------|---------------|-------------------------|--------|----------------|---------------|
| File Name                                                              | customer      | File Type               | S      | External Name  | CUSTOMER      |
| SQL Access Class                                                       | 1             | SQL DBEfilesset         | CUSTFS | Key Field List |               |
| Key                                                                    | Dup           | Key                     | Dup    | Key            | Dup           |
| 1. customer.no                                                         | N             | 8.                      |        | 15.            |               |
| 2.                                                                     |               | 9.                      |        | 16.            |               |
| 3.                                                                     |               | 10.                     |        | 17.            |               |
| 4.                                                                     |               | 11.                     |        | 18.            |               |
| 5.                                                                     |               | 12.                     |        | 19.            |               |
| 6.                                                                     |               | 13.                     |        |                |               |
| 7.                                                                     |               | 14.                     |        |                |               |
| Description                                                            |               |                         |        |                |               |
| customer                                                               |               |                         |        |                |               |
| AUTHOR: rmjones                                                        |               |                         |        |                |               |
| This base table contains customer records for the SQL                  |               |                         |        |                |               |
| training application.                                                  |               |                         |        |                |               |
| Last Modification:                      Date                      Time |               |                         |        |                |               |
| File Defn.                                                             | Reformat File | Delete File             | 3 34   | System Keys    | Commit Data   |
|                                                                        |               |                         |        | Help           | Previous Menu |

## Entering the Field Values

Accept the default entry of *customer* for the file name. This is the only input field on the screen. When you press **(Return)**, HP ALLBASE/4GL displays the details you have defined for the table.

Press the **Commit Data** function key to commit the screen and create the table. HP ALLBASE/4GL displays a message telling you that it is connecting to the application database, and then displays a message to confirm that the table has been created successfully.

## HP ALLBASE/SQL Table Ownership

HP ALLBASE/4GL creates tables as *owner.group.table* where *owner.group* is the SQL owner group defined for the application. The name of this owner group is specified on the application definition screen in the administrator.

The *customer* table is created as *SQLGRP.CUSTOMER* in the application database environment.

## Table Format

HP ALLBASE/4GL creates the table using the field specifications on the table record layout to define the columns of the table. HP ALLBASE/4GL creates columns defined by numeric field specifications as decimal columns, and columns defined by other field specification types as CHAR columns. All columns created permit null values.

HP ALLBASE/4GL uses the key fields on the record layout for a table to create indexes for the table. The index corresponding to key field number 1 is a clustering index.

---

## Summary

In this lesson you created a record layout for an HP ALLBASE/SQL table, created the definition for a table, and then created a base table in the application database environment.

HP ALLBASE/SQL tables have the following characteristics:

- An HP ALLBASE/SQL table can only have one record layout.
- The record layout for an HP ALLBASE/SQL table cannot contain field specifications defined as having multiple occurrences.
- The record layout for an HP ALLBASE/SQL table does not need to contain any key fields. If it does, HP ALLBASE/4GL uses the key fields to create indexes for a table.
- The access and locking provisions for HP ALLBASE/SQL tables created from within HP ALLBASE/4GL are controlled by the normal HP ALLBASE/SQL access and locking systems.

---

---

## Lesson 30 – SQL Logic Blocks

---

---

### Objectives

This lesson introduces you to SQL logic blocks. SQL logic blocks are the mechanism for passing SQL commands from HP ALLBASE/4GL to HP ALLBASE/SQL. They provide the means for accessing and updating HP ALLBASE/SQL tables.

In this lesson you will:

- Define and generate a number of SQL logic blocks.
- Use the SQL command in process and function logic blocks to execute SQL logic blocks.
- Use the SELECT command in an SQL logic block to declare and open a cursor on an HP ALLBASE/SQL table.
- Use the FILE command in conjunction with SQL logic blocks to access and update an HP ALLBASE/SQL table.
- Gain an understanding of rules for naming and using HP ALLBASE/SQL cursors.
- Use some of the HP ALLBASE/4GL facilities for detecting and handling errors detected by HP ALLBASE/SQL.

In this lesson, you will create a number of logic blocks and screens to add, delete and modify records in the *customer* table.

---

### Some Preparations

Once again, this part of the application uses a number of items that you have already defined in the KSAM-based *training* application. Use the utilities menu copying screen to copy the following items from the *training* application.

- **Variables**
  - file\_status
- **Alphanumeric constants**
  - record, no-record
- **Screens**

- main, customer\_scrn
- **Function Keys**
  - customer\_keys
- **Functions**
  - set\_add, set\_delete, set\_modify, set\_review
- **Messages**
  - add, delete, modify, review
  - add\_ok, delete\_ok, modify\_ok
  - no\_add, no\_delete, no\_modify, no\_review
  - file\_error, state\_code\_error

### The Customer Screen

You must modify the screen field details for the *customer\_scrn* screen to work with the record layout you have defined for the *customer* table. Call up the *customer\_scrn* screen on the screen field details screen, and modify the primary data movement specification for fields numbered three, four, and five. Modify the data movement fields to read *F-address.customer*, *F-address1.customer*, and *F-address2.customer* respectively.

### A New Constant

You must also define a new numeric constant for this application. Define a numeric constant with the name *beg\_end\_of\_file* and the value 60110. This constant is the value of the error return if a record is not found when you attempt to access an HP ALLBASE/SQL table. You will see how this constant is used shortly.

You can also define an application title *application\_name* now if you wish. (The application will work without this title.)

---

## Working With SQL Data

To understand the operation of the SQL command, the FILE command, and the techniques for manipulating SQL data, you must create a process, a function, and three SQL logic blocks. You can then run the application and see how these commands work together.

The structure of this part of the application is similar to the *customer\_proc* process in the KSAM-based *training* application.

## The Customer Process

The *Customer Details* option on the main menu calls a process *customer.proc*. This process displays the *customer\_scrn* screen, which in turn, executes the *customer\_func* function as an after function on the first screen field.

In overall terms, the operation of this process is the same as the corresponding process in the earlier application. The process displays the *customer\_scrn* screen, and then updates the customer table. The updates are performed with the FILE command, or with SQL commands in an SQL logic block depending on the nature of the update. The process uses switches number 1 to 4 to determine the nature of the update.

The meaning of the switches is as follows:

- 1 = Add a record
- 2 = Modify a record
- 3 = Delete a record
- 4 = Review a record

The process *customer.proc* reads as follows. Create and generate this process now.

```
1 MODE *WRITE customer
2 SCREEN customer_scrn
3 IF 3 *ON THEN ENTER 13
4 IF 1 *ON THEN ENTER 6
5 IF 2 *ON THEN ENTER 10 ELSE ENTER 8
6 FILE *INSERT customer
7 MESSAGE add_ok
8 SQL commit
9 ENTER 2
10 SQL update ; ENTER 16
11 MESSAGE modify_ok
12 ENTER 8
13 FILE *DELETE customer ; ENTER 16
14 MESSAGE delete_ok
15 ENTER 8
16 MESSAGE file_error
17 ENTER 2
```

This process contains SQL commands in steps 8 and 10. These commands execute the SQL logic blocks *commit* and *update*. Like some other commands, the SQL command allows you to specify an optional command to be executed if an error is detected. The SQL command in step 10 contains an optional error command.

The process operates as outlined below:

1. This step declares that the process can write to the *customer* table. All access to HP ALLBASE/SQL tables depends on the mode declared for the table in the current process.
2. This step displays the *customer\_scrn* screen. This screen calls the *customer\_func* function as an after function on the first field. This function is also similar to the corresponding function in the earlier application. It attempts to retrieve a record with a customer number matching the value entered by user from the *customer* table. You will create this function soon.
- 3, 4, 5. These steps test the status of user switches number 1 to 4, and pass control to the remaining steps in the process depending on the current update mode.
6. Control passes to this step if switch 1 is *on*. The FILE \*INSERT command inserts a new record into the *customer* table.
7. This step displays a confirmation message.
8. This step calls the SQL logic block *commit*. This SQL logic block contains the single SQL command COMMIT WORK. (HP ALLBASE/SQL requires that you issue an explicit COMMIT WORK command to make any transaction permanent in the database. If you don't commit a transaction, HP ALLBASE/SQL automatically reverses the transaction.)
9. After a transaction has been committed successfully, control passes back to step 2.
10. Control passes to this step if *modify* mode is selected. This step calls the SQL logic block *update*. This SQL logic block contains the commands to update an existing row in a table. This step contains an error command. If HP ALLBASE/SQL detects an error, control passes to step 16 to display an error message.
12. If the record is updated successfully in the previous step, this step passes control to step 8 to commit the transaction.
13. Control passes to this step if *delete* mode is selected. The FILE \*DELETE command deletes an existing row from the *customer* table.

The operation of the remaining steps in the process is self-explanatory.

This process introduces the SQL command. The important points to note are:

- The SQL command executes the commands in an SQL logic block. If the SQL block executes successfully, control passes to the next command in the logic block.

- The SQL command allows you to specify an optional error clause. If HP ALLBASE/SQL detects an error when it executes the commands in the SQL logic block, control passes to the error command. (The error handling logic can invoke an SQL logic block containing an SQLEXPLAIN command to retrieve details of the HP ALLBASE/SQL error condition. Refer to the *HP ALLBASE/4GL Developer Reference Manual* for details of this procedure.)

This process also introduces some of the means for handling SQL data from HP ALLBASE/4GL. The important points to note here are:

- You can use the FILE \*INSERT command to insert new rows into an HP ALLBASE/SQL table.
- You can use the FILE \*DELETE command to delete an existing row from a table.
- You must use SQL commands in an SQL logic block to modify an existing row in a table.

When you have created the next function and the SQL logic blocks, you can see how these data handling techniques work.

## The Customer Function

The function *customer\_func* is called as an after function on the first field on the *customer\_scrn* screen. The function uses the customer number entered by the user to retrieve a record from the *customer* table. If it retrieves a record successfully, it displays the record and sets *review* mode on. If the function does not find a record, *add* mode is set and the file buffer and screen buffer are cleared to allow the user to enter the details of a new record.

The function *customer\_func* reads as follows. Create and generate this function now.

```
1 SQL customer ; ENTER 13
2 FILE *NEXT customer ; ENTER 7
3 MOVE C-record V-file_status
4 VISIT set_review
5 SHOW *REFRESH
6 EXIT
7 IF *IOSTATUS <> N-beg_end_of_file THEN
 MESSAGE file_error ; EXIT
```



```
8 MOVE C-no_record V-file_status
9 VISIT set_add
10 CLEAR *MAP S-customer_name S-zip_code
11 FILE *BUFFER customer
12 EXIT
13 MESSAGE file_error
14 EXIT
```

The function operates as outlined below:

1. This step executes the SQL commands in the SQL logic block *customer*. This SQL block contains a SELECT command to retrieve the record matching the customer number value entered by the user. If an error occurs, control passes to step 13. Note that finding zero rows matching the search condition for the SELECT command is **not** an error condition.
2. This command operates in conjunction with the SELECT command called by the previous step. The FILE \*NEXT command retrieves the first record found by the SQL select command into the file buffer. If an error occurs in this command, control passes to step 7. Encountering an end-of-file condition without retrieving a record means that the SQL logic block in step 1 did not select any rows. This is treated as an error condition.
- 4, 5, 6. These steps perform the same actions as the corresponding steps in the earlier KSAM application *customer\_func* function.
7. This step tests the value in the communication area field \*IOSTATUS. If the value is 60110 (the value of the constant *N-beg\_end-of-file*), the FILE command has encountered an end-of-file condition. This means that there is no record in the table matching the customer number entered by the user. Control then passes to the remaining steps in the function.

If the file error is due to any condition other than an end-of-file condition, this step displays an error message, and the function exits.

Steps 9 through 12 in this function perform similar actions to the corresponding steps in the KSAM application *customer\_func* function. Step number 13 displays an error message if an error is detected during execution of the SQL logic block called by step 1.

## SQL Logic Blocks

The *customer\_proc* process and *customer\_func* function use three SQL logic blocks. These are the *customer* SQL logic block, the *update* SQL logic block, and the *commit* SQL logic block.

The next few pages describe the procedures for creating SQL logic blocks, and outline some of the rules covering the permissible SQL commands for SQL logic blocks.

### Defining an SQL Logic Block

Defining an SQL logic block is similar to defining a logic block. You must define an SQL logic block header, then define the details for the logic block, and finally, generate the logic block to convert it to an executable form.

The screens you use to define SQL logic blocks are accessible from the logic menu.

### The Customer SQL Logic Block

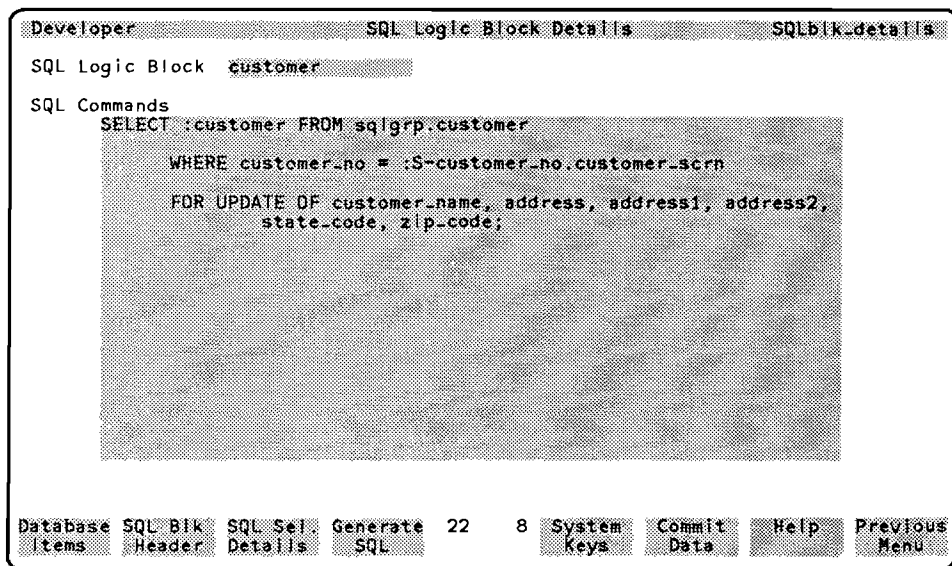
To create the *customer* SQL logic block, you must first create the logic block header. To do this, display the SQL logic block header screen.

This screen is similar in appearance and operation to the function and process header screens.

Enter **customer** in the *SQL Logic Block Name* field, and enter a suitable description in the description fields. Press the **Commit Data** function key to create the SQL logic block header.

### SQL Logic Block Details

You can now create the details for the SQL logic block. Go to the SQL logic block details screen by pressing the **SQL Blk Details** function key.



This screen contains a free format text entry area that allows you to enter the SQL commands that make up the SQL block.

You can use the normal terminal keyboard keys to enter and edit the commands in this text entry area.

The *customer* SQL block contains a single SQL SELECT command as follows:

```
SELECT :customer FROM sqlgrp.customer
WHERE customer_no = :S-customer_no.customer_scrn
FOR UPDATE OF customer_name, address, address1, address2,
state_code, zip_code;
```

Enter this command now. Make sure that you include the colon (:) before the term *customer* in the first line, and the term *S-customer\_no.customer\_scrn* in the second line. Note also that the command must be terminated with a semicolon (;) at the end of the last line of the command.

When you have entered the command, press the **Commit Data** function key.

## Generating an SQL Logic Block

You can now generate the *customer* SQL logic block. Start the generation process by pressing the **Generate SQL** function key.

HP ALLBASE/4GL calls the generate program, and then connects to the database environment for the application.

During SQL logic block generation, HP ALLBASE/4GL validates and resolves references to data items. The generation process also converts the SQL commands in the SQL block to an executable form and stores them in the database environment as stored sections in a database module.

### Generation Errors

If HP ALLBASE/4GL detects any errors in an SQL logic block during generation, it displays a generate error screen.

The generate error screen may specify a step number and a character position for the error. For SQL logic blocks, this position refers to the cursor position at or near the cause of the error. (Most HP terminals display the current cursor position in terms of the screen line number and column number between the two blocks of function key labels.)

If you receive an error message when you generate this SQL block, check that you have entered it correctly. Make sure that you have terminated the command with a semicolon, and that you have included the colons where they are required.

### SQL Logic Block Limitations

The *customer* SQL logic block introduces a number of features and limitations that apply to all SQL logic blocks.

The next few paragraphs describe these items.

- Some SQL commands are not permitted in SQL logic blocks. The *HP ALLBASE/4GL Developer Reference Manual* contains a full list of the commands you cannot use.
- If an SQL logic block contains a SELECT command, this command must be the only command in the SQL logic block.
- If an SQL logic block doesn't contain a SELECT command, it can contain up to eight other SQL commands.
- Each command in an SQL logic block must be terminated with a semicolon (;).

## The SELECT Command

HP ALLBASE/4GL modifies the syntax of the SQL SELECT command slightly. In SQL logic blocks, you must enter the command in the following format.

```
SELECT { : table_name } FROM clauses [other clauses]
 { : select_list }
 [FOR UPDATE OF column_name [, column_name ...]];
```

In this expression, *table\_name* is an HP ALLBASE/SQL table name that has been defined on the file/SQL table definition screen in the dictionary, and *select\_list* is a select list name that has been defined in the dictionary. A host variable reference cannot be used to specify the name of the table or select list.

You cannot use the SELECT \* FROM ... form of the SELECT command in an SQL logic block, and you cannot use the INTO clause in a SELECT command.

At generate time, HP ALLBASE/4GL converts this command to a DECLARE CURSOR command in the format:

```
DECLARE SQL_block_name CURSOR FOR select_command
FOR UPDATE OF
```

where *SQL\_block\_name* is the name of the SQL logic block containing the SELECT command.

When HP ALLBASE/4GL executes an SQL block containing a SELECT command, it performs the same actions as an SQL DECLARE CURSOR command and an OPEN CURSOR command. You will see how HP ALLBASE/4GL uses this SQL cursor when you run the application.

## Host Variables

You can include references to the following data items in SQL logic blocks.

- File record fields and work area fields.
- Variables and calculated items.
- Numeric and alphanumeric constants.
- Screen fields.

However, you cannot use host variable referencing to replace the name of a table.

References to HP ALLBASE/4GL data items in SQL logic blocks must be prefixed with a colon (:). The reference must be in the format:

```
:data_ref
```

where *data\_ref* is the full reference to the item.

For example, the term *:S-customer.no.customer.scrn* in the SQL logic block you have just created is a reference to the *customer.no* field on the *customer.scrn* screen. (Note that references to screen fields in SQL logic blocks must be fully qualified references by name.) Refer to the *HP ALLBASE/4GL Developer Reference Manual* for more information about the referencing rules.

