Module: The HP ALLBASE/SQL Interface I

Instructor Notes

Module - HP ALLBASE/SQL Interface Student Notes

SQL100 - SLIDE: Module Objectives

SQL INTERFACE

Introduction to the HP ALLBASE/4GL SQL Interface
Base Tables
SQL Logic Blocks
A Simple SQL Application

OBJECTIVES:

Upon completion of this module, the student will be able to:

1. List the three components of the HP ALLBASE/4GL SQL interface.
2. Define and create SQL Base Tables within HP ALLBASE/4GL.
3. Explain the purpose of SQL Logic Blocks and how they are defined.
4. Develop a simple application based on an SQL database.

Student Notes

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Module Objective

This module will introduce the students to the interface to HP ALLBASE/SQL provided by HP ALLBASE/4GL.

Student Objectives

At the end of this module, each student will be able to do the following:

- List the different components of the HP ALLBASE/4GL to HP ALLBASE/SQL interface.
- Describe each of the components of the interface.
- Build a simple database using **isql** and a command file.
- Describe the differences between the ISAM/KSAM file interface and the SQL interface.
- Use the interface to copy the data from an ISAM/KSAM file to an SQL table.

Resource Allocation

Equipment

The following items are required for this module:

- Instructor notes.
- Student notes.
- Slides.
- A laboratory worksheet.
- Access to an HP 3000 Series 900 machine or an HP 9000 Series 800 or HP 9000 Series 300 machine with HP ALLBASE/4GL installed..
- The complete set of HP ALLBASE/4GL manuals.

Time

This module should take approximately 1 hour; there is no practical session in this module. That will be done when **The HP ALLBASE/SQL Inteface II** is completed.

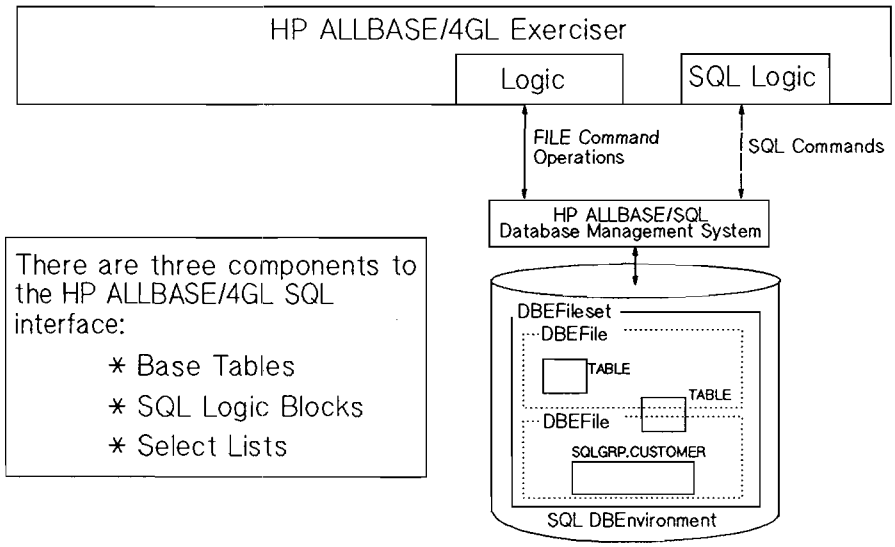
Pacing

Read through the slides reasonably quickly.

SQL101 - SLIDE: Introduction to the HP ALLBASE/4GL SQL Interface

SQL INTERFACE

Introduction to the HP ALLBASE/4GL SQL Interface



There are three components to the HP ALLBASE/4GL SQL interface:

* Base Tables
* SQL Logic Blocks
* Select Lists

Australian Software Operation
sql101

HEWLETT PACKARD

Student Notes

Introduction to the HP ALLBASE/4GL Interface to SQL

SQL Components

The HP ALLBASE/4GL interface to HP ALLBASE/SQL has three parts, namely **Base tables**, **select lists** and **SQL Logic Blocks**.

Tables

As seen in the diagram, tables exist in the HP ALLBASE/SQL database - they are represented as squares in the example. The tables can be created by **isql** using a definition in the database **schema** file or they can be created from within HP ALLBASE/4GL. Whichever way they are created, a definition for the table must exist within HP ALLBASE/4GL, using the **File/SQL Table Definition** screen.

SQL Logic Blocks

HP ALLBASE/4GL communicates with the database through SQL logic blocks, or the **FILE** command. Both of these operations move data into or out of the database, and use file buffers to make that data available to the application.

All operations on the HP ALLBASE/SQL database are converted into SQL commands, so that they can be executed by SQL. Thus all the **FILE** operations are converted into the equivalent HP ALLBASE/SQL operations.

Select Lists

Select lists are the HP ALLBASE/4GL implementation of SQL views. They provide a virtual table, which could be a subset of an existing table, or a combination of several tables and calculated fields.