## Two More SQL Logic Blocks

This application requires two further SQL logic blocks. These are the *update* SQL logic block and the *commit* SQL logic block.

### SQL Logic Block – *update*

The *update* SQL logic block contains the following SQL command.

```
UPDATE sqlgrp.customer SET
 customer_name = :F-customer_name.customer,
 address = :F-address.customer,
 address1 = :F-address1.customer,
 address2 = :F-address2.customer,
 state_code = :F-state_code.customer,
 zip_code = :F-zip_code.customer
WHERE CURRENT OF customer;
```

Create and generate this SQL logic block now.

### SQL logic block – *commit*

The *commit* SQL logic block contains the single command:

```
COMMIT WORK;
```

Remember to terminate the command with a semicolon. Create and generate this SQL logic block now.

---

## SQL Application Generation

Before you try to run the application, you must generate the items you copied from the original *training* application. To do this, select the *Generates* option on the main menu, and select the *All* option on the generates menu.

When you generate an entire application, or all the SQL logic blocks in an application, HP ALLBASE/4GL stores the definition of the database module for the application in a file in the HP4DBMSQL group in the HP4GL account. The name of this file is the same as the name of the application.

During generation, HP ALLBASE/4GL connects to the database environment and installs the executable form of the SQL logic blocks as stored sections in a database module. The name of this module is *owner\_group.name* where *owner\_group* is the SQL owner group for the application, and *name* is the name of the application.

---

## SQL Data Manipulation

You are now ready to test the application and use HP ALLBASE/4GL to manipulate data in an HP ALLBASE/SQL table.

Use the application testing option to run the application. At this stage, the *Customer Details* option on the application main menu is the only usable selection. Select this option to display the *customer\_scrn* screen.

Enter a number in the *Number* field, and press **Return**. HP ALLBASE/4GL displays a message to tell you that it is connecting to the application database. This is typical behavior for an application at run-time. When the user activates an action that accesses the database, the application automatically connects to the database. HP ALLBASE/4GL then maintains the connection until the user terminates the application.

Add a number of *customer\_rcrd* records, and then try using the application to modify, delete, and review them. These various actions use some of the facilities for accessing and working with HP ALLBASE/SQL data. As you work through the next few sections, you may want to use the trace mode facility to examine the commands that HP ALLBASE/4GL executes.

## Retrieving SQL Data

When you enter a customer number and press **Return**, HP ALLBASE/4GL executes the *customer.func* function as an after function on the *customer number* field. This function contains the commands:

```
1 SQL customer ; ENTER 13
2 FILE *NEXT customer ; ENTER 7
```

These commands demonstrate the standard method of retrieving data from an SQL table. The SQL logic block *customer* contains the command:

```
SELECT :customer FROM sqlgrp.customer
WHERE customer.no = :S-customer.no.customer.scrn
FOR UPDATE OF customer.name, address, address1, address2,
state.code, zip.code;
```

In its generated form, this command is equivalent to the following SQL DECLARE CURSOR command:

```
DECLARE customer CURSOR FOR
SELECT * FROM sqlgrp.customer
WHERE customer.no = :S-customer.no.customer.scrn
FOR UPDATE OF customer.name, address, address1, address2,
state.code, zip.code;
```

Note that the cursor name in this command is the name of the SQL logic block containing the SELECT command. This is a standard rule that applies to all HP ALLBASE/SQL cursors defined from HP ALLBASE/4GL.

At run-time HP ALLBASE/4GL executes this command as a DECLARE CURSOR command, followed by an OPEN CURSOR command.

The effect of this command is to declare and open a cursor on the *customer* table. The active set for the cursor is the set of rows in the table that match the search condition specified by the WHERE clause in the SELECT command. In this particular case, the active set consists of a maximum of one record because the *customer.no* field is defined as a unique index to the table. If no record matches the search condition, the active set for the cursor is empty.

The FILE \*NEXT command following the SQL command in the *customer.func* function is equivalent to an SQL FETCH command. This command positions the cursor on the first row of the active set, and retrieves the record into the file buffer for the *customer* table. The remaining commands in the logic block then use



this data as required. If the active set for the cursor is empty, the FILE \*NEXT command encounters an end-of-file condition. In this case, the data manager returns a value of 60110 to the communication area field \*IOSTATUS.

If the active set for the cursor defined by a SELECT command contains more than one row, the first FILE \*NEXT command positions the cursor on the first row of the active set, and retrieves the first row into the file buffer. Each subsequent FILE \*NEXT command positions the cursor on a subsequent record, and retrieves it into the file buffer. After the last record in the active set, a subsequent FILE \*NEXT command causes an end-of-file error.

This example demonstrates the standard technique for retrieving SQL data from an application. To retrieve a record you must perform the following steps:

1. Use a SELECT command in an SQL logic block to declare and open a cursor.
2. Use a FILE \*NEXT command in a logic block to position the cursor on the first record, and retrieve the record into the file buffer.

Both these steps must be in the same process, although they do not need to be in the same logic block. For example, you can use the SELECT command in an SQL logic block called from one function, and the FILE \*NEXT command in another function, provided both functions are called from the same process. You cannot use a SELECT command in one process, and the FILE \*NEXT command in another process because HP ALLBASE/4GL automatically closes all open HP ALLBASE/SQL cursors at the end of a process.

If you intend to use the HP ALLBASE/SQL cursor to update or delete records in the table, you must include the FOR UPDATE OF clause in the SELECT command to specify the columns of the table that are being updated. In this example, commands in the *customer\_proc* process use the cursor to update and delete records so the SELECT command does contain a FOR UPDATE OF clause.

## Adding New Records

When you add a new record using *Add* option, the record is inserted into the *customer* table by the FILE \*INSERT command in step 6 of the *customer\_proc* process.

In this application, control passes to step 6 after the user has entered a customer number, and selected *Add* mode. The *set.add* function does not allow the user to select *Add* mode if the record already exists.

Step 6 of the *customer\_proc* process contains the command:

```
FILE *INSERT customer
```

This command inserts the current contents of the *customer* file buffer into the *customer* table as a new row.

This FILE \*INSERT command is equivalent to the SQL command:

```
INSERT INTO sqlgrp.customer
 VALUES (:F-customer.no.customer,
 :F-customer.name.customer,
 :F-address.customer,
 :F-address1.customer,
 :F-address2.customer,
 :F-state_code.customer,
 :F-zip_code.customer,
```

In this application, the user cannot attempt to insert a row into the *customer* table if a row with the same value in the *customer\_no* field already exists. If the application logic allows the user to attempt an insert operation that causes an index uniqueness violation, HP ALLBASE/SQL generates an error message. If the FILE command has an optional error command, control would pass to the error command if such an error occurred.

This example demonstrates one technique for inserting a new record into an SQL table. You can also use the SQL INSERT command directly in an SQL logic block.

## Modifying a Record

To modify a record, the user must enter a number in the *customer\_no* field. When the user presses **Return** for the *customer\_no* field, HP ALLBASE/4GL executes the *customer\_func* function. If the function retrieves a record, the user can select *Modify* mode.

When *Modify* mode is set, control passes to step 10 of the *customer\_proc* process on exit from the screen.

This step calls the SQL logic block *update*. This SQL logic block contains the command:

```
UPDATE sqlgrp.customer SET
 customer_name = :F-customer.name.customer,
 address = :F-address.customer,
 address1 = :F-address1.customer,
```

```

address2 = :F-address2.customer,
state_code = :F-state_code.customer,
zip_code = :F-zip_code.customer
WHERE CURRENT OF customer;

```

This command demonstrates the use of an HP ALLBASE/SQL cursor to update a table. The SQL command and the FILE \*NEXT command in the *customer\_func* function declare and open the cursor, and position the cursor.

The important point to note is the name of the cursor. The cursor name is always the name of the SQL logic block containing the SELECT command that declares and opens the cursor. In this example the cursor name is *customer* since the cursor was declared and opened by the SQL logic block *customer*. The SELECT command must contain the FOR UPDATE OF clause to specify the columns to be updated. You cannot update columns that are not specified in the FOR UPDATE OF clause.

This example shows one of the standard techniques for modifying a record. To modify an existing row in a table, you must perform the following actions:

1. Use a SELECT command in an SQL logic block to declare and open a cursor on the table to be updated. The SELECT command must contain a FOR UPDATE OF clause to specify the columns to be updated.
2. Use a FILE \*NEXT command in a logic block to position the cursor on the row to be modified. The FILE \*NEXT command retrieves the row into the file buffer, so you can test the buffer contents to verify that you have retrieved the correct table row before you update it.
3. Use an UPDATE command in an SQL logic block to update the current table row. The UPDATE command must contain a WHERE CURRENT OF *cursor\_name* clause. The cursor name is the name of the SQL logic block containing the SELECT command that defined the cursor.

## Deleting Records

To delete a record, the user must enter a number in the *customer\_no* field. When the user presses **Return** for the *customer\_no* field, HP ALLBASE/4GL executes the *customer\_func* function. If the function retrieves a record, the user can select *Delete* mode.

If *Delete* mode is selected, control passes to step 13 of the *customer\_proc* process on exit from the screen. This step contains the command:

**FILE \*DELETE** customer; **ENTER** 16

This command is equivalent to the following SQL command.

```
DELETE FROM sqlgrp.customer WHERE CURRENT OF customer;
```

This command uses a cursor to delete a table row. The **SELECT** command in the SQL block called by the *customer\_func* function declares and opens the cursor. The **FILE \*NEXT** command in the function positions the cursor on the first row of the active set, and retrieves the row into the file buffer. Once again, the name of the cursor is the name of the SQL logic block containing the **SELECT** command that declares and opens the cursor.

This example shows one of the standard techniques for deleting a record. To delete an existing row in a table, you must perform the following actions:

1. Use a **SELECT** command in an SQL logic block to declare and open a cursor on the table to be updated. The **SELECT** command must contain a **FOR UPDATE OF** clause to specify the columns to be updated.
2. Use a **FILE \*NEXT** command in a logic block to position the cursor on the row to be modified. The **FILE \*NEXT** command retrieves the row into the file buffer, so you can test the buffer contents to verify that you have retrieved the correct table row before you delete it.
3. Use the **FILE \*DELETE** command in a logic block to delete the record. This command is exactly equivalent to the SQL **DELETE WHERE CURRENT** command.

## Committing Transactions

HP ALLBASE/SQL requires you to issue an explicit **COMMIT WORK** command to store a transaction in the database. The *commit* SQL logic block contains a commit work command.

HP ALLBASE/4GL executes this command as step 8 of the *customer\_proc* process. Control passes to this step after an insert, delete, modify, or review operation has been performed successfully. The **COMMIT** command closes all open cursors.

HP ALLBASE/SQL automatically reverses any incomplete transaction if you terminate a transaction or attempt to start a new transaction without committing a currently open transaction.

---

## Summary

In this lesson, you have created three SQL logic blocks and a number of other items. These SQL logic blocks have allowed you to access and update data in an HP ALLBASE/SQL table.

### SQL Logic Blocks

SQL logic blocks are the means of passing SQL commands to HP ALLBASE/SQL. The important characteristics of SQL logic blocks are listed below.

- An SQL logic block comprises a header and the logic block details. You cannot complete the details for an SQL logic block until you have defined the header.
- If an SQL logic block contains a SELECT command it must be the only command in the logic block. An SQL logic block can contain up to eight SQL commands if it does not contain a SELECT command.
- SQL logic blocks must be generated to produce an executable form of the commands in the logic block.
- The commands in an SQL logic block are executed by the SQL command in a function or a process. You can specify an optional error command on the SQL command. Control passes to the error command if HP ALLBASE/SQL detects an error while it is executing the commands in the SQL logic block.

### The SELECT Command

The *customer* SQL logic block contains a SELECT command. The SELECT command has the following characteristics:

- During generation, HP ALLBASE/4GL converts the SELECT command into a DECLARE CURSOR command. At run-time, the SELECT command becomes a DECLARE CURSOR command and an OPEN CURSOR command.
- You can use a SELECT command in an SQL logic block, followed by a FILE \*NEXT command in a logic block to access an HP ALLBASE/SQL table and retrieve records from the table. The FILE \*NEXT command is equivalent to the SQL FETCH command.

- If you intend to use the **SELECT** command to declare a cursor for updating a table, you must include a **FOR UPDATE OF** clause in the **SELECT** command to specify the columns that are to be updated.
- The name for a cursor defined by a **SELECT** command in an SQL logic block is the name of the SQL logic block containing the **SELECT** command.

## SQL Data Manipulation

In combination, commands in SQL logic blocks, and commands in logic blocks allow you to retrieve, insert, modify, or delete rows in an HP ALLBASE/SQL table.

### Retrieving Data

To retrieve rows from an HP ALLBASE/SQL table, you can use the following procedure:

1. Use a **SELECT** command in an SQL logic block to declare and open a cursor.
2. Use a **FILE \*NEXT** command in a function or process to position the cursor on the first row of the active set, and retrieve the row into the file buffer.
3. Use further **FILE \*NEXT** commands to position the cursor on subsequent rows (if any) of the active set, and retrieve these rows into the file buffer.

### Inserting Data

To insert new rows into an HP ALLBASE/SQL table, you can use the **FILE \*INSERT** command in a function or a process. This command inserts the current file buffer contents into the table as a new row.

### Modifying Data

To modify an existing row in an HP ALLBASE/SQL table, you can use the following procedure:

1. Use a **SELECT** command in an SQL logic block to declare and open a cursor. The **SELECT** command must contain a **FOR UPDATE OF** clause.

2. Use an SQL UPDATE command with a WHERE CURRENT OF clause in a subsequent SQL logic block to update the current record for the cursor. The cursor name is the name of the SQL logic block containing the SELECT command.

Both SQL logic blocks must be in the same process.

### Deleting Records

To delete an existing record in an HP ALLBASE/SQL table, you can use the following procedure:

1. Use a SELECT command in an SQL logic block to declare and open a cursor on the table.
2. Use a FILE \*NEXT command in a function or process to position the cursor on the first row of the active set and retrieve the record into the file buffer.
3. Use a FILE \*DELETE command in a function or process to delete the record.

The SQL logic block containing the SELECT command, and the subsequent FILE commands must be executed in the same process.

### Commit Work

HP ALLBASE/SQL requires you to issue an explicit COMMIT WORK command to complete a transaction and make it permanent in the database. If you don't issue the commit work command, HP ALLBASE/SQL reverses any currently open transaction.

You can use an SQL logic block called from a function or process to issue the COMMIT WORK command.

---

## Lesson 31 – Reporting From an HP ALLBASE/SQL Database

---

### Objectives

In this lesson you will develop a simple report that prints all the records in the customer table.

The format of this report is identical to the *customer\_rept* report you created for the KSAM-based *training* application.

### Preparations

This report is almost identical to the earlier report. The only difference is that this report requires a start of report function.

Use the utilities menu copying screen to copy the *customer\_rept* report from the *training* application.

---

### Report Header

HP ALLBASE/4GL can use an existing HP ALLBASE/SQL table as the primary file for a report.

To access the table, you must use a start of report function to call an SQL logic block. This SQL logic block must contain a **SELECT** command to declare and open a cursor on the table. The rows in the active set for the cursor then become the primary file for the report.

#### The Customer Report

Call up the *customer\_rept* report on the report header screen.

Press the **Return** key to step over the fields on the screen until you reach the *Start of Report Function Name* field. Enter the name **select\_customer** in this field. Press the **Commit Data** function key to update the report header.



## Start of Report Function

### Function *select\_customer*

The *select\_customer* function reads as follows:

- 1 SQL *select\_customer*
- 2 EXIT

This function simply calls the SQL logic block *select\_customer*. Note that this function does **not** contain a FILE \*NEXT command. The report generator performs the equivalent operation to a FILE \*NEXT command when it accesses the primary file for the report. In this case, the primary file for the report is the active set for the cursor defined by the SQL block *select\_customer*.

Create and generate this function now.

## Report SQL Logic Block

### SQL Logic Block *select\_customer*

The *select\_customer* SQL logic block contains the following command:

```
SELECT :customer FROM sqlgrp.customer
ORDER BY customer-no;
```

This SELECT command contains an ORDER BY clause. This clause ensures that records are retrieved in the required order for the report.

Create and generate this SQL logic block now.

## Completing the Report

Call up the *customer\_rept* report with the report painter. You can now edit the field details for the customer address fields on the D1.01, D1.02, and D1.03 lines so they refer to the fields *F-address.customer*, *F-address1.customer*, and *F-address2.customer* respectively.

The *customer\_rept* report does not require any further changes. Call up the report again on the report header screen, and press the **Generate Report** function key to generate the report.

You should now be able to run the report by selecting the *Customer Report* option on the application main menu.

## Summary

The report generator can use an HP ALLBASE/SQL table as the primary file for a report. To use an HP ALLBASE/SQL table for a report, you must use a start of report function to execute SQL logic block containing a SELECT command. The SELECT command declares and opens a cursor on the table.

The report generator then uses the active set for the cursor as the report primary file. All the normal facilities are available to format the report.

# Contents

---

## Chapter 12: Further HP ALLBASE/SQL Techniques

|                                 |        |
|---------------------------------|--------|
| Introduction . . . . .          | 12 - 1 |
| Application Structure . . . . . | 12 - 1 |

### Lesson 32 – Using SQL Data

|                                   |         |
|-----------------------------------|---------|
| Objectives . . . . .              | 12 - 3  |
| Preparations . . . . .            | 12 - 3  |
| Dictionary Requirements . . . . . | 12 - 4  |
| Record Layouts . . . . .          | 12 - 4  |
| HP ALLBASE/SQL Tables . . . . .   | 12 - 5  |
| Application Logic . . . . .       | 12 - 6  |
| Processes . . . . .               | 12 - 6  |
| Data Retrieval . . . . .          | 12 - 7  |
| Updating SQL Data . . . . .       | 12 - 9  |
| Adding New Records . . . . .      | 12 - 10 |
| Modifying Records . . . . .       | 12 - 11 |
| Deleting Records . . . . .        | 12 - 13 |
| Deleting a Product . . . . .      | 12 - 13 |
| Deleting Options . . . . .        | 12 - 16 |
| Summary . . . . .                 | 12 - 17 |
| SQL Logic Blocks . . . . .        | 12 - 17 |
| *ROWCOUNT . . . . .               | 12 - 17 |

### Lesson 33 – SQL Select Lists

|                                        |         |
|----------------------------------------|---------|
| Objectives . . . . .                   | 12 - 18 |
| Select Lists . . . . .                 | 12 - 18 |
| Select List Definition . . . . .       | 12 - 19 |
| Generating a Select List . . . . .     | 12 - 21 |
| Scrolling Data on the Screen . . . . . | 12 - 21 |
| Running the Application . . . . .      | 12 - 23 |
| Summary . . . . .                      | 12 - 23 |

## **Lesson 34 – Further HP ALLBASE/SQL Reporting Techniques**

|                                              |         |
|----------------------------------------------|---------|
| Objectives . . . . .                         | 12 – 25 |
| Preparations . . . . .                       | 12 – 25 |
| The Product Report . . . . .                 | 12 – 26 |
| Retrieving Data for Reporting . . . . .      | 12 – 26 |
| Report Header . . . . .                      | 12 – 28 |
| Report Sorting . . . . .                     | 12 – 28 |
| Record Selection and File Linkages . . . . . | 12 – 28 |
| Line Headers . . . . .                       | 12 – 29 |
| Painting the Report . . . . .                | 12 – 29 |
| Completing the Report . . . . .              | 12 – 32 |
| Generating the Report . . . . .              | 12 – 32 |
| Summary . . . . .                            | 12 – 32 |

# 12 Further HP ALLBASE/SQL Techniques

---

## Introduction

This chapter describes a series of enhancements to the existing *sqltrain* application to add features that perform the same task as the *product* process in the KSAM-based *training* application.

As you work through this chapter, you will define a number of items to allow you to manipulate data in two HP ALLBASE/SQL tables and report on the data in the two tables.

From the user's viewpoint this part of the application appears to be identical to the corresponding parts of the KSAM-based application. The processing rules are also the same.

This chapter is divided into three lessons. In the first lesson, you will create a number of dictionary items, and a number of processes and functions. The second lesson in this chapter introduces HP ALLBASE/4GL select lists. Select lists provide an additional means of accessing and manipulating SQL data. The third lesson in this chapter describes some further reporting techniques using HP ALLBASE/SQL data.

Throughout this lesson you will use most of the facilities that have been introduced in earlier lessons. Rather than describe these facilities in detail, the explanations in this chapter concentrate on the facilities and techniques for accessing and using HP ALLBASE/SQL data.

Once again, this part of the application uses items that are copied from the KSAM-based *training* application. In some, you will need to modify the items you copy before you can use them in this application.

## Application Structure

The structure of this part of the application is very similar to the corresponding process in the KSAM-based *training* application. This application does not include any module building (as in lesson 28), but the process is exactly the same as for KSAM-based modules.

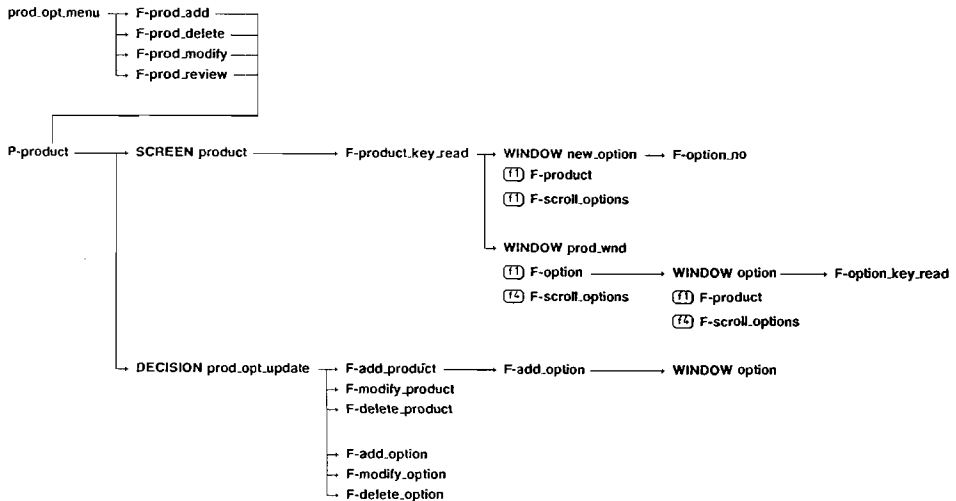
This application involves simultaneous use of two tables, according to a given set of rules. These are the *product* and *option* tables.

In operation, the user can select *Add*, *Modify*, *Delete*, or *Review* options from a menu. Each menu item calls a function that sets the appropriate mode, and then calls the *product* process.

This process displays a screen, and possibly a window, to allow the user to access product and option records as required. When the user terminates the screen, control returns to the process, which uses a decision table to call the appropriate database updating functions.

The application also allows the user to scroll the details for options for the current product on the screen.

The following diagram provides an overview of the structure of this part of the application.



## Lesson 32 – Using SQL Data

---

---

### Objectives

In this lesson, you will use a number of SQL logic blocks and HP ALLBASE/4GL logic blocks to work with SQL data.

---

### Preparations

This part of the application uses a number of components from the KSAM-based *training* application. Use the utilities menu copying screen to copy the following items from the *training* application.

- **Field Specifications**
  - cost, description, lead\_time, qty\_on\_hand, option\_no, product\_no, supplier\_no
- **Validation Range**
  - product\_no
- **Variables**
  - confirm, mode, product\_status, option\_status, current\_record
- **Alphanumeric Constants**
  - add, delete, modify, review, product, option
- **Numeric Constant**
  - zero
- **Messages**
  - add\_option, add\_prod
  - delete\_option, delete\_options, delete\_product, del\_prod\_opt
  - no\_option\_review, no\_product, no\_prod\_warn
  - option\_warn, opt\_add\_ok, opt\_delete\_fail, opt\_delete\_ok, opt\_modify\_ok, opt\_not\_exist
  - product\_no\_error, product\_warn, prod\_add\_ok, prod\_corrupt, prod\_delete\_fail, prod\_delete\_ok, prod\_modify\_ok, prod\_not\_exist

- **Screens**
  - prod\_opt\_menu, product, option, new\_option, prod\_wnd
- **Function Keys**
  - option, product
- **Functions**
  - add\_option, add\_product, delete\_option
  - modify\_option, modify\_product
  - new\_option, option, option\_key\_read, option\_no
  - product, product\_key\_read
  - prod\_add, prod\_delete, prod\_modify, prod\_review
  - scroll\_options, get\_1st\_option
- **Process**
  - prod\_opt
- **Decision Table**
  - prod\_opt\_update

Many of these items can be used directly without modification. Other items require some changes before they can be used in this application. You also need to create some new items while you develop this application.

## Dictionary Requirements

This application uses two HP ALLBASE/SQL tables, called *product* and *option*. To create these tables, you must create and generate two record layouts.

### Record Layouts

Create and generate a record layout called *product* with the following structure.

| Field No. | Field Spec. Name | Key No. | Duplicate Keys |
|-----------|------------------|---------|----------------|
| 1         | product_no       | 1       | N              |
| 2         | description      |         |                |
| 3         | supplier_no      |         |                |
| 4         | lead_time        |         |                |



Create and generate another record layout called *option* with the following structure. Note that this record layout does not require any fields to be defined as key fields.

| Field No. | Field Spec. | Name |
|-----------|-------------|------|
| 1         | product_no  |      |
| 2         | option_no   |      |
| 3         | description |      |
| 4         | cost        |      |
| 5         | qty_on-hand |      |

## HP ALLBASE/SQL Tables

Define and create two tables called *product* and *option* according to the following definitions.

### product

|                  |                |
|------------------|----------------|
| File Name        | <b>product</b> |
| File Type        | <b>S</b>       |
| External Name    | <b>PRODUCT</b> |
| SQL DBfileset    | <b>PRODFS</b>  |
| SQL Access Class | <b>1</b>       |
| Record Layout    | <b>product</b> |

### option

|                  |               |
|------------------|---------------|
| File Name        | <b>option</b> |
| File Type        | <b>S</b>      |
| External Name    | <b>OPTION</b> |
| SQL DBfileset    | <b>PRODFS</b> |
| SQL Access Class | <b>1</b>      |
| Record Layout    | <b>option</b> |

---

## Application Logic

The following subsections show the processes and functions used by this part of the application. In general, these subsections don't include complete descriptions of the step by step operation of the logic blocks.

Where necessary, these pages describe the SQL commands and HP ALLBASE/4GL logic commands that operate on SQL data. The following subsections describe the application components in the context of data retrieval, adding new records, modifying data, and deleting data.

The technique for scrolling data on the screen is described in the next lesson.

---

## Processes

This part of the application uses one process. This is the process *product*.

### Process – *product*

The purpose of the *product* process is similar to the same process in the KSAM-based *training* application. It is called from one of the mode-setting functions. It then displays the *product* screen, and calls the *prod\_opt\_update* decision table to update the appropriate tables.

The *product* process contains the following commands. Create and generate this process now.

- 1 **MODE** \*WRITE product option
- 2 **MOVE** C-product V-current\_record
- 3 **MOVE** C-no\_record V-product\_status
- 4 **SCREEN** product
- 5 **DECISION** prod\_opt\_update
- 6 **SQL** commit
- 7 **ON** 1
- 8 **ENTER** 4

Step 6 of this process calls the SQL logic block *commit*. This SQL logic block contains a COMMIT WORK command to terminate the current SQL transaction.

Step 7 sets user switch number 1 *on*. This switch resets the mechanism for scrolling option data on the screen. (This is necessary to ensure that the function *get\_1st\_option* is executed if the user presses the **Scroll Options** function key after pressing the **Commit Data** function key. This is explained in more detail in the next lesson.)

---

## Data Retrieval

The functions that perform the data retrieval activities in this application are the functions associated with the *product.no* and *option.no* screen fields. These are the functions *product\_key\_read* and *option\_key\_read*.

### Function – *product\_key\_read*

The *product\_key\_read* function contains the following commands. This is one of the functions you copied from the earlier *training* application. Modify the function by inserting the SQL command at step 8, and modifying the FILE command at step 9. You must also modify the IF command at step 14.

```

1 IF *ENTERED *ON THEN ENTER 8
2 IF V-current_record = C-product THEN
 WINDOW prod_wnd ; EXIT
3 SHOW *REFRESH S-product.no S-lead_time
4 MATH F-option.no.option + 1 = F-option.no.option
5 SHOW *REFRESH S-option.no
6 MOVE "5" *FIELDNO
7 EXIT
8 SQL prod_key_sel
9 FILE *NEXT product ; ENTER 14
10 SHOW *REFRESH S-product.no S-lead_time
11 MOVE C-record V-product_status
12 ON 1
13 EXIT
14 IF *IOSTATUS <> N-beg_end_of_file THEN MESSAGE file_error ;
 SERIES 15 17 ; EXIT
15 FILE *BUFFER product
16 CLEAR *MAP S-description S-lead_time
17 MOVE C-no_record V-product_status
18 IF V-mode <> C-add THEN MESSAGE prod_not_exist
 ELSE MESSAGE add_prod ; VISIT new_option ; TIE 2
19 EXIT

```

Apart from the SQL command, this function operates in a similar manner to its counterpart in the earlier application. Step 8 of this function calls the SQL logic block *prod\_key\_sel*. This SQL logic block declares and opens a cursor on the *product* table. The FILE \*NEXT command at step 9 retrieves the first record of the active set for the cursor.

**SQL Logic Block – *prod\_key\_sel***

The *prod\_key\_sel* SQL logic block contains the following command. Create and generate this SQL logic block now.

```

SELECT :product FROM sqlgrp.product
 WHERE product_no = :S-product_no.product
 FOR UPDATE OF description, supplier_no, lead_time;

```

This SQL logic block declares and opens a cursor on the *product* table. The active set for this cursor is the set of rows containing a value in the *product\_no* field that matches the value entered in the *product\_no* screen field. Since this table has a unique index defined for the *product\_no* field, there can never be more than one row in the active set.

**Function – *option\_key\_read***

The *option\_key\_read* function is also one of the functions you copied from the *training* application.

This function requires a new variable *V-option\_no*. This variable has a length of three characters, and a type N edit code. You must create this variable before you can generate the *option\_key\_read* function.

Insert step 3, and modify steps 4, 5, and 9 so that the function reads as follows:

```

1 IF V-current_record = C-product THEN EXIT
2 IF V-mode = C-delete & * = N-zero THEN
 MESSAGE del_prod_opt ; EXIT
3 MOVE * V-option_no
4 SQL option_key_sel
5 FILE *NEXT option ; ENTER 9
6 SHOW *REFRESH S-option_no S-qty-on-hand
7 MOVE C-record V-option_status
8 EXIT
9 IF *IOSTATUS <> N-beg_end_of_file THEN MESSAGE file_error ;
 SERIES 10 12 ; EXIT
10 FILE *BUFFER option
11 CLEAR *S 6 8
12 MOVE C-no_record V-option_status
13 IF V-mode <> C-add THEN MESSAGE opt_not_exist
 ELSE MESSAGE add_option
14 EXIT

```

Step 3 of this function moves the value in the current screen field (the *option\_no* field on the option window) to the variable *V-option\_no*. The SQL logic block *option\_key\_sel* that is called by step 4 uses this variable.

This function does not need to generate a unique key as was required in the KSAM-based *training* application. This application does not require a unique key for the option table since the *opt\_key\_sel* SQL logic block uses a combined search condition to find the required record in the *option* table.

### SQL Logic Block – *option\_key\_sel*

The *option\_key\_sel* SQL logic block contains the following command:

```
SELECT :option FROM sqlgrp.option
 WHERE product_no = :F-product_no.product
 AND option_no = :V-option_no
FOR UPDATE OF product_no, option_no, description, cost,
 qty_on_hand;
```

This SQL logic block declares and opens a cursor on the *option* table. The SQL logic block also shows the use of an AND clause to specify a compound search condition. In this case, the SELECT command locates a row in the table with the *product\_no* field equal to the current value in the *product\_no* field on the *product* file buffer, and the *option\_no* field equal to the value in the variable *V-option\_no*. The FILE \*NEXT command at step 5 of the *option\_key\_read* function retrieves the first record in the active set for the cursor.

This example demonstrates the host variable referencing technique for SQL logic blocks. In this SQL logic block, the term *:F-product\_no.product* is a reference to the field *product\_no* on the file record buffer *product*, and the term *:V-option\_no* is a reference to the variable *option\_no*. Note that you must use the full reference for the data item, and prefix the reference with a colon (:).

## Updating SQL Data

This application uses a number of functions to add, modify, or delete records in the *product* and *option* tables. As in the earlier KSAM application, these functions are called by the *prod\_opt\_update* decision table.

Most of these functions are similar to the corresponding functions in the KSAM-based *training* application. The main difference is that some functions call SQL logic blocks to access the HP ALLBASE/SQL tables.

---

## Adding New Records

### Function – *add\_product*

The *add\_product* function is one of the functions you copied from the earlier KSAM application. Modify step 2 of the function so it reads as follows:

```
1 IF V-product_status = C-record THEN
 MESSAGE product_warn ; EXIT
2 FILE *INSERT product ; ENTER 8
3 MOVE C-record V-product_status
4 ON 1
5 MESSAGE prod_add_ok
6 VISIT add_option
7 EXIT
8 MESSAGE file_error
9 EXIT
```

This function contains a FILE \*INSERT command to insert a new record into the *product* table. The FILE \*INSERT command inserts the current buffer contents into the table as a new row.

The FILE \*INSERT command is equivalent to the following SQL command:

```
INSERT INTO sqlgrp.product
VALUES (:F-product_no.product,
 :F-description.product,
 :F-supplier_no.product,
 :F-lead_time.product);
```

In this application, the *add\_product* function cannot be executed unless the record about to be added has been shown not to exist. Even though a uniqueness violation error cannot occur when the user adds a record, HP ALLBASE/SQL may still detect an error condition. (For example, an error would occur if the user does not have the correct access authority for the *product* table). If HP ALLBASE/SQL does detect an error, control passes to step 8.

### Function – *add\_option*

The *add\_option* function is called directly by the *prod\_opt\_update* decision table. It is also called from the *add\_product* function by the VISIT command at step 6.

The *add\_option* function is one of the functions you copied from the earlier application. Delete the command at step 3, and modify the FILE command so the function reads as follows:

```

1 IF V-option_status = C-record
 THEN MESSAGE option_warn ; EXIT
2 MOVE S-product_no F-product_no.option
3 FILE *INSERT option ; ENTER 7
4 MESSAGE opt_add_ok
5 IF F-option_no.option = N-zero THEN
 MOVE C-option V-current_record ;
 WINDOW option ; MOVE "1" *NEWTIE
6 EXIT
7 MESSAGE file_error
8 EXIT

```

The FILE \*INSERT command in this function inserts the current buffer contents into the *option* table as a new row.

The FILE \*INSERT command is equivalent to the SQL command:

```

INSERT INTO sqlgrp.option
VALUES (:F-product_no.option,
 :F-option_no.option,
 :F-description.option,
 :F-cost.option,
 :F-qty_on_hand.option);

```

## Modifying Records

### Function – *modify\_product*

The *modify\_product* function is called from the *prod\_opt\_update* decision table. This is one of the functions you copied from the earlier application. Modify step 2 of the function to read as shown below:

```

1 IF V-product_status = C-no_record THEN
 MESSAGE no_prod_warn ; EXIT
2 SQL modify_product ; ENTER 5
3 MESSAGE prod_modify_ok
4 EXIT
5 MESSAGE file_error
6 EXIT

```

This function calls the SQL logic block *modify-product*.

### SQL Logic Block – *modify-product*

The *modify-product* SQL logic block modifies a row in the *product* table. Create and generate this SQL block now.

```
UPDATE sqlgrp.product SET
 description = :F-description.product,
 supplier_no = :F-supplier_no.product,
 lead_time = :F-lead_time.product
WHERE CURRENT OF prod_key_sel;
```

This SQL logic uses a cursor to update the *product* table. The name of the cursor is *prod\_key\_sel*. The function *product\_key\_read* declares and opens this cursor by calling the SQL logic block *prod\_key\_sel*. Note that the cursor name is the name of the SQL logic block containing the SELECT command that declares and opens the cursor.

### Function – *modify-option*

The *modify-option* function is called from the *prod-opt-update* decision table. This is also one of the functions you copied from the earlier application. Modify step 2 of the function so it reads as follows:

```
1 IF V-option_status = C-no_record THEN
 MESSAGE no_option_warn ; EXIT
2 SQL modify_option ; ENTER 5
3 MESSAGE opt_modify_ok
4 EXIT
5 MESSAGE file_error
6 EXIT
```

This function calls the *modify-option* SQL logic block to update the *option* table.