SQL102 - SLIDE: Base Tables I

SQL INTERFACE

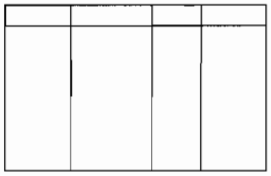
Base Tables

Record Layout: student_info

student_number	student_name	student_sex	student_course

Base Table: student_table

File Type: 'S'
(SQL Base Table)



* Like all HP ALLBASE/4GL Files, SQL Base Tables are defined in the File Definition Screen of the Dictionary.
* The structure of an SQL Base Table is defined by a Record Layout.
* HP ALLBASE/4GL maintains a file record buffer for SQL Tables in the same way as other HP ALLBASE/4GL file types.
* Base Tables can be accessed using the FILE command and SQL Logic Block.

Student Notes

Base Tables I

Definition

HP ALLBASE/SQL base tables are treated by HP ALLBASE/4GL in the same way as other files. The developer must first create a record layout, and then define the table using the **File/SQL Table Definition** screen. Refer to the screen image at the end of this module.

Each record in the table, or **row** as it is known in HP ALLBASE/SQL, is the same format as the record layout.

Creation

HP ALLBASE/SQL base tables can be created by **isql** using a definition in the database **schema** file or they can be created from within HP ALLBASE/4GL, using the **File/SQL Table Creation** screen. Refer to the screen image at the end of this module.

If the table is created using **isql**, it must still be defined in the HP ALLBASE/4GL data dictionary before it can be accessed by an application.

Although it is very easy to create base tables within HP ALLBASE/4GL, greater flexibility is provided by using **isql** to create them within the database. For example, when creating a table within HP ALLBASE/4GL, each column in the table will allow **NULL** characters, which may waste space in the database.

File Buffers

A file buffer is created when the HP ALLBASE/SQL table is used, and data movement to and from the HP ALLBASE/SQL table will generally use this buffer. For example, when an SQL logic block selects a row from an HP ALLBASE/SQL table, the retrieved row will be placed into the appropriate file buffer.

Many of the **FILE** logic commands also expect to find data in, or place data into, the table's file buffer.

SQL103 - SLIDE: Base Tables II

SQL INTERFACE

Base Tables

Base Table: student_table

File Type: 'S'
(SQL Base Table)

student_number	student_name	student_sex	student_course
00054551	Mary Kumar	F	LAW
00054552	Phil Anderson	M	SCIENCE
00054553	Tricia Matsoukas	F	SCIENCE
00054554	Michael O'Connor	M	MEDICINE

SQL Base Tables:

* Do not require key fields to be defined for keyed access.
* Can be included in Select Lists and Views to create cross table joins.
* Do not allow repeated fields.
* Table names and column names are case INSENSITIVE;
  e.g. total, Total and TOTAL are treated the same.
* Can only have one record layout.

Australian Software Operation
sql103

HEWLETT PACKARD

Student Notes

Base Tables II

Unlike ISAM files, HP ALLBASE/SQL tables have the following features

Key Fields

Key fields are not required for keyed access as any column in the table can be used as an index internally by HP ALLBASE/SQL. For a table created within HP ALLBASE/4GL, the primary index for the table will become a **clustering index** with in the database. The developer should have very good reasons for choosing a clustering index; there are inherent limitations, due to HP ALLBASE/SQL, with such a choice. Another limitation is with numeric key fields in the HP ALLBASE/4GL definition; HP ALLBASE/4GL automatically treats all numbers as decimal which may slow down data accesses.

It is best to define a table without key fields within the HP ALLBASE/4GL definition and create it using **isql**.

Select Lists

Base tables can be referenced in SQL select lists, allowing the developer to create **virtual tables** or **views**. The **virtual table** may contain a subset of the columns (fields) in the table, or it may contain columns from a number of different tables, producing a cross table join.

Repeated Fields

Base tables cannot contain repeated fields. If necessary, convert repeated fields from an ISAM/KSAM file record layout to the required number of identical fields so that the record layout can be used with an HP ALLBASE/SQL table.

Case Insensitivity

Table names and column names (field names) are not case sensitive in the HP ALLBASE/SQL internal dictionary. Thus any field or table name that the developer defines will be treated as UPPER-CASE in HP ALLBASE/SQL. Therefore you cannot have two tables whose names only differ in case, such as **total**, **Total** or **TOTAL**.

Only One Record Layout

Each table can only have one record layout.

Intentionally blank

SQL104 - SLIDE: SQL Logic Blocks I

SQL INTERFACE

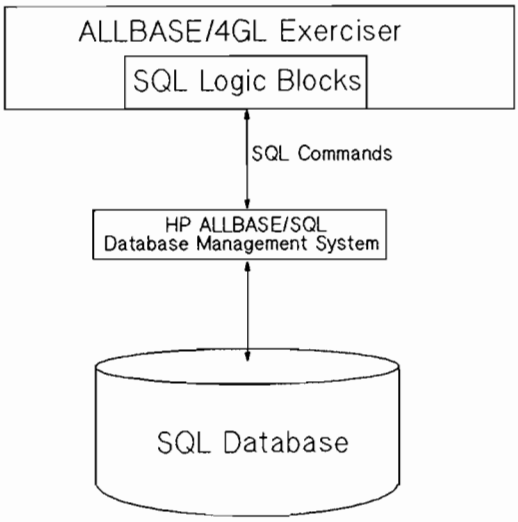
Base Tables

SQL Logic Blocks provide '/4GL applications with direct access to the flexibility and granularity of the SQL language.