### SQL Logic Block – *modify-option*

The *modify-option* SQL logic block also uses a cursor to update the *option* table. Create and generate this SQL logic block now.

```
UPDATE sqlgrp.option SET
 description = :F-description.option,
 cost = :F-cost.option,
 qty_on_hand = :F-qty_on_hand.option
WHERE CURRENT OF option_key_sel;
```



This SQL logic block updates the row of the table defined by the present position of the *option\_key\_sel* cursor. The *option\_key\_read* function declares and opens this cursor when it calls the SQL logic block *option\_key\_sel*. Note that the cursor name is the name of the SQL logic block containing the SELECT command that declares and opens the cursor.

---

## Deleting Records

The rules for deleting records in this part of the application are the same as the rules in the earlier KSAM application. They are:

- For the current product, an option other than option number 000 can be deleted at any time.
- Option number 000 for any product cannot be deleted.
- A product cannot be deleted if there are any options other than option number 000.
- When a product is deleted, option number 000 is also deleted.

The following logic blocks show one way of achieving this result.

### Deleting a Product

#### Function – *delete\_product*

The *delete\_product* function is a new function.

It checks for the existence of option records other than option number 000 for the current product. If they exist, it displays a message requesting the user to confirm their deletion. The function then requests the user to confirm the deletion of the product. If the user chooses to delete the product, the function deletes the record for option 000 and deletes the product record.

The function is called from the *prod\_opt\_update* decision table.

This function calls three SQL logic blocks. They are *opt\_sel*, *opt\_del*, and *prod\_del*. The operation of the function and these SQL logic blocks is explained below.

This function also requires a new message, *all\_opt\_deleted*.

Create and generate the function and the SQL logic blocks now. You must also create the new message.

```

1 IF V-product_status = C-no-record THEN
 MESSAGE no_product_warn ; EXIT
2 SQL opt_sel ; ENTER 15
3 FILE *NEXT option ; ENTER 14
4 MESSAGE delete_options
5 IF V-confirm <> "Y" THEN ENTER 16
6 SQL opt_del ; ENTER 15
7 MESSAGE all_opt_deleted
8 MESSAGE delete_product
9 IF V-confirm <> "Y" THEN ENTER 16
10 SQL prod_del ; ENTER 15
11 MESSAGE prod_delete_ok
12 MOVE C-no-record V-product_status
13 EXIT
14 IF *IOSTATUS = N-beg_end_of_file THEN ENTER 8
15 MESSAGE file_error
16 MESSAGE prod_delete_fail
17 EXIT

```

Before deleting a product record, this function checks that a product record is current. The function also checks for the existence of option records associated with the product. If options do exist for the product, the user is requested to confirm their deletion.

If the user confirms the deletion of the options, the function deletes the options, and then displays a message requesting the user to confirm deletion of the base product.

### SQL Logic Block – *opt\_sel*

The SQL command in step 2 executes the SQL logic block *opt\_sel*. This SQL logic block contains the command:

```

SELECT :option FROM sqlgrp.option
 WHERE product_no = :F-product_no.product
 AND option_no > :N-zero;

```

This SQL logic block declares and opens a cursor on the *option* table. The active set for this cursor is the set of records for the current product that have an option number that is strictly greater than zero. Note that this select command does not find option number zero for a product. If a product has no options other than the base option (number 000), the active set for this cursor is empty.

The FILE \*NEXT command in step 3 of the function retrieves the first record for the cursor. If this command is successful, steps 4 to 7 of the function display a message, and then delete the options if the user enters “Y” in response to the message.

If the FILE \*NEXT command encounters an end-of-file condition without retrieving a record, the SELECT command has not found any option records for the product. In this case, control passes to step 14 and then to step 8, to allow the user to delete the product. If the file error is due to any condition other than encountering the end of file, control passes to step 15.

### SQL Logic Block – *opt-del*

The SQL logic block called by step 6 of the function deletes records from the *option* table. The *opt-del* SQL logic block contains the following command:

```
DELETE FROM sqlgrp.option
 WHERE product_no = :F-product_no.product
 AND option_no > 000;
```

This SQL block demonstrates the technique for deleting records without using a cursor. In this case, the SQL DELETE command uses an explicit search condition to specify the rows to be deleted.

### Message – *all-opt-deleted*

Step 7 of the *delete\_product* function calls the message *all-opt-deleted*. Create and generate this message now.

|          |                                                                                                           |
|----------|-----------------------------------------------------------------------------------------------------------|
| Name     | <b>all_opt_deleted</b>                                                                                    |
| Type     | <b>MESS</b>                                                                                               |
| Contents | <b>"Number of options deleted for product "</b><br><b>F-product_no.product</b><br><b>" is " *ROWCOUNT</b> |

This message introduces a new communication area field. After any operation that modifies a table, HP ALLBASE/SQL returns a value to \*ROWCOUNT to indicate the number of rows that are affected by the command. In this example, the value in \*ROWCOUNT is the number of rows deleted from the *option* table.

Step 10 of the *delete\_product* function calls the SQL logic block *prod-del*. This SQL logic block deletes the record for option number 000 from the *option* table, and then deletes the product record from the *product* table.

**SQL Logic Block *prod\_del***

The *prod\_del* SQL logic block contains the following commands:

```
DELETE FROM sqlgrp.option
 WHERE product_no = :F-product_no.product;
```

```
DELETE FROM sqlgrp.product
 WHERE product_no = :F-product_no.product;
```

This SQL logic block demonstrates the use of more than one command in an SQL logic block. An SQL logic block can contain up to eight SQL commands if it does not contain a SELECT command. Note that you must terminate each command with a semicolon.

**Deleting Options**

The technique for deleting option records is much simpler than the method for deleting a product.

**Function – *delete\_option***

The *delete\_option* function is called by the *prod\_opt\_update* decision table. This is one of the functions you copied from the earlier application.

Modify step 4 of the existing *delete\_option* function and add steps 8 and 9 so it reads as follows:

```
1 IF V-option_status = C-no-record THEN
 MESSAGE no_option_warn ; EXIT
2 MESSAGE delete_option
3 IF V-confirm <> "Y" THEN MESSAGE opt_delete_fail ; EXIT
4 FILE *DELETE option ; ENTER 8
5 MESSAGE opt_delete_ok
6 MOVE C-no-record V-option_status
7 EXIT
8 MESSAGE file_error
9 EXIT
```

This function uses a FILE \*DELETE command to delete a record from the *option* table. This command is equivalent to the SQL command:

```
DELETE FROM sqlgrp.option
 WHERE CURRENT OF option_key_sel;
```

The cursor is declared and opened by the SQL logic block *option\_key\_sel* called by the function *option\_key\_read*. Note that the cursor name is the same as the name of the SQL logic block that declares and opens the cursor.

---

## Summary

In this lesson you have used some further SQL logic blocks to access and manipulate SQL data. You have also used the \*ROWCOUNT communication area field in a user message.

## SQL Logic Blocks

In the SQL logic blocks, you have used the SELECT command to declare and open cursors for updating tables. The *opt\_sel* SQL logic block contains a SELECT command that opens a cursor for data retrieval only. It does not contain the FOR UPDATE OF clause. The SELECT commands in the *option\_key\_sel* and *opt\_sel* SQL logic blocks contain compound search conditions to specify the active set for the cursor.

In this lesson, you also used the DELETE command in an SQL logic block. The *opt\_del* and *prod\_del* SQL logic blocks both use DELETE commands that contain a WHERE clause to specify the rows to be deleted.

## \*ROWCOUNT

After any operation that changes a table, HP ALLBASE/SQL returns a number indicating the number of rows that are affected to the \*ROWCOUNT communication area field. You can include this communication area field in a message to tell the user how many rows are changed by an add, delete, or modify operation on a base table.

---

---

## Lesson 33 – SQL Select Lists

---

---

### Objectives

In this lesson, you will use an SQL select list as part of a system for scrolling data on the screen.

In the earlier KSAM-based application, you used a secondary record layout for the option data file as a means of reading successive records from the option file, while still holding a record in the default record buffer for the file. This allows the user to retrieve an option record, and simultaneously use the **Scroll Options** function key to review other options for the same product.

HP ALLBASE/SQL tables do not permit multiple record layouts, so the same scrolling system cannot be used. However, you can use a select list to perform the same function as a secondary record layout for a file or table. Using a select list allows you to open a second cursor to retrieve option records without affecting any currently open cursors.

---

### Select Lists

You can think of a select list as being a “virtual” file. A select list consists of columns, just like an HP ALLBASE/SQL table. The columns of a select list can be derived from one or more base tables or views. The values in a select list column can be taken directly from a column in an existing table or view, or can be values derived in any of the following ways:

- Values computed from a base table column using an arithmetic expression.
- Values computed from the values in various base table columns using an arithmetic expression.
- Values computed from a group of base table column values with an aggregate function (AVG, MAX, MIN, SUM, and COUNT).
- Constants, or values computed from an expression involving constants.

---

## Select List Definition

Defining a select list is similar to defining many other items. You must define a select list header, then complete the details for the select list, and finally generate the select list.

Each entry in a select list definition has the following format:

*field\_specification* [= *column\_definition*]

The *field\_specification* must be the name of a dictionary field specification. The term *column\_definition* means any valid SQL column definition. You can use SQL aggregate functions and expressions in the column definitions for SQL select lists. Note that you must terminate each column definition except the last with a comma (,). Each column definition does not necessarily need to be on a separate line of the screen. You can put more than one column definition on the same line.

To abbreviate the entry of select lists, you can omit the right hand side of each entry if the column definition is exactly the same as the field specification name, and no ambiguity results from the omission.

This part of the application uses a select list called *optscrol*. It uses the *option-no*, *description*, *cost*, and *qty\_on\_hand* fields from the option table. These fields contain the information necessary to scroll the option details on the screen.

The screens you use to define select lists are accessible from the database items menu in the dictionary menu. Go to the *Select List Header* screen and define a header for the *optscrol* select list now. This screen is similar in appearance and operation to the other header screens.

When you have defined the select list header, go to the *Select List Details* screen to define the details for the select list. This screen contains a free-format data entry area that allows you to enter the details for a select list.

| Developer           | SQL Select List Details                              | SQLsel_details                         |
|---------------------|------------------------------------------------------|----------------------------------------|
| Select List         | optscrol                                             | Secured <input type="checkbox"/> (Y/N) |
| Select List Details | <pre>option_no, description, cost, qty_on_hand</pre> |                                        |
| Field Specs.        | SQL Sel Header                                       | SQL BIK Menu                           |
| Generate List       | 3 15                                                 | System Keys                            |
|                     | Commit Data                                          | Help                                   |
|                     |                                                      | Previous Menu                          |

### Select List *optscrol*

This is the select list you are about to define:

```
option_no,
description,
cost,
qty_on_hand
```

This select list is equivalent to the following expanded expression.

```
option_no = option_no,
description = description,
cost = cost,
qty_on_hand = qty_on_hand
```

In this example, the column definitions for the select list are exactly the same as the field specification names, so you can omit the right hand side of each item.

Use the select list details screen to complete this select list definition now.



## Generating a Select List

You must generate a select list to convert the definition to an executable run-time format.

When you have completed the select list details, press the **Generate List** function key to generate the select list. During generation, HP ALLBASE/4GL uses the field specification names on the left hand side of each select list column definition to create and generate a record layout for the select list.

Since generating a select list builds a record layout, a select list cannot have the same name as an existing file or base table.

## Scrolling Data on the Screen

The scrolling system for this application is similar to the scrolling system you used in the earlier KSAM-based application. It uses three functions. The functions are *scroll\_options*, *get\_1st\_option*, and *get\_next\_option*. The function *get\_1st\_option* calls an SQL logic block *optscrol* to declare and open a cursor on the *optscrol* select list.

### Function – *scroll\_options*

This is one of the functions you copied from the earlier application. Modify the DEFINE command at step 1 so the function reads as follows:

```

1 DEFINE %FILE% optscrol
2 IF V-product_status <> C-record THEN
 MESSAGE no-product ; EXIT
3 IF 1 *ON THEN VISIT get_1st_option ; ENTER 7
 ELSE VISIT get_next_option
4 IF 1 *OFF THEN ENTER 7
5 SCROLL 9 "***** End of options for product #
 " S-product_no.product " *****"
6 EXIT
7 SCROLL 3 F-option_no.%FILE% 4 F-description.%FILE%
 5 F-cost.%FILE% 8 F-qty_on_hand.%FILE%
8 EXIT

```

The method of operation of this function is exactly the same as its counterpart in the KSAM-based application.

This function demonstrates the technique for referencing fields on select list buffers. Step 7 contains references that are expressed in the format *F-field\_name.%FILE%*.

When expanded, these references are equivalent to *F-field\_name.optscrol*. That is, references to select list fields use exactly the same format as references to file record fields.

### Function *get\_1st\_option*

This is also one of the functions you copied from the earlier application. Modify step 1 and steps 3, 4, and 5 so the function reads as follows.

```

1 DEFINE %FILE% optscrol
2 DISPLAY *RESET=S ""
3 SQL optscrol
4 FILE *NEXT %FILE% ; ENTER 9
5 IF F-option-no.%FILE% <> N-zero THEN ENTER 9
6 SCROLL 9 "***** Start of options for product #
 " S-product_no.product " *****"
7 OFF 1
8 EXIT
9 MESSAGE prod_corrupt
10 PROCEED prod_opt ; NOTE Product/option records corrupted
 since option 000 does not exist. Clear buffers and display
 prod/opt menu so user can delete the product.
```

This function calls the SQL logic block *optscrol*. This SQL logic block contains a *SELECT* command to retrieve the options for the current product.

### SQL Logic Block – *optscrol*

The *optscrol* SQL logic block contains the following command:

```

SELECT :optscrol FROM sqlgrp.option
 WHERE product_no = :F-product_no.product
 ORDER BY option_no;
```

This SQL logic block demonstrates the use of a select list in a *SELECT* command. Note that you must precede the select list name with a colon(:). This *SELECT* command also includes an *ORDER BY* clause to retrieve the options for a product in option number order.

This select command declares and opens a cursor on the *optscrol* select list. The active set for this cursor consists of all the option records for the current product. The records are ordered by option number. The *FILE \*NEXT* command in the function *get\_1st\_option* positions the cursor on the first record in the active set, and retrieves the record into the select list buffer.

**Function – *get\_next\_option***

The *get\_next\_option* function is called by the *scroll\_options* function to retrieve the second and subsequent option records for a product.

The *get\_next\_option* function contains the following commands. Create and generate this function now.

```
1 FILE *NEXT optscrol ; ENTER 3
2 EXIT
3 ON 1
4 EXIT
```

This function reads the next record for the active set for the *optscrol* select list. If a record is retrieved successfully the function exits. If a record is not found (indicating an end-of-file condition and no further option records exist) the function sets switch number 1 on, and then exits. When set on, switch number 1 resets the scrolling system.

When you defined the *product* process, you added a step to set switch number 1 on after the *prod\_opt\_update* decision table exits and the current transaction has been committed. This is necessary to force execution of the function *get\_1st\_option* if the user has pressed the **Scroll Options** function key after pressing the **Commit Data** function key.

Executing the function *get\_1st\_option* declares and opens a cursor on the *optscrol* select list. This is necessary because the COMMIT WORK command called by the *product* process closes all open cursors. Attempting to execute the function *get\_next\_option* without first declaring and opening a cursor generates an error condition.

---

## Running the Application

You should now be able to test the option scrolling system. Make sure that you have generated the new components before you test the application.

---

## Summary

In this lesson, you used a select list to retrieve data from an HP ALLBASE/SQL base table. Using a select list allows you to declare and open an additional cursor without changing any existing cursors.

In this application, the *optscrol* select list allows you to retrieve information from option records for a product, without changing the existing cursor used to retrieve the current option record that may be displayed on the option window.

Select lists have the following characteristics:

- A select list is a “virtual” file. You can reference fields on a select list as though they are file record fields.
- A select list definition consists of the select list header, and the select list details.
- Each entry in a select list has the format:  
*field\_spec [= column\_definition],*

Columns can be derived directly from columns of existing tables or views, or can be derived using SQL expressions.

To simplify the procedure for defining a select list, you can omit the right hand side of each entry if the column definition is exactly the same as the field specification name.

- A select list must be generated. When HP ALLBASE/4GL generates a select list, it creates a generated record layout using the field specifications on the left hand side of the select list entries.

## Lesson 34 – Further HP ALLBASE/SQL Reporting Techniques

---

---

### Objectives

In this lesson you will develop a report that accesses information in two different SQL tables. In developing this report, you will see how the data selection and sorting capabilities of HP ALLBASE/SQL can be used in conjunction with HP ALLBASE/4GL to format and print a report.

---

### Preparations

Start by copying the following items from the KSAM-based *training* application.

- **Functions**
  - `select_products`
- **Screens**
  - `select_products`
- **Calculated Item**
  - `option_cost`

### Some New Items

#### Field Specifications

This report also requires an additional field specification *opt\_descript*. This field specification is identical to the existing *description* field specification. Use the copying utility to create it now.

#### Variables

This report requires two variables called *first\_product* and *last\_product*. Both variables have a length of six characters and have a type U edit code. Create these variables now.

---

## The Product Report

This report serves exactly the same purpose as the *product* report in the earlier KSAM-based application. However, the technique for deriving the report is a little different.

This report uses an HP ALLBASE/SQL select list as the primary report file. The select list contains columns from two SQL tables. (The *product* table and the *option* table.) While you develop this report, you will see how you can use the selection and sorting facilities provided by HP ALLBASE/SQL in conjunction with HP ALLBASE/4GL.

### Retrieving Data for Reporting

In exactly the same way as the earlier *customer* report, this report uses a start-of-report function to display a screen to allow the user to specify the starting and finishing product numbers. For this report, the start-of-report function also calls an SQL logic block to declare and open a select list cursor. This cursor is the primary file for the report.

#### Select List – *prod\_opt*

Create and generate the *prod\_opt* select list now. The *prod\_opt* select list is as follows:

```
product_no = product.product_no,
description = product.description,
supplier_no,
lead_time,
option_no,
opt_descript = option.description,
cost,
qty_on_hand
```

This select list uses the new field specification *opt\_descript*. This is necessary to avoid duplicate field specification names in the select list.

This select list requires you to enter the right hand side of the *product\_no*, *description*, and *opt\_descript* columns. These entries are required to define the columns completely to avoid ambiguity.

This select list also demonstrates the use of qualified names in the column definitions. The column definitions for the *product\_no* column and both *description*

columns contain the name of the table as a qualifier. This qualifier is necessary to avoid ambiguous column definitions. The terms on the right hand side of these entries use the standard SQL syntax for referencing table columns. That is, you must reference a column as *table\_name.column\_name*.

### Function – *select\_products*

The *select\_products* function is one that you have copied from the earlier application. Modify steps 5 and 6, and insert step 7 so the function reads as shown below. Then generate the function.

```

1 SCREEN select_products
2 IF *S01 >= *S02 THEN MOVELOOP 2 *S01 1 *P02 -1
 ELSE ENTER 5
3 MOVELOOP 2 *P01 1 *S01 1
4 SHOW
5 MOVE *S01 V-first_product
6 MOVE *S02 V-last_product
7 SQL prod_opt
8 EXIT

```

This function performs the same checking and correction on the product numbers entered by the user as its counterpart in the earlier application. It then moves the first and last product numbers for the report to the variables *first\_product* and *last\_product*. The function then calls the SQL logic block *prod\_opt* to declare and open a cursor to form the primary file for the report.

Note that this function does **not** contain a FILE \*NEXT command. The report generator issues the equivalent of a FILE \*FIRST command to retrieve the first record in the active set for the cursor.

### SQL Logic Block – *prod\_opt*

The *prod\_opt* SQL logic block contains the following SELECT command. Create and generate this SQL logic block now.

```

SELECT :prod_opt FROM product, option
 WHERE product.product_no = option.product_no
 AND product.product_no >= :V-first_product
 AND product.product_no <= :V-last_product
ORDER BY product_no, option_no;

```

To join two tables, HP ALLBASE/SQL requires that they have at least one common column. In this case, both the *product* table and the *option* table contain a *product\_no* column.

This **SELECT** command declares and opens a cursor on the *prod\_opt* select list. The active set for this cursor contains all the records where the value in the *product\_no* column is equal to or greater than the value in the variable *first\_product* and equal to or less than the value in the variable *last\_product*. For each product number in the specified range of product numbers, this select command retrieves the corresponding product records, and all option records for these products.

The select command also contains an **ORDER BY** clause to specify that the select list records are sorted in product number order, and within each product number, records are sorted by option number.

## Report Header

Use the report header screen to define a header for the *product* report.

The report has the following details.

|                               |                        |
|-------------------------------|------------------------|
| Report Name                   | <b>product</b>         |
| Report File Designator        | <b>prod</b>            |
| Primary File[.Record]         | <b>prod_opt</b>        |
| Characters per line           | 100                    |
| Start of Report Function Name | <b>select_products</b> |

Accept the default entries for the other fields on the report header.

## Report Sorting

Use the report sorting screen to specify the following details for the *product* report.

|                              |                            |
|------------------------------|----------------------------|
| 1st Sort level               | <b>product_no</b>          |
| Order                        | <b>A</b> (ascending order) |
| Is File already in sequence? | <b>Y</b> (yes)             |

Specifying the *product\_no* field as a sort field is required to generate control breaks to print subheading and subtotal lines on the report. You will see how these are used when you print the report.

## Record Selection and File Linkages

This report does not require any record selection criteria or file linkage definitions.

The record selection and data retrieval for the report is performed by the **SELECT** command in the *prod\_opt* SQL logic block.



### Line Headers

Use the report line header screen to define headers for the following lines:

| Line Group | Line Number | Skip Lines Before Print | Skip Lines After Print |
|------------|-------------|-------------------------|------------------------|
| P1         | 01          | 0                       | 1                      |
|            | 02          | 0                       | 1                      |
| H1         | 01          | 2                       | 1                      |
|            | 02          | 1                       | 1                      |
|            | 03          | 1                       | 2                      |
| D1         | 01          | 0                       | 1                      |
| T1         | 01          | 2                       | 1                      |
| TF         | 01          | 3                       | 1                      |

Accept the defaults for any values not shown here. This report does not require any before print or after print line functions.

### Painting the Report

You can now use the report painter to paint this report. The following report layout shows the print lines and the fields for this report. Where possible, use the dictionary field specifications to create the report line fields.

|       |                                                                                                      |                              |   |   |                                  |                  |                     |   |   |            |
|-------|------------------------------------------------------------------------------------------------------|------------------------------|---|---|----------------------------------|------------------|---------------------|---|---|------------|
|       | 1                                                                                                    | 2                            | 3 | 4 | 5                                | 6                | 7                   | 8 | 9 | 10         |
|       | 1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890 |                              |   |   |                                  |                  |                     |   |   |            |
| P1.01 |                                                                                                      |                              |   |   |                                  |                  |                     |   |   | Page: NNNN |
| P1.02 |                                                                                                      |                              |   |   |                                  |                  |                     |   |   |            |
| H1.01 | Product #                                                                                            | Description                  |   |   | Supplier                         | Lead Time        |                     |   |   |            |
| H1.02 | AAAAAA                                                                                               | AAAAAAAAAAAAAAAAAAAAAAAAAAAA |   |   | NNNNNN                           | NN               |                     |   |   |            |
| H1.03 | Option #                                                                                             | Description                  |   |   | Cost per Unit                    | Qty on Hand      | Cost of Stock       |   |   |            |
| D1.01 | NNN                                                                                                  | AAAAAAAAAAAAAAAAAAAAAAAAAAAA |   |   | \$NNN,NNN,NNN.NN                 | NNNNN            | \$NN,NNN,NNN,NNN.NN |   |   |            |
| T1.01 |                                                                                                      |                              |   |   | Total cost of Stock for Product  | AAAAAA           | \$NN,NNN,NNN,NNN.NN |   |   |            |
| TF.01 |                                                                                                      |                              |   |   | Total Cost of Stock for Products | AAAAAA to AAAAAA | \$NN,NNN,NNN,NNN.NN |   |   |            |

The following table summarizes the details for the fields on each report line. The field numbers refer to the number of the field from the left end of the report line, and the starting position is the cursor position at the beginning of the field.

Use these details as a guide when you are painting the report lines.

| Line  | Field | Start Pos. | Field Details                          |
|-------|-------|------------|----------------------------------------|
| P1.01 | 1     | 28         | literal                                |
|       | 2     | 68         | *TIME[1,5]                             |
|       | 3     | 91         | literal                                |
|       | 4     | 97         | *PAGENO                                |
| P1.02 | 1     | 60         | literal                                |
|       | 2     | 68         | *DATE (D edit code)                    |
| H1.01 | 1     | 01         | literal                                |
|       | 2     | 24         | literal                                |
|       | 3     | 57         | literal                                |
|       | 4     | 69         | literal                                |
| H1.02 | 1     | 02         | F-product_no.prod_opt                  |
|       | 2     | 15         | F-description.prod_opt                 |
|       | 3     | 58         | F-supplier_no.prod_opt                 |
|       | 4     | 73         | F-lead-time.prod_opt                   |
| H1.03 | 1     | 05         | literal                                |
|       | 2     | 24         | literal                                |
|       | 3     | 50         | literal                                |
|       | 4     | 68         | literal                                |
|       | 5     | 85         | literal                                |
| D1.01 | 1     | 07         | F-option_no.prod_opt                   |
|       | 2     | 15         | F-opt_descript.prod_opt                |
|       | 3     | 49         | F-cost.prod_opt                        |
|       | 4     | 70         | F-qty_on_hand.prod_opt                 |
|       | 5     | 83         | U-option_cost<br>Total into *TOTALS(1) |
| T1.01 | 1     | 42         | literal                                |
|       | 2     | 74         | F-product_no.prod_opt                  |
|       | 3     | 83         | *TOTALS(1)                             |

|       |   |    |                               |
|-------|---|----|-------------------------------|
| TF.01 | 1 | 31 | literal                       |
|       | 2 | 64 | V-first_product (U edit code) |
|       | 3 | 71 | literal                       |
|       | 4 | 74 | V-last_product (U edit code)  |
|       | 5 | 83 | *TOTALS(1)                    |

This report uses subheading lines (the H1.01, H1.02, and H1.03 lines) at control breaks to print the details of each product.

The primary file for this record is the active set for the cursor defined by the SELECT command in the *prod\_opt* SQL logic block. A series of typical records in this active set has the following format. The details of the structure of the “product” and “option” portion of each record are determined by the field specifications you used to define the *prod\_opt* select list.

|                  |              |
|------------------|--------------|
| product # XY1000 | option # 000 |
| product # XY1000 | option # 001 |
| ...              | option # 003 |

...

...

|                  |              |
|------------------|--------------|
| product # XY1002 | option # 000 |
| product # XY1002 | option # 001 |
| ...              | option # 002 |

...

...

|                  |              |
|------------------|--------------|
| product # XYnnnn | option # 000 |
| product # XYnnnn | option # 001 |

...

...

The primary file for the report contains one record for each option corresponding to the selected range of products. The report uses a type D1.01 detail line to print the details for each option. The D1.01 line includes the details in the “option” part of each record.

Each record in the report select list also contains the details of the product corresponding to the option.

Since the *product\_no* field is defined as the first sorting field on the report sorting screen, a control break occurs each time the product number changes from one record to the next. At a control break, the report generator prints the subtotal line (T1 line) for the old value of the product number, and then prints subheading lines for the new value of the product number. In this case, the H1 subheadings contain the product headings, the details for the product, and subheadings for the option details. These lines are only printed once for each product, and are printed immediately after the product number changes from one record to the next.

## Completing the Report

### Calculated Item – *option\_cost*

Before you can generate the report, you must edit the definition of the calculated item *option\_cost*. Use the calculated items screen to call up the *option\_cost* calculated item, and amend the references for the *cost* and *qty\_on\_hand* fields so they refer to the select list *prod\_opt*. The CALC command should read as follows:

```
CALC F-cost.prod_opt × F-qty_on_hand.prod_opt = U-option_cost
```

Pressing the **Commit Data** function key generates the calculated item automatically.

## Generating the Report

The definition for the *product* report is now complete. Call up the report on the report header screen, and press the **Generate Report** function key to generate the report.

You must also generate the *select\_products* screen before you can run the application.

You should now be able to test the report by running the application using application testing mode.

---

## Summary

In this report, you used a select list to retrieve records from two HP ALLBASE/SQL tables. The SELECT command that declares and opens the select list cursor performs all the record selection and sorting for the report.

By using sort level definitions in the report generator, you have been able to use control breaks to print subheading and subtotal lines for each product listed in the report.

# Contents

---

## Appendix A: Starting a New Application

|                                           |       |
|-------------------------------------------|-------|
| The Sign-On Screen . . . . .              | A - 1 |
| Administrator Sign-On . . . . .           | A - 2 |
| Administrator Main Menu . . . . .         | A - 2 |
| Developer Validation . . . . .            | A - 2 |
| Training Application Definition . . . . . | A - 4 |
| Application Password Definition . . . . . | A - 5 |
| Development Security Code . . . . .       | A - 6 |
| Users and Groups List . . . . .           | A - 7 |
| SQL Application Definition . . . . .      | A - 7 |
| SQL Database Definition . . . . .         | A - 8 |
| Database Access . . . . .                 | A - 9 |



**Intentionally Blank**

# A Starting a New Application

---

This appendix describes the procedures for creating an application definition in the HP ALLBASE/4GL *administ* application. A developer cannot develop an application until it has been defined in the *administ* application.

You may need to ask your system administrator to define the application if you do not have access to the administrator application.

---

## The Sign-On Screen

The sign-on screen is the first screen that all users will see after starting up HP ALLBASE/4GL.

```
HP ALLBASE/4GL B.01.00 HP30601 Copyright(c) 1986-89 Hewlett-Packard Aust. Ltd.

HP ALLBASE/
4GL

User Name User Password

Start Again 17 18 Help Commit Data Exit
```

## Administrator Sign-On

Enter the administrator user name **administ**, and then press **Return**.

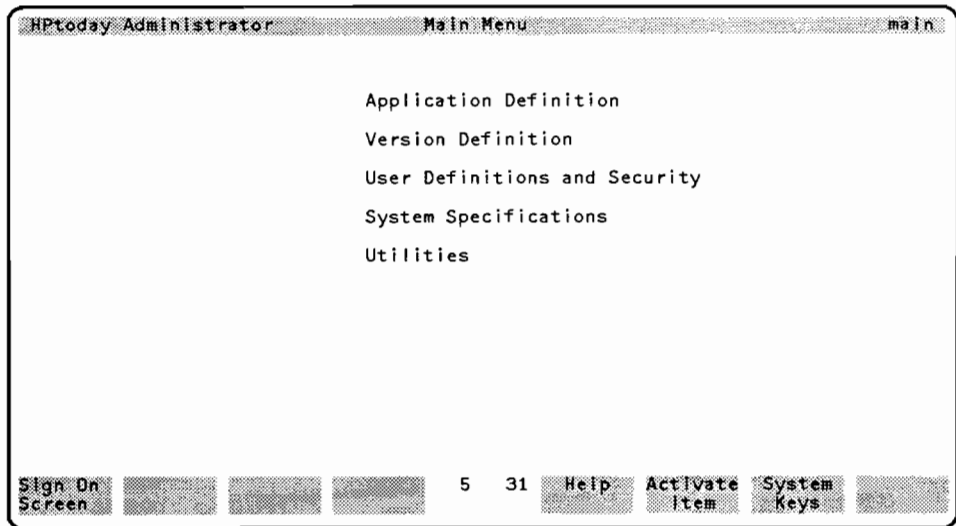
HP ALLBASE/4GL checks that you have entered the correct administrator user name. If a password is required, you will be prompted to enter it. The password does not appear on the screen as you enter it.

Press the **Commit Data** function key to run the administrator application.

---

## Administrator Main Menu

When the administrator application starts, the administrator main menu is displayed.



## Developer Validation

Before you can use the developer to develop an application, you must be registered in the HP ALLBASE/4GL administrator as a developer user. The screens used to define HP ALLBASE/4GL users are accessed from the user validation submenu of the administrator main menu. To display the developer validation screen, select the *User Definitions and Security* option from the main menu. This displays the user validation menu. Select the *Developer Validation* option on this menu to display the developer validation screen.



You may also want to assign yourself an end user name. You must be registered as an end user in the administrator if you want to run your applications as an end user. To display the *End User Validation* screen, select the *End User Validation* option from the user validation menu. Refer to the *HP ALLBASE/4GL Developer Administration Manual* if you need more information about this screen.

| Administrator                                                                           | Developer Validation                  | developer     |
|-----------------------------------------------------------------------------------------|---------------------------------------|---------------|
| Developer Name                                                                          | <input type="text" value="username"/> |               |
| Current Password                                                                        | <input type="password"/>              |               |
| New Password                                                                            | <input type="password"/>              |               |
| Repeat New Password                                                                     | <input type="password"/>              |               |
| Description                                                                             |                                       |               |
| username                                                                                |                                       |               |
| Developer name for new developer using ALLBASE/4GL to develop the training application. |                                       |               |
| Last Modification:                                                                      | Date                                  | Time          |
|                                                                                         | 4 31                                  |               |
| End User Validation                                                                     | Menu Security                         | System Keys   |
|                                                                                         |                                       | Commit Data   |
|                                                                                         |                                       | Help          |
|                                                                                         |                                       | Previous Menu |

## Completing the Screen

Before you complete this screen, check the details with your system administrator. The administrator may have established standards for developer names, user names, or passwords at your site.

### Developer Name.

Enter your selected developer name in this field. Developer names can use alphabetic characters, 0 to 9, and \_ (underscore). The name must start with an alphabetic character. HP ALLBASE/4GL is case sensitive with respect to developer names. The names *administ* and *developr* are reserved names, and cannot be used as developer names.

**Current Password.** No entry is possible at this time as this is a new entry. This field is only active if you have already defined a password for this developer name.

**New Password.** This is optional.

If you want to assign a password to this developer name, enter the password in this field. Passwords can be up to eight characters long, and can use alphabetic characters, 0 to 9, and `_` (underscore). The password must start with an alphabetic character. HP ALLBASE/4GL is case sensitive with respect to passwords.

**Repeat New Password.** This is optional.

If you entered a password in the *New Password* field above, you must repeat exactly the same password in this field. If you don't re-enter exactly the same password, HP ALLBASE/4GL generates an error message, and you must re-enter the password in the *New Password* field.

**Description.** Enter a brief description of the developer used for documentation purposes.

Press the **Commit Data** function key to create this developer definition. You can now go on to complete the application definition screen.

---

## Training Application Definition

The following paragraphs describe the procedure for creating the application definition for the *training* application. Refer to "SQL Application Definition" on page A - 7 for details of the *sqltrain* application definition.

The application definition screen allows you to define the essential details about the administration of an application. You must define the application name, password, and the users or user groups. You must also define the name and the type of the first action to be executed when the application starts.

You don't have to strictly adhere to the following suggested entries. They are the minimum requirements. There may be some administrative standards implemented at your site that require you to alter some of the values specified here. The following text draws attention to fields that must **not** be changed.

| Administrator                                                                                                      |              | Application Definition            |             | applic_defn |    |             |             |      |               |
|--------------------------------------------------------------------------------------------------------------------|--------------|-----------------------------------|-------------|-------------|----|-------------|-------------|------|---------------|
| Application                                                                                                        | training     |                                   |             |             |    |             |             |      |               |
| Current Password                                                                                                   |              | Current Development Security Code |             |             |    |             |             |      |               |
| New Password                                                                                                       |              | New Development Security Code     |             |             |    |             |             |      |               |
| Repeat New Password                                                                                                |              | Repeat New Security Code          |             |             |    |             |             |      |               |
| Initial Action Name                                                                                                | main         | Type                              | M (M/P)     |             |    |             |             |      |               |
| SQL owner Group                                                                                                    |              |                                   |             |             |    |             |             |      |               |
| SQL Data Base Name                                                                                                 |              |                                   |             |             |    |             |             |      |               |
| Valid Users/Groups                                                                                                 |              |                                   |             |             |    |             |             |      |               |
| *ALL                                                                                                               |              |                                   |             |             |    |             |             |      |               |
| Description                                                                                                        |              |                                   |             |             |    |             |             |      |               |
| Training application                                                                                               |              |                                   |             |             |    |             |             |      |               |
| This is the training application for new developers who are completing the ALLBASE/4GL self-paced training course. |              |                                   |             |             |    |             |             |      |               |
| Last Modification:      Date      Time                                                                             |              |                                   |             |             |    |             |             |      |               |
| Version Defn.                                                                                                      | Utility Menu | User Menu                         | System Menu | 3           | 21 | System Keys | Commit Data | Help | Previous Menu |

## Entering the Field Values

**Application.** Enter `training`

This is the name that developers and end users will use to access this application.

You don't need to use the name `training`. However, if you do use a different name, ensure that you use that name in place of any reference to the application name `training` throughout the *Developer Self-Paced Training Guide*. Remember that the names `administ` and `developr` are reserved names, and cannot be used as application names.

### Application Password Definition

The next three fields allow you to define a user password for this application. If you do define a password, any end user wanting to use it must enter the password before gaining access to the application.

**Current Password.** No entry is possible in this field as no password exists for a new application.

This field is bypassed as no password is set when the application is first defined. When the application does have a password you must enter it in this field before you can change the existing password.

**New Password.** Leave this field blank.

If you want to enter a password for this application then you can enter a password in this field. Passwords can use alphabetic characters, 0 to 9, and \_ (underscore). The first character of a password must be alphabetic. As with all other names, HP ALLBASE/4GL is case sensitive with respect to passwords.

**Repeat New Password.** Leave this field blank.

If you want to enter a password for this application then you should re-enter the password in this field. If the two password entries don't match, you must re-enter the new password in the *New Password* field.

## Development Security Code

The next three fields allow you to define a development security code for the application. If you do define a security code, any developer wishing to change secured components in the application must enter the correct security code when signing on to the application.

**Current Development Security Code.** No entry is possible in this field.

This field is only active if a security code already exists for the application.

**New Development Security Code.** Leave this field blank.

This is where you can enter a security code for this application. If you do enter a code it is not echoed to the screen.

**Repeat New Security Code.** Leave this field blank.

If you enter a new security code you must repeat it in this field. If the two entries don't match you must re-enter the *New Development Security Code*.

**Initial Action Name.** Enter **main**

This field defines the name of the first item in the application that HP ALLBASE/4GL executes when the application starts. You must enter **main** for this application.

**Type.** Enter **M**

This field specifies the action type for the initial action. It can be either a **menu** or a **process**.

**SQL Owner Group.** Leave blank.

**SQL Data Base Name.** Leave blank.

### Users and Groups List

The next 12 fields list the valid user names or groups who can access particular application. Only the developers and end users listed here can access this application.

**Valid Users/Groups.** Enter \*ALL.

\*ALL is a special entry specifying that anyone gaining access to HP ALLBASE/4GL may access this application. Your system administrator may require you to enter specific user names in these fields.

Leave the remaining Valid Users/Groups fields blank.

**Description.** Enter a suitable description of the application:

```
Training application
This is the training application for ...
```

This is a brief description of the application that is used for documentation purposes.

### When the Entries are Complete

Press the **Commit Data** function key to complete this screen. Now that you have created the application definition, you can proceed with developing the application.

Return to the main menu by pressing the **Previous Menu** function key, and then press the **Sign on Screen** function key to return to the sign-on screen.

---

## SQL Application Definition

The following paragraphs describe the procedure to creating the application definition for the *sqltrain* application.

The procedure to create the application definition for the SQL based application is essentially the same as the procedure for defining the earlier *training* application.

Call up the application definition screen in the administrator, and create an application definition for the application *sqltrain*. For this application, you must complete two additional fields on the application definition screen. These are the *SQL Owner Group* field and the *SQL Data Base Name* field. The entries shown below are the suggested entries for these fields.

**SQL Owner Group.** Enter sqlgrp

HP ALLBASE/4GL allows application developers to create base tables in the application database environment. HP ALLBASE/4GL also creates a database module when an application is generated. HP ALLBASE/SQL requires that base tables and modules have an owner.

HP ALLBASE/4GL transfers ownership of all base tables and the module for this application to the owner group specified in this field.

You can enter a different name in this field if required. However, if you do use a different owner group name, all references to the owner group *sqlgrp* in this manual must be replaced with the correct owner group name.

---

### Note

You cannot change the SQL owner group name for an application after development of the application has commenced.

---

**SQL Data Base Name.** Enter *sqldbbase*

This is the name of the database environment for the application. The name of a database environment is the name of the DBEConfile for the database.

HP ALLBASE/4GL uses a MPE XL variable HP4SQLPATH to identify the account and group containing the database DBEConfile. Refer to the *HP ALLBASE/4GL Developer Administration Manual* for more information about the MPE XL variables used by HP ALLBASE/4GL.

Once again, you can use a different name for the database environment provided you substitute the new name for all references to *sqldbbase* in this guide. You cannot change the database environment name after application development has commenced.

## SQL Database Definition

HP ALLBASE/4GL does not create database environments directly. You must create the database environment using ISQL before application development can commence.

The database environment for an application must contain at least one DBEfileset and one DBEfile.

The commands listed below show the procedure for creating a database environment that is suitable for the *sqltrain* application. This database is the smallest

workable size for developing this application. You may want to enlarge or modify the database if you have sufficient free disk space.

To create the database environment, you must start the ISQL program in the account and group for the database environment DBEConfile. Typically, the command to start HP ISQL is simply ISQL.

Enter the following commands at the ISQL prompt.

```
isql=> START DBE 'sqlbase' MULTI NEW
> TRANSACTION = 1, 6
> DBEFILE0 DBEFILE sqlDBEO
> WITH PAGES = 50, NAME = 'sqlFO',
> LOG DBEFILE sqlDBElog1
> WITH PAGES = 100, 500
> NAME = 'sqllog1';
```

```
isql=> CREATE DBEFILESET CUSTFS;
isql=> CREATE DBEFILESET PRODFS;
```

```
isql=> CREATE DBEFILE custDBEfile WITH PAGES = 10, 50
> NAME = 'custfile', TYPE = MIXED;
```

```
isql=> CREATE DBEFILE prodDBEfile WITH PAGES = 30,
> NAME = 'prodfile', TYPE = MIXED;
```

```
isql=> ADD DBEFILE custDBEfile TO DBEFILESET CUSTFS;
isql=> ADD DBEFILE prodDBEfile TO DBEFILESET PRODFS;
isql=> commit work;
isql=> exit;
```

In this dialogue, user input is shown underlined for clarity. The characters `isql=>` represent the normal ISQL prompt, and the character `>` is the ISQL continuation prompt that appears when a command extends beyond one line.

Note that you must terminate each command with a semicolon (;).

### Database Access

The MPE XL user who creates a database environment automatically becomes the database administrator. This user must grant appropriate access authority to other users.

Application developers must have **CONNECT** authority and **RESOURCE** authority for the application database environment. Application developers must also be members of the **SQL** owner group for the application.



# Glossary

---

This glossary explains the meaning of some of the terms and words used with HP ALLBASE/4GL. In some cases, the terms used in HP ALLBASE/4GL may differ in meaning from the same terms used in a conventional programming environment.

## A

**Abort.** A message type that informs the user that a serious error has been detected. The HP ALLBASE/4GL session terminates after the user acknowledges the message.

**\*ABSENT.** VALIDATE command argument that is used to check if the specified KSAM file record field does not exist.

**Administrator.** The person who controls the HP ALLBASE/4GL site configuration and controls access by developers and users.

**administ.** The name of the administrator application. It is a reserved name.

**administrator.** The application used by the system administrator to control the system-wide site configuration. Note that this name has a lowercase "a".

**\*ALL.** OFF and ON logic command argument for referencing all user switches 1 through 8.

**\*ALPHA.** IF and IFLOOP logic command argument used for checking if a field contains only alphabetic characters.

## B

**BACKGRND.** Logic command for executing a process in background mode. Only an HP ALLBASE/4GL process can be executed in this manner. The process may execute any facility except displaying a screen. A background process cannot send messages to its originating terminal or accept input from the terminal.

**\*BLANK.** IF and IFLOOP command argument used to check if a field contains only space characters.

**\*BUFFER.** FILE command argument used to clear a file record buffer. This command also unlocks a file record that has been locked to the file buffer.

**\*BYPASS.** A switch used by the report generator. When the switch is set on, it indicates that a record is not to be processed.

**C**

**CALC.** Logic command for performing a mathematical operation.

**Calculated Item.** A variable that is evaluated each time it is referenced. In the simple form, the calculated item is the result obtained by evaluating a CALC logic command. In the extended form, HP ALLBASE/4GL executes a function each time the calculated item is referenced. In this case, the calculated item function assigns a value to the calculated item.

**CHECK.** Logic command for matching data against an indexed file record or table. The CHECK command places the matching position number into the communication area field \*PASS.

**CLEAR.** Logic command for clearing all or part of the screen or scratch-pad.

**\*CLOSE.** FILE command argument used to close the nominated file.

**\*COMMIT.** Command argument used with the EXIT command in an after function on a data screen input field, a prior function on a display-only field on a data screen, or a function called from a function key on a data screen. This argument terminates the current screen in the same way as the I-COMMIT internal routine.

**Communication Area Field.** A field addressable by HP ALLBASE/4GL and the logic of the application. Communication area fields may be either alterable or read only. Any non-numeric communication area field may be referenced using the substring conventions. Numeric communication area fields are held as numbers and may not be referenced by substring.

**\*COUNT(n).** Alterable communication area fields used primarily in the report generator for counting the number of times a nominated report line is printed. HP ALLBASE/4GL has five line counter fields, identified as \*COUNT(1) through \*COUNT(5). These fields are treated as numeric.

**\*CROSS(n).** Alterable communication area fields used primarily in the report generator for across-the-page totalling of numeric fields. HP ALLBASE/4GL has five cross add fields, \*CROSS(1) through \*CROSS(5). These fields are treated as numeric.

## D

**Data screen.** The normal HP ALLBASE/4GL screen on which the user can enter data, and on which the system displays information. Data screens can contain a scrolling area or a window or both. The SCREEN logic command or the D- action on a menu initiates display of a data screen.

**\*DATE.** Read-only communication area field containing the current date. This field may be referenced by substring.

**Date Format.** Each HP ALLBASE/4GL site has one date format (specified by the system administrator) that applies to all developers and users. The format is either the US date format MM/DD/YY, or the European date format DD/MM/YY. Regardless of the date format, HP ALLBASE/4GL stores all dates on file records in the format YMMDD. The chosen system-wide data format controls the presentation of dates on screens and reports.

**DECISION.** Logic command for executing a decision table.

**Decision table.** A decision table consists of up to 8 questions, up to 31 different combinations of outcomes for the questions, and up to eight actions. For each combination of outcomes, the decision table specifies which actions are to be performed, and the sequence in which they are performed.

**DEFINE.** Logic command that allows you to define a macro in a logic block. The DEFINE command substitutes an expression for a one to four character string surrounded by % signs.

**\*DELETE.** FILE command argument that specifies file record deletion.

**developr.** The name of the developer application.

**DISPLAY.** Logic command for clearing and displaying data within the scroll area of a data screen.

## E

**Edit code.** A code assigned to a field specification, a screen field, or a report field that defines the type of data stored or displayed in the field.

**\*ENDFIELD.** A switch used by the screen processing logic. A function called from a function key can set this switch *on*. If \*ENDFIELD is *on*, normal input processing for the current field is terminated, and processing continues at the next field on the screen when the function exits.

**\*ENDLINE.** A switch used by the report generator. When this switch is *on*, it indicates that no more components of the current line group are to be printed, or no more records are to be read from the current link file.

**ENTER.** Logic command specifying the step number of the next command step to be executed in the current logic block. The **ENTER** command is equivalent to an unconditional **GOTO** command.

**\*ENTERED.** A switch used by the screen processing logic. This switch is *off* if the user has not entered data into a screen field, and is set *on* after the user enters data.

**Error.** A message type that informs the user of an error condition that must be corrected.

**EXIT.** Logic command that terminates the current logic block.

**EXTERNAL.** Logic command that passes control to an external program written in a language other than HP ALLBASE/4GL.

## F

**FIELD.** Logic command for altering the behavior or attributes of a field on the current screen.

**\*FIELDNO.** An alterable communication area field that contains the current screen field number. It is treated as a numeric.

**Field Specification.** A dictionary entry that defines the characteristics of a field. A field specification defines the editing and data validation characteristics of a field.

**FILE.** Logic command for specifying reading from, writing to, or deleting from a file.

**\*FILENAME.** Communication area field that contains the external name for the current or most recently accessed KSAM file or serial file. This field is also used for dynamic naming of serial files. You can specify the external name for a file as **\*FILENAME**, and then move the appropriate name into **\*FILENAME** to specify the file to be accessed at run time.

**\*FIND.** **FILE** command argument used for locating a record on an KSAM data file. The **FILE \*FIND** command positions the file pointer at the first record with a key value equal to or greater than the value in the key field of the record buffer, or the value specified by the **\*KEY=** argument.

**\*FIRST.** **FILE** command argument used for specifying that the first record on a serial or KSAM file is to be read.

**\*FUNCTION.** Alterable communication area field that contains the name of the function being executed (or blank if not currently executing a function).

**Function.** HP ALLBASE/4GL logic entity. A function is similar to a subroutine in a conventional language system. It consists of between 1 and 30 lines of logic commands.

## G

**Generate.** Validate and transform HP ALLBASE/4GL source parameters to executable format.

## H

**Hash.** The character #. This character is used in the MATH and MATHLOOP commands to specify that a number is to be treated as an absolute number.

## I

**IF.** Logic command for performing conditional tests.

**IFLOOP.** Logic command for repeated conditional testing.

**\*INDEX=.** Command argument for the CHECK, FILE, and VALIDATE commands. This argument specifies the index to be used to access the file for the particular file access. Using \*INDEX= with these commands does not change the current value in \*INDEXNO.

**\*INDEXNO.** Alterable communication area field that contains the number of the file index to be used for subsequent accesses of an application data file. This field is treated as numeric.

**\*INSERT.** FILE command argument used for specifying that the current contents of the file buffer are written to the file.

**\*IOSTATUS.** Alterable communication area field that contains the return status code from the data manager after accessing an application data file. This field contains the value zero if no error is encountered. If a file error does occur, the HP ALLBASE/4GL data manager writes a file error code to this field. This field is treated as numeric.

**items.** Individual, named, components of the application, which together make up the working application. Each separate name in a catalogue list names an application item.

## J

**\*JOINER=.** LINK and LINKLOOP command argument for specifying insertion of a character or string of characters between each field to be concatenated.

**Justification.** Alignment of data on a boundary or margin. HP ALLBASE/4GL provides automatic justification of data within the left and right boundaries (minimum and maximum length) of the field. The data may be justified on the first character position (left justified) or the last character position (right justified). Data may also be centered within the field, or no justification need be performed.

## K

**KEYS.** Logic command to display a set of function keys on the screen.

**\*KEY=.** CHECK, FILE, and VALIDATE command argument specifying the key value used to access KSAM files.

## L

**\*LAST.** FILE command argument specifying that the last record in an indexed file is to be read.

**LENGTH.** Logic command for calculating the length of the contents of a field.

**LINK.** Logic command for concatenating nominated fields.

**LINKLOOP.** Logic command used for concatenating a number of consecutive fields.

**Literal.** A string of characters surrounded by quotes.

**Logic block.** The commands that make up a process or function. Each logic block contains from 1 to 30 lines of logic commands.

## M

**\*MAP.** CLEAR command argument that specifies that the screen or scratch-pad field position is to be mapped (within the specified screen or the scratch pad).

**MATH.** Logic command for performing an arithmetic operation.

**MATHLOOP.** Logic command for performing repeated arithmetic operations.

**Mess.** A message type that is displayed for information only and does not require a response from the user.

**MESSAGE.** Logic command for displaying a message on the screen.

**MODE.** Logic command used for specifying the names of the files that are being used in the current process. This command can only be used in a process. Files can be defined as being updated or locked to the current process. Files that are not defined as being updated are available for reference only. The MODE command remains in effect until the next process is executed.

**\*MODIFY.** FILE command argument specifying that the contents of the file buffer are to be used to update or change an existing record on a fixed length record serial file or KSAM file. This record must be the last one read for that file.

**\*MOREREC.** A switch used in conjunction with variable length record serial files. If HP ALLBASE/4GL reads sufficient characters to fill the file record buffer without encountering a newline character when reading from a serial file with

## Glossary

variable length records, the data manager sets the switch **\*MOREREC** *on*. If HP ALLBASE/4GL encounters a newline character before the buffer is filled, or the number of characters read exactly matches the length of the buffer, **\*MOREREC** is set *off*.

**MOVE.** Logic command for copying contents of one field (or literal) to another field. The **MOVE** command can also copy the contents of one buffer to another.

**MOVELOOP.** Logic command for repeated copying of one set of fields to another.

## N

**\*NEWTIE.** An alterable communication area field that may be set to contain the next screen field number to be selected. Moving a value into **\*NEWTIE** performs the same function as the logic command **TIE**. This field is numeric.

**\*NEXT.** **FILE** command argument specifying that the next record on the file is to be read.

**NOTE.** Logic command that allows the insertion of a comment in a logic block. HP ALLBASE/4GL ignores **NOTE** command steps when it executes the logic block.

**\*NULL.** **IF** and **IFLOOP** command argument for testing the status of the null indicator variable associated with fields on record buffers for HP ALLBASE/SQL tables and select lists. Also used as a command argument for the **MOVE** command to set the indicator variables to the null status.

**\*NUMERIC.** **IF** and **IFLOOP** command argument used for checking if a field contains only numeric characters. (That is, 0 through 9, +, -.)

## O

**\*OFF.** **IF** and **IFLOOP** command argument used for testing if a switch is currently *off*.

**OFF.** Logic command that sets a switch to its *off* state.

**\*ON.** **IF** and **IFLOOP** command argument used for testing if a switch is currently *on*.

**ON.** Logic command that sets a switch to its *on* state.

## P

**\*P.** CLEAR command argument indicating that the field or fields to be cleared are scratch-pad fields. By itself, the \*P argument indicates that all scratch-pad fields are to be cleared.

**Pad character.** A specified character used to fill a field. Typically a space or 0.

**\*PAGELINE.** Alterable communication area field containing the number of the current print line on the current page of a report. This field is treated as numeric.

**\*PAGENO.** Alterable communication area field containing the current page number of a report. This field is treated as numeric.

**\*PASS.** Alterable communication area field used to pass arguments and results. For example, the CHECK command places the matching position for data in a file or table into \*PASS. This field may be referenced by substring.

**\*PRESENT.** VALIDATE command argument used to check if the specified KSAM data file record field exists.

**\*PREVFLD.** Communication area field containing the number of the last screen field to be successfully committed.

**\*PREVIOUS.** FILE command argument specifying that the previous record on a fixed length record serial file or KSAM file is to be read.

**PRINT.** Logic command to print a defined report line group from within a function that has been called during the generation of a report. The PRINT command is only available during report generation, and is ignored at any other time.

**PROCEED.** Logic command to execute a process logic block.

**Process.** An HP ALLBASE/4GL process is similar to a program in a conventional language system. Each process logic block contains from 1 to 30 lines of logic commands.

**\*PROCESS.** Alterable communication area field name containing the name of the current process.

## Q

**Query.** A message type that requires the user to enter a response.



## R

**Range.** A validation range specifying the lower and upper limits for the contents of a field.

**\*RANGE=.** Report generator record selection option that specifies the name of a HP ALLBASE/4GL range to be used for testing the contents of a specified field when selecting records for reporting.

**\*READ.** FILE command argument specifying that the record with the given key is to be read.

**\*RECNO.** Communication area field that contains the number of the record just read or written for a fixed length record serial data file. The value in \*RECNO is undefined after HP ALLBASE/4GL accesses any other data file type.

**\*REFRESH.** SHOW command argument specifying that the contents of the nominated fields are to be obtained from the main or default data movement field.

\*REFRESH is also a command argument for the EXTERNAL command. If \*REFRESH is specified, HP ALLBASE/4GL restores the terminal configurations and refreshes the screen on return from an external program.

**REPORT.** Logic command to execute a report

**\*REPORT.** Alterable communication area field containing the name of the report being executed (or blank if not currently executing a report).

**\*RESET=.** DISPLAY command argument that clears all lines from the nominated line to the end of the scroll area before any further display within the scroll area.

**\*ROUTINE.** Alterable communication area field which contains the name of the current external routine (or blank if not currently in a called routine).

**\*ROWCOUNT.** Communication area field containing a number indicating the number of rows that HP ALLBASE/SQL processes as the result of a command that changes an SQL table.

**S**

**\*S.** CLEAR command argument indicating that the field or fields to be cleared are screen fields. When used by itself, \*S indicates that all fields on the screen are to be cleared.

**Scratch Pad.** The HP ALLBASE/4GL scratch pad contains up to 99 scratch-pad fields. The scratch-pad fields are available for temporary storage of data. The scratch pad is dynamic and each scratch-pad field takes on the attributes of the data moved into it.

**\*SCREEN.** Alterable communication area field containing the name of the current screen (or blank if no screen).

**SCREEN.** Logic command to display a screen.

**SCROLL.** Logic command that displays data in the scroll area of a screen.

**SELECT.** Logic command that executes one of a series of commands from a selection list. The system executes the command identified by the value contained in the communication area field \*PASS.

**select list.** A “virtual” SQL table. The columns of a select list are taken directly from an existing table or view, or computed using an expression or aggregate function. HP ALLBASE/4GL automatically builds a record layout for each select list.

**SERIES.** Logic command that executes the specified command step number, or range of command steps in the current logic block. After the nominated commands have been executed, the command immediately following the SERIES command is then executed.

**SHOW.** Logic command that displays the current contents of a specified screen field or fields. When used with the \*REFRESH option, the SHOW command refreshes the screen buffer from the main or default data movement fields.

**\*SHOWING.** A switch used by the screen processing logic. When the SHOW command is displaying the contents of a field, \*SHOWING is set to on.

**SQL Logic Block.** Mechanism for passing SQL commands from HP ALLBASE/4GL to HP ALLBASE/SQL. An SQL logic block can contain up to eight SQL commands. These commands are passed to HP ALLBASE/SQL when the SQL logic block is invoked by an SQL command in an HP ALLBASE/4GL logic block.

**Subscript.** An integer or integers used to refer to a specific occurrence of a file record field when the field is specified as occurring more than once. HP ALLBASE/4GL subscripts are enclosed in parentheses.

**Substring.** A portion of the contents of a field. A substring is referenced by giving its starting character position in the field, a comma, and the length (in characters) of the substring, all enclosed in square brackets. For example, [3,2] is a substring of length 2, starting at character position number 3.

**\*SUITE.** Read-only communication area field containing the name of the application being run. If the current application is a version derived from a base application, \*SUITE contains the name of the base application.

## T

**\*TABLE=.** CHECK command argument that identifies the table against which checking is to be performed.

**TIE.** Logic command for specifying the next screen field to be selected. Used where the next screen field to be accessed is not necessarily the next field in sequence on the screen.

**\*TIME.** Read-only communication area field containing the current time. This field may be referenced by substring.

**TOP.** Logic command for specifying that step number 1 of the logic block is the next step to be executed.

**\*TOTALS(n).** A set of 16 alterable communication area fields used primarily in the report generator for totalling the values of numeric fields. The fields are identified as \*TOTALS(1) through \*TOTALS(16). These fields are treated as numeric.

**Training mode.** Operating mode used to prevent the updating of application data files from a specific user. While in training mode, all requests to update data files are ignored. The system administrator can set training mode *on* permanently for any user.

**TRANSACT.** Logic command that defines blocks of logically related file transaction commands.

## U

**UPDATE.** Logic command that updates all KSAM and serial files whose buffer contents have been modified. This command has the same effect as issuing a FILE \*WRITE command for each relevant file.

**\*USER.** Read-only communication area field containing the name of the current user.

- V** **VALIDATE.** Logic command used to check if the contents of a specified field are missing (\*ABSENT) or present (\*PRESENT) on the relevant KSAM file record.
- Version.** An application that has been modified from a base application for a particular end user or group of end users.
- \*VERSION.** A read-only communication area field that contains the name of the current version being run.
- VISIT.** Logic command for executing a function logic block.
- W** **Warn.** A message type that beeps the terminal bell to alert the user, but does not require a response.
- WINDOW.** Logic command to display a window on the current screen.
- Window.** Screen that is overlaid on the current screen starting at the line number defined for the current screen. Window screens operate exactly as normal data screens. However, they may not contain windows or scroll areas. Window screens cannot be displayed independently of a normal data screen or on a menu screen.
- \*WRITE.** FILE command argument used for specifying that the contents of the file buffer are to be written to a serial file or KSAM file.
- X** (no entry)
- Y** (no entry)
- Z** **ZIP.** Logic command that performs no operation and allows the next command to be executed.

# Index of Names

## alphanumeric constants

- add 10-14, 12-3
- delete 10-14, 12-3
- modify 10-14, 12-3
- no.record 9-17, 11-12
- option 10-14, 12-3
- product 10-15, 12-3
- record 9-17, 11-12
- review 10-15, 12-3

## application title

- application.name 9-12, 11-12

## calculated item

- option.cost 10-93, 12-25, 12-32

## data files

- customer 4-22
- option 10-11
- product 10-11
- supplier 10-101

## decision table

- prod.opt.update 10-42, 12-4

## field specifications

- address 4-11, 11-5
- address1 11-5
- address2 11-5
- cost 10-10, 12-3
- customer.name 4-11, 11-5
- customer.no 4-7, 11-5
- description 10-9, 12-3
- lead.time 10-9, 12-3
- opt.descript 12-25
- option.key 10-9
- option.no 10-10, 12-3
- product.no 10-9, 12-3

- qty.on.hand 10-10, 12-3
- state.code 4-11, 11-5
- supplier.name 10-100
- supplier.no 10-9, 12-3
- zip.code 4-12, 11-5

## files

- customer 4-22
- option 10-11
- product 10-11
- supplier 10-101

## function keys

- customer.func 11-12
- customer.keys 9-20
- option 10-27, 12-4
- product 10-27, 12-4

## functions

- add.option 10-55, 12-4, 12-11
- add.product 10-54, 12-4, 12-10
- customer.func 6-13, 9-24, 11-15
- del.all.options 10-69
- del.corrupt.prod 10-71
- delete.option 10-73, 12-4, 12-16
- delete.product 10-67, 12-14
- get.lst.option 10-61, 12-4, 12-22
- get.next.option 10-63, 12-23
- mb.supplier.L08 10-108
- modify.option 10-66, 12-4, 12-11, 12-12
- modify.product 10-65, 12-4
- new.option 10-34, 12-4
- option 10-37, 12-4
- option.cost 10-93
- option.key.read 10-34, 12-4, 12-8
- option.no 10-39, 12-4
- option.present 10-86
- print.option 10-86
- prod.add 10-28, 12-4

- prod\_delete 10-28, 12-4
- prod\_modify 10-29, 12-4
- prod\_review 10-29, 12-4
- product 10-38, 12-4
- product\_key\_read 10-31, 12-4, 12-7
- scroll\_options 10-59, 12-4, 12-21
- select\_customer 11-32
- select\_products 10-77, 12-25, 12-27
- set\_add 9-22, 11-12
- set\_delete 9-23, 11-12
- set\_modify 9-23, 11-12
- set\_review 9-24, 11-12

## help screens

- product 10-98
- product\_no 10-97
- product\_no\_error 10-97

## messages

- add 9-30, 11-12
- add\_ok 9-32, 11-12
- add\_option 10-53, 12-3
- add\_prod 10-52, 12-3
- all\_opt\_deleted 12-15
- corrupt\_prod\_del 10-73
- delete 9-32, 11-12
- delete\_ok 9-32, 11-12
- delete\_option 10-74, 12-3
- delete\_options 10-68, 12-3
- delete\_product 10-69, 12-3
- del\_prod\_opt 10-36, 12-3
- file\_error 9-31, 11-12
- modify 9-31, 11-12
- modify\_ok 9-32, 11-12
- no\_add 9-32, 11-12
- no\_delete 9-33, 11-12
- no\_modify 9-32, 11-12
- no\_option\_review 10-53, 12-3
- no\_option\_warn 10-66
- no\_product 10-53, 12-3
- no\_prod\_warn 10-65, 12-3
- no\_review 9-33, 11-12
- no\_transact 10-31
- opt\_add\_ok 10-56, 12-3
- opt\_delete\_fail 10-74, 12-3

- opt\_delete\_ok 10-74, 12-3
- option\_warn 10-56, 12-3
- opt\_modify\_ok 10-66, 12-3
- opt\_not\_exist 10-53, 12-3
- prod\_add\_ok 10-55, 12-4
- prod\_corrupt 10-64, 12-4
- prod\_delete\_fail 10-69, 12-4
- prod\_delete\_ok 10-69, 12-4
- prod\_modify\_ok 10-65, 12-4
- prod\_not\_exist 10-52, 12-4
- product\_no\_error 10-52, 12-3
- product\_warn 10-54, 12-4
- review 9-32, 11-12
- state\_code\_error 9-37, 11-12

## modules

- supplier 10-103, 10-108

## numeric constants

- beg\_end\_of\_file 11-12
- end\_of\_file 10-15
- record\_not\_found 9-15
- zero 10-15, 12-3

## processes

- customer\_proc 6-4, 9-4, 11-13
- mb\_supplier 10-108
- prod\_opt 10-64, 12-4
- product 10-29, 12-6

## range

- product\_no 10-11, 12-3

## record layouts

- customer\_rcrd 4-15, 11-6
- option 10-11, 12-5
- option\_scroll 10-59
- product 10-10, 12-4
- supplier 10-100

## reports

- customer\_rept 8-5, 11-31
- product 10-76, 12-28

## screens

customer\_scrn 5-19, 11-12  
 main 5-5, 10-18, 11-12  
 mb\_supplier 10-5, 10-108  
 new\_option 10-25, 12-4  
 option 10-22, 12-4  
 prod\_opt\_menu 10-18, 12-4  
 product 10-19, 12-4  
 prod\_wnd 10-25, 12-4  
 select\_products 10-76, 12-25

**select lists**

optscrol 12-20  
 prod\_opt 12-26

**SQL logic blocks**

commit 11-21  
 customer 11-18  
 modify\_option 12-12  
 modify\_product 12-12  
 opt\_del 12-15  
 option\_key\_sel 12-9  
 optscrol 12-22  
 opt\_sel 12-14  
 prod\_del 12-16  
 prod\_key\_sel 12-8  
 prod\_opt 12-27  
 select\_customer 11-32  
 update 11-21

**SQL tables**

customer 11-8  
 option 12-5  
 product 12-5

**table**

state\_code 9-35, 11-5

**validation range**

product\_no 10-11, 12-3

**validation table**

state\_code 9-35, 11-5

**variables**

confirm 10-13, 12-3  
 current\_record 10-13, 12-3  
 file\_status 9-14, 11-12  
 first\_product 12-25  
 last\_product 12-25  
 mode 10-14, 12-3  
 option\_no 12-8  
 option\_status 10-14, 12-3  
 product\_status 10-14, 12-3

**windows**

new\_option 10-25, 12-4  
 option 10-22, 12-4  
 prod\_wnd 10-25, 12-4

**Intentionally Blank**



# Subject Index

## A

- ABORT message 9-28
- access class, SQL 11-8
- action codes 4-17
- action items on menus 5-14
- action prefixes 4-4
- actions from menus 5-39
- activating menu items 2-4
- administrator
  - application 1-3
  - application definition 1-3
  - responsibilities 1-4
  - sign-on A-2
- after function if blank 5-30
- after selection functions 10-81
- alphanumeric constants 9-16
- AND/OR connectives 9-7
- application
  - definition A-4
  - definition (SQL) A-7
  - generation 7-6
  - name 2-3
  - password A-5
  - prototyping 1-12
  - structure 3-3
  - testing 7-2
  - testing environment 5-41
  - titles 5-14, 9-11
- application items 10-99
- auto flow, screen fields 5-30
- automatic numbering, data screen fields 5-19

## B

- behavioral characteristic, screen fields 5-29
- blank when zero 4-9, 5-47
- block functions in screen painter 5-10

- \*BUFFER argument, FILE command 9-25
- \*BYPASS switch 8-12, 10-81

## C

- calculated item functions 10-92
- calculated items 1-9, 10-92
- case conversion 4-11, 5-47
- catalog display utility 7-20
- class, access SQL 11-8
- clear fields on commit 5-19
- clearing fields on screens 5-29
- closing cursors 11-24
- column entry mode, in screen painter 5-21
- command windows 6-5
- commit actions
  - automatic 5-30
  - data key 2-6
  - field 2-6, 5-43, 5-45, 7-15
  - field commit 10-37
  - function keys 5-38
  - screen 2-6, 5-43, 5-44
- COMMIT WORK command 11-14, 11-27
- communication area fields
  - \*COUNT 8-3
  - \*CROSS 8-3, 8-21
  - \*FIELDNO 7-11, 7-17, 10-37
  - \*IOSTATUS 9-25, 11-24
  - \*NEWTIE 7-11
  - \*ROWCOUNT 12-15
  - \*SCREEN 5-12
  - \*TIME 5-13
  - \*TOTALS 8-3, 8-21, 10-89
- component printing utility 7-24
- conditional tests 9-7
- connectives, AND/OR 9-7
- control breaks 1-15
- control breaks in reports 10-78

- copying fields on reports 8-24
- copying utility 7-21
- \*COUNT communication area fields 8-3
- \*CROSS communication area fields 8-3, 8-21
- cursor keys 2-4
- cursor position, in screen painter 5-7
- customer\_proc process modifications 9-4

## D

- D- actions 5-38
- data definition 1-6
- data fields on reports 8-19
- data file manager 4-17
- data files**
  - ALLBASE/4GL names 4-22
  - creation 4-24
  - default record 4-23
  - definition 4-21
  - error return codes 9-15
  - external name 4-23
  - file type codes 4-22
  - KSAM 4-21
  - multiple record layouts 10-58
  - record layout list 4-23
  - report primary file 8-1
- data formatting on screens 5-46
- data justification 4-8
- data manager 1-3
- data movement fields**
  - default 5-27, 5-53
  - definition 5-27
  - other 5-27, 5-53
  - primary 5-27, 5-31, 5-53
  - screen field 5-52
- data movement in screens 5-51, 5-54
- data movement in SHOW command 5-57
- data screen processing logic 5-43
- data screens**
  - after function 7-17
  - after function if blank 5-30
  - after functions 5-30
  - auto flow 5-30
  - automatic field numbering 5-19
  - both functions 5-30

- buffers 5-51
- clearing fields 5-19, 5-29
- data formatting 5-46
- data movement 5-51, 5-54, 7-17, 7-18
- data movement fields 5-52
- data validation 7-16
- decimal place validation 5-49
- definition 5-17
- display fields 5-29, 5-41
- display processing 5-45
- edit code validation 5-49
- field behavior specification 5-29
- field commit action 5-43, 10-37
- field details 5-25
- field display logic 7-12
- field functions 5-30, 5-56
- field level help 5-29
- field level processing 5-45
- field sequence number 5-26
- generation 5-33
- header 5-18
- help screen name 5-29
- include in SHOW 5-29
- input field logic 7-14
- input fields 5-29, 5-41
- input processing 5-45
- introduction to 1-10
- item types 5-41
- justification of fields 5-47
- padding of fields 5-47
- painting 5-20
- prior functions 5-30
- processing logic 5-43, 7-10
- properties 5-37
- range validation 5-50
- required fields 5-29, 5-48
- screen commit action 5-43
- screen level processing 5-43
- scroll area 5-43, 10-19
- SHOW functions 5-30
- system items 5-21, 5-41
- table validation 5-49
- text items 5-21
- text literals 5-41
- tutorial 5-40

- uppercase fields 5-47
- validation of data 5-48
- window display 10-22
- window starting line 10-19
- data validation**
  - data screen input 7-16
  - edit code checking 5-49
  - error messages 7-16
  - field specification requirements 4-9
  - number of characters 5-49
  - number of decimal places 5-49
  - ranges 5-48, 5-50, 10-12
  - tables 5-48, 5-49, 9-34
- database definition, for SQL application A-8
- database items 1-9
- date format 5-13
- date system item 5-12
- decimal places, in field specification 4-9
- decision tables**
  - actions 10-45
  - definition 1-14
  - generation 10-51
  - header screen 10-42
  - IF test 10-43
  - questions screen 10-43
  - relationships 10-47
  - usage 10-41
- DECLARE CURSOR command 11-20
- default data movement field 5-27, 5-53
- default record layout 4-23
- DEFINE command 10-59
- defining a process 6-4
- DELETE command (SQL) 11-27
- deleting items on screens 5-9
- deletions utility 7-23
- description fields 4-10
- developer user name A-3
- developer validation A-2
- development security codes 4-7, A-6
- dictionary**
  - application titles 5-14
  - database items 1-9
  - definitions 1-6
  - field specifications 1-7, 4-5
  - help screens 1-10

- messages 1-9
- record layouts 1-9
- select list name 11-20
- storage items 1-8, 9-11
- table name 11-20
- validation items 1-8
- DISPLAY command 10-61
- display enhancements 5-43
- display field processing 7-12
- display fields on screens 5-22, 5-24, 5-29, 5-41
- display listing utility 7-20
- display only fields on screens 5-24
- displaying screens 5-38
- documentation fields 4-10
- duplicate keys 4-17

## E

- edit code validation 5-49
- edit codes 1-8, 4-7
- editing logic blocks 9-6
- editing text, in screen painter 5-8
- end user environment 1-4
- \*ENDLINE switch 8-12, 10-81, 10-87
- \*ENTERED switch 7-16, 10-32
- error message 7-16, 9-28
- errors during generation 4-20
- EXIT command 9-26
- exiting from ALLBASE/4GL 2-9
- external file name 4-23

## F

- FETCH command (SQL) 11-23
- field commit 2-6
- field commit actions 5-43, 5-45, 7-15
- field details function key 2-8
- field level help, data screens 5-29
- field level processing, data screens 5-45
- field specifications**
  - blank when zero 4-9
  - data validation 4-9
  - decimal places 4-9
  - definition screen 4-5
  - dictionary 1-7

- edit code 4-7
- forced uppercase 4-11
- help screen name 4-9
- justification code 4-8
- length 4-7
- minimum entry 4-7
- pad character 4-8
- repeated fields 4-7
- report fields 8-22
- required field 4-7
- screen field definition 5-26
- screen fields 5-22
- \*FIELDNO communication area 7-11
- \*FIELDNO communication area field 7-17, 10-37
- FILE command**
  - \*BUFFER argument 9-25
  - \*FIND argument 10-61
  - \*INDEXNO= 6-14
  - \*INSERT argument (SQL) 11-25
  - \*KEY= 6-15
  - \*NEXT argument 10-62
  - \*NEXT argument (SQL) 11-23
  - usage 6-8
- file linkages in reports 1-15, 10-83
- file manager 9-15
- file name, external 4-23
- files**
  - creation 4-24
  - indexes 1-9
  - KSAM 4-21
  - record layouts 1-9
  - record references 5-28
- \*FIND argument, FILE command 10-61
- FOR UPDATE OF clause 11-24
- forced case conversion 4-11
- function details screen 6-13
- function header screen 6-12
- function keys**
  - actions 5-38, 9-19
  - default definitions 5-38, 9-20
  - definition 9-19
  - field commit from function 7-15
  - internal actions 5-38
  - introduction to 1-11

- label text 9-21
- more keys 2-9
- standard keys 2-7
- switches 9-21
- system keys 2-8
- functions**
  - after 5-30, 7-17
  - after print 8-12
  - after selection 10-81
  - before print 8-12
  - both 5-30
  - calculated item 10-92
  - end of report 8-8
  - generation 6-16
  - introduction to 1-14
  - modification 9-6
  - prior 5-30
  - screen field 5-30, 5-56
  - start of report 8-7

## G

- generate program 4-19
- generation**
  - database module files 11-22
  - decision tables 10-51
  - error messages 4-20
  - functions 6-16
  - log file 7-27
  - menu 7-26
  - messages 9-31
  - processes 6-9
  - record layouts 4-19
  - reports 8-26
  - resolution of references 7-6
  - screens 5-33
  - select lists 12-21
  - SQL applications 11-22
  - SQL commands 11-20
  - SQL logic blocks 11-18

## H

- help function key 2-7

**help screens**

- definition 10-97
- dictionary 1-10
- function keys 2-7
- names 4-9, 5-29
- host variable referencing, (SQL) 11-20
- HP ALLBASE/SQL 1-16, 11-1

**I**

- IF command 9-6
- include in SHOW 5-29
- initial action**
  - correcting mistakes 7-8
  - name A-6
  - type A-6
- input fields on screens 5-22, 5-29, 5-41
- input justification 4-8
- input processing, data screen fields 5-45
- internal actions 5-38
- internal routines, commit data 5-44
- \*IOSTATUS communication area field 9-25, 11-24
- item names 4-1
- item types on screens 5-41

**J**

- joining HP ALLBASE/SQL tables 12-27
- justification codes 4-8
- justification on screens 5-47

**K**

- key fields 4-17
- KSAM data manager 1-3

**L**

- layout keys, in screen painter 5-9
- line drawing characters 5-43, 10-20
- line groups in reports 1-15, 8-2
- line headers in reports 8-8

- line numbers in reports 8-2
- LINK command 10-35, 10-39
- linkages in reports 1-15, 10-83
- LINKLOOP command 10-55
- listing application components 7-20
- local printers 8-7
- log file, generate errors 7-27
- log file, trace mode 7-6
- logic block 6-1
- logic block modification 9-4, 9-6

**logic commands**

- DEFINE 10-59
- DISPLAY 10-61
- EXIT 9-26
- FILE 6-8
- IF 9-6
- LINK 10-35, 10-39
- LINKLOOP 10-55
- MODE 6-5
- MOVELOOP 10-77
- PROCEED 10-29, 10-62
- SCREEN 5-38, 6-7
- SCROLL 10-60
- SHOW 5-51, 5-56, 6-15
- SQL 11-14
- TIE 7-11
- TOP 6-9
- TRANSACT 10-29, 10-39
- VISIT 9-25

**logic constructs**

- decision tables 1-13, 10-41
- functions 1-13, 6-1
- processes 1-13, 6-1
- SQL logic blocks 1-13, 11-2
- logic step modification 9-8

**M**

- menu bypass 2-4
- menu item selection 2-4
- menu item, activating 2-4
- menu names 2-4
- menus**
  - action items 5-14, 5-36
  - actions 5-38, 5-39

- features 5-36
- headers 5-5
- introduction 1-10
- menu bypass 8-27
- menu path 1-17
- processing 5-39
- MESS message 9-28
- messages**
  - construction 9-30
  - definition 9-29
  - generation 9-31
  - help screen name 9-30
  - introduction 1-9
  - types 9-28
- minimum entry length 4-7
- MODE command 6-5
- modifying a step 9-8
- modifying logic blocks 9-4, 9-6
- module builder**
  - building process 10-108
  - description 10-99
- module files, SQL applications 11-22
- more keys function keys 2-9
- MOVELOOP command 10-77
- moving and copying items on screens 5-10
- MPE XL operating system 1-2
- multiple record layouts 10-58

## N

- naming rules 4-1
- naming rules, SQL cursors 11-26
- \*NEWTIE communication area field 7-11
- \*NEXT argument, FILE command 10-62
- number of characters 5-49
- numeric constants 9-14

## O

- object code 1-5
- operating system 1-2
- operating system interface 1-2
- OR connective tests 9-7
- other data movement field 5-27, 5-53

## P

- pad character 4-8, 5-47
- pad character, variables 9-14
- painter program 1-12
- painting a window 10-23
- painting data screens 5-20
- painting menus 5-7
- paper for reports 8-7
- password, application A-5
- password, developer A-3
- perform on SHOW 5-30
- prefixes 4-1
- prefixes, item referencing 4-2
- previous menu action 5-38
- previous menu key 2-4
- primary data movement field 5-27, 5-53
- primary data movement, automatic default 5-31
- primary file for report 1-14, 8-1
- primary key fields 4-17
- print line suppression 8-11
- printers 8-7
- printing screen images 2-8
- printing utility 7-24
- PROCEED command 10-29, 10-62
- processes**
  - definition 6-4
  - generation 6-9
  - introduction to 1-13
  - logic constructs 6-1
  - modification 9-6
  - process details screen 6-5
  - process header screen 6-4
- processing logic, data screens 5-43
- prototyping applications 1-12

## Q

- QUERY message 9-28

## R

- ranges 10-11, 10-12

**record layouts**

- definition 1-9, 4-14
- details screen 4-16
- duplicate keys 4-17
- generation 4-19
- header 4-14
- key fields 4-17
- multiple 10-58
- primary key 4-17
- select list 12-21
- SQL tables 11-6
- record selection for reporting 1-15, 10-80
- record sorting for reports 1-15, 10-78
- referencing prefixes 4-2
- referencing rules 4-2
- referring to the wrong step 9-9
- repeated fields 4-7

**report painter**

- copying fields 8-24
- data fields 8-19
- dictionary field specifications 8-22
- entry prompt 8-15
- field details 8-20
- menu path 8-15
- saving reports 8-25
- text literals 8-18
- window keys 8-17

**reports**

- after print line function 8-12
- before print line function 8-12
- control breaks 1-15, 10-78, 12-31
- counting print lines 8-11
- definition 1-14, 8-1
- end of report function 8-8
- file linkages 10-83
- generation 8-26
- header screen 8-5
- line groups 1-15, 8-2, 8-9
- line headers 8-8
- line numbers 8-2, 8-10
- linkages 1-15
- manual record selection 10-81
- output file 8-6
- page size definition 8-7
- painter 1-16

- primary file 1-14, 8-1, 8-6
- printers 8-7
- record selection 1-15
- record selection and sorting (SQL) 12-28
- record sorting 1-15, 10-78
- selection criteria 10-80
- sort order 10-78
- start up function 8-7, 10-76
- stationery 8-7
- suppressing print lines 8-11
- totalling 8-3
- totalling facilities 10-89
- user input 8-7, 10-76
- using SQL data 11-31, 12-26
- required fields 2-6, 4-7, 5-29, 5-48
- \*ROWCOUNT communication area field 12-15
- run-time environment 1-4

**S**

- saving a screen 5-16
- scratch-pad fields 1-9, 10-77
- SCREEN command 5-38, 6-7
- screen commit action 5-43
- \*SCREEN communication area field 5-12
- screen field buffers 5-51
- screen field details
  - definition screen 5-25
  - introduction to 5-25
- screen field functions
  - after 5-30
  - both 5-30
  - prior 5-30
  - show 5-30
  - usage 5-30
- screen field sequence number 5-26
- screen help function key 2-8
- screen images, printing 2-8
- screen painter
  - block functions 5-10
  - column entry mode 5-21
  - cursor movement 5-9
  - cursor position 5-7
  - data fields 5-22
  - date fields 5-12

- deleting items 5-9
- display fields 5-22, 5-24
- editing action items 5-15
- entering text 5-8
- entry prompt 5-7
- field specifications 5-22
- input fields 5-22
- layout keys 5-9
- line drawing characters 10-20
- menu action items 5-14
- menu path 5-6
- moving and copying items 5-10
- number of characters 5-23
- overlay screen 10-23
- painting a menu 5-7
- painting data screens 5-20
- painting windows 10-23
- saving screens 5-16
- screen name 5-12
- special text 10-20
- system items 5-11
- time fields 5-13
- screen processing**
  - input field logic 7-14
  - logic 7-10
  - logic, field display 7-12
  - screen processing cycle 7-9
- screens**
  - data 1-10, 5-37
  - function keys 1-11
  - menu 1-10
  - menu header 5-5
  - menus 5-36
  - module builder 10-101, 10-103
  - painter 1-12
  - types 1-10, 5-36
  - windows 1-11, 5-37
- scroll area on screens 5-43, 10-19
- SCROLL command 10-60
- security**
  - development security codes 4-7
  - system security 1-4
- SELECT command (SQL) 11-20, 12-9
- SELECT command limitations 11-20
- select lists**
  - as report primary file 12-26
  - definition 11-2, 12-18
  - details screen 12-20
  - field referencing 12-21
  - generation 12-21
  - header screen 12-19
  - in SELECT command 12-22
  - joining tables 12-27
  - record layout 12-21
- selecting menu items 2-4
- selecting records for reporting 1-15, 10-80
- selection criteria screen 10-80
- sequence number, screen field 5-26
- SHOW command**
  - data buffers 5-51
  - data movement 5-57
  - include field 5-29
  - \*REFRESH argument 6-15
  - usage 5-56
- SHOW functions 5-30
- sign-on screen 2-2
- signing off 2-9
- slave printers 8-7
- sorting in reports 10-78
- sorting records for reporting 1-15
- source code 1-4, 1-5
- source code security 4-7
- special text 5-43
- special text, on screens 10-20
- SQL application generation 11-22
- SQL command 11-14
- SQL command generation 11-20
- SQL commands**
  - COMMIT WORK 11-14, 11-27
  - DELETE 11-27
  - SELECT 11-20, 12-9
  - UPDATE 11-26
- SQL cursor naming 11-26
- SQL cursors 11-23, 12-8
- SQL data manipulation**
  - delete without using a cursor 12-15
  - deleting records 11-15, 11-26, 12-13
  - inserting records 11-15, 11-25, 12-10
  - modifying records 11-15, 11-25, 12-11



- retrieving data 12-7
- retrieving records 11-23
- updating records 12-9
- SQL database**
  - access authority A-9
  - definition A-8
  - module files 11-22
  - name A-8
- SQL DECLARE CURSOR command 11-20
- SQL FETCH command 11-23
- SQL interface 1-16
- SQL logic blocks**
  - definition 11-17
  - details screen 11-18
  - generation 11-18
  - header screen 11-17
  - host variable references 12-9
  - introduction to 1-14
  - limitations 11-19
  - multiple commands 12-16
  - purpose 11-2
- SQL owner group A-7
- SQL table**
  - creation 11-8
  - definition 11-7
  - format 11-10
  - ownership 11-9
- start of report function 8-7
- stationery for reports 8-7
- step modification 9-8
- storage items 1-8, 9-11
- storage items**
  - alphanumeric constants 9-16
  - application titles 9-11
  - calculated items 10-92
  - numeric variables 9-14
  - scratch-pad fields 10-77
  - variables 9-13
- structure diagram 3-5
- switches**
  - \*BYPASS 8-12, 10-81
  - display in function keys 9-3
  - \*ENDLINE 8-12, 10-81, 10-87
  - \*ENTERED 5-31, 7-16, 10-32
  - function key 9-21

- \*SHOWING 5-31
- user 9-3
- system items on screens 5-11, 5-41
- system keys function keys 2-8
- system specifications 1-4

## T

- tabbing sequence 2-5
- table creation, SQL 11-8
- terminal keys 2-5
- terminals, touch-screen 2-4
- terminology 1-17
- testing environment 5-41
- tests, connective 9-7
- text entry, in screen painter 5-8
- text items on reports 8-18
- TIE command 7-11
- \*TIME communication area field 5-13
- time fields, on screens 5-13
- titles, application 5-14, 9-11
- TOP command 6-9
- \*TOTALS communication area fields 8-3, 8-21, 10-89
- totals in reports 8-3, 10-89
- touch-screen terminals 2-4
- trace log file 7-6
- trace mode 7-4
- TRANSACT command 10-29, 10-39
- tutorial application 5-40

## U

- U edit code 5-47
- undefined items 7-6
- UPDATE command (SQL) 11-26
- uppercase fields on screens 5-47
- uppercase forced 4-11
- user switches 9-3, 9-4
- users**
  - developer names A-3
  - names 2-1
  - user input for reports 10-76
- utilities**
  - catalog display 7-20

copying 7-21  
deletions 7-23  
generates menu 7-26  
menu 7-19  
printing 7-24

**window**  
data screen 1-11  
definition 10-22  
display 10-22  
screens 5-37  
starting line 10-19

## V

### validation

items 1-8  
of data, on screens 5-48  
ranges 5-50, 10-12  
tables 5-49  
validation table definition 9-34  
variables 1-9, 9-13  
vertically aligned prompts on screens 5-21  
VISIT command 9-25

## W

WARN message 9-28

## X

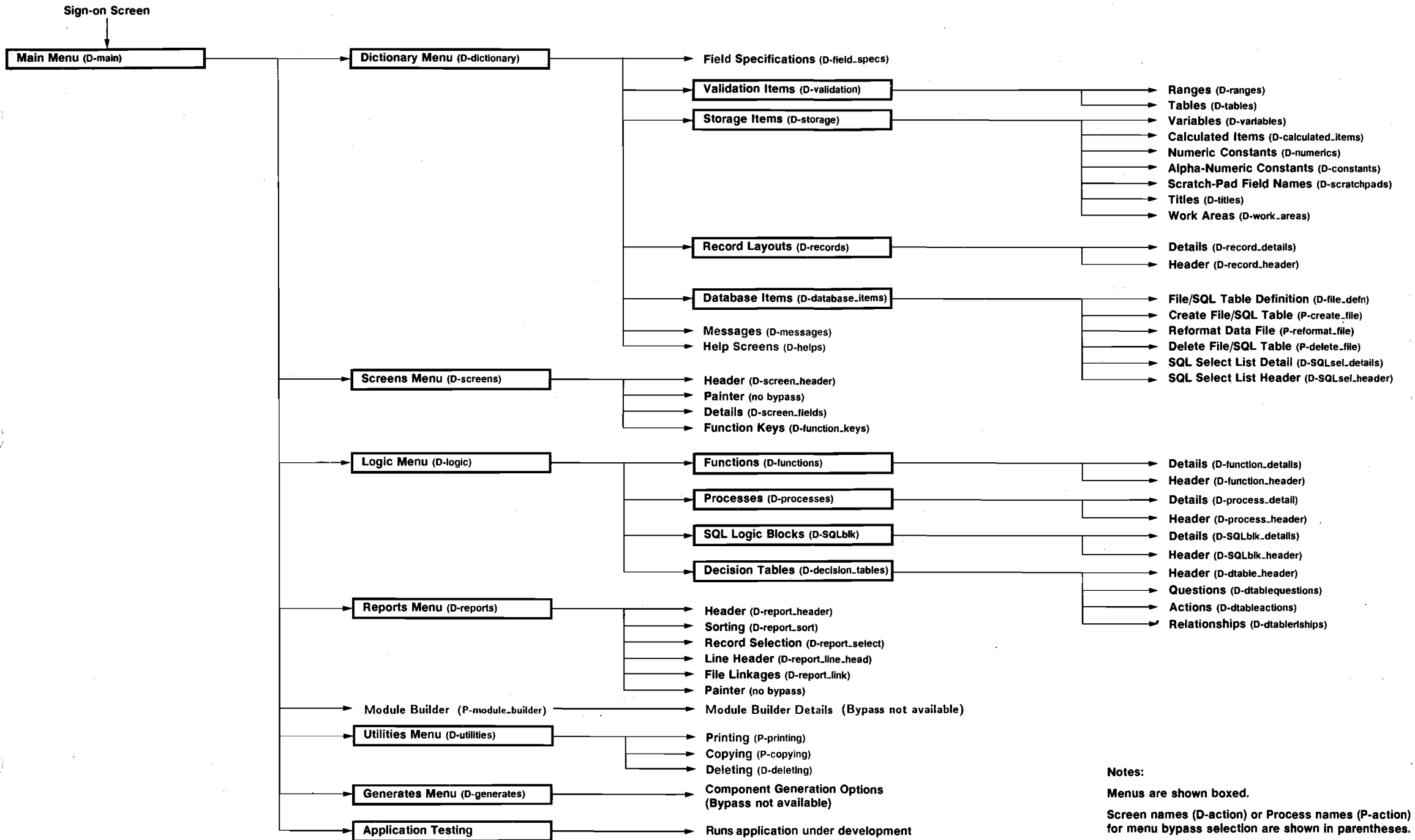
no entry

## Y

no entry

## Z

no entry



**Notes:**  
Menus are shown boxed.  
Screen names (D-action) or Process names (P-action) for menu bypass selection are shown in parentheses.