SQL Logic Blocks are defined in the SQL Logic Details Screen.

Logic Blocks are syntax checked, compiled by HP ALLBASE/SQL, and stored directly in the database.

Stored Logic Blocks are invoked using the 'SQL' Logic command.



Australian Software Operation
sql104

HEWLETT PACKARD

Student Notes

SQL Logic Blocks I

SQL logic blocks are the HP ALLBASE/4GL constructs which allow the developer to directly enter HP ALLBASE/SQL logic commands. Any developer familiar with **ANSI-standard** SQL can use the HP ALLBASE/4GL interface to HP ALLBASE/SQL immediately.

To create SQL logic blocks, the developer types in SQL commands into a free-format screen. When the commands are **generated**, they are checked for syntax errors. If the commands pass the syntax check, they undergo a process known as **pre-processing**, where the executable form of the SQL logic block is stored **in** the HP ALLBASE/SQL database, in a ready-to-run form. This ensures faster execution than runtime interpretation.

Equivalent functionality is provided for some HP ALLBASE/SQL commands via HP ALLBASE/4GL logic commands. For example, the **FILE *INSERT** command will be translated into the HP ALLBASE/SQL command **INSERT INTO**. However, the translation occurs at runtime, making execution much slower than if it was pre-processed and stored in the database.

Question

Can you create and generate SQL logic blocks before you have created your SQL database?

Answer:

No. The generated form must be stored in the database, which of course must pre-exist. It is certainly possible to define the SQL Logic Blocks without attempting to generate. Attempting to generate will not do any damage, except to your ego! You will simply get an error message indicating that HP ALLBASE/4GL could not connect to the database specified in the **Application Definition** screen within the HP ALLBASE/4GL administrator.

SQL105 - SLIDE: SQL Logic Blocks II

SQL INTERFACE

SQL Logic Blocks

SQL Logic Blocks may contain:

* one SELECT command
or * up to eight other SQL commands

SQL Logic Blocks may reference HP ALLBASE/4GL variables using the standard "*:data_ref*" syntax. *data_ref* may refer to many HP ALLBASE/4GL data objects:

* file record fields
* work area fields
* variables
* calculated items
* constants
* fully qualified screen fields

SQL Logic Blocks cannot reference:

* scratch pad fields
* communication area fields

Host variable references cannot use substrings.

Student Notes

SQL Logic Blocks II

Constraints

If a logic block does **not** contain a **SELECT** statement, it may contain up to eight other commands that generate into SQL stored data base sections. The SQL logic block can also contain additional commands that do not generate into stored database sections. Those commands are **BEGIN WORK, COMMIT WORK, ROLLBACK WORK** and **SAVEPOINT.**

Host Variables

SQL logic blocks can contain references to HP ALLBASE/4GL data items such as file record fields, work area fields, variables, calculated items and constants. Data items whose length can vary at run-time, such as scratch pad fields or communication area fields, cannot be used as their length is not known at generate time.

To reference the data item, it should be included in the SQL logic block with a colon preceding it, and the correct data item prefix. For example, to use a variable, the syntax is:

```
:V-variable_name
```

BEGIN WORK options

HP ALLBASE/4GL allows all of the options to **BEGIN WORK** supported by HP ALLBASE/SQL. These are **RR, Repeatable Read; CS, Cursor Stability; RU, Read Uncommitted;** and **RC, Read Committed.**

Question

If you wanted to refer to the value of a screen field in an SQL logic block, would you use the reference **S-field.screen** or ***S01**?

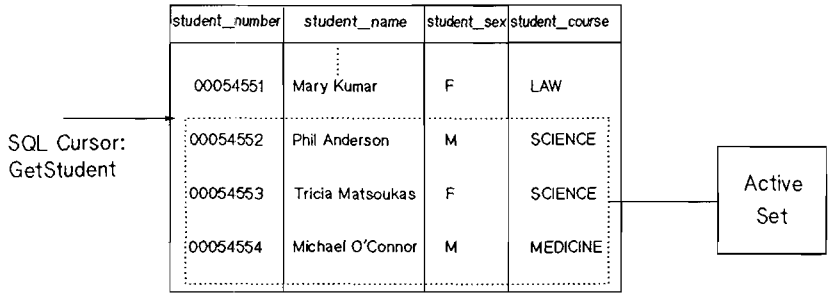
Answer:

You must use S-field.screen, as the length of ***S01** cannot be determined at generate time.

SQL106 - SLIDE: SQL Logic Blocks III

SQL INTERFACE

SQL Logic Blocks: Select Command



To access SQL Tables and Views, we must first define an ACTIVE SET.

In HP ALLBASE/4GL we define an ACTIVE set using the SELECT command within an SQL Logic Block.

The SELECT command defines how we will access the table and which rows within the table will be included.

Student Notes

SQL Logic Blocks III

Active Set

When a **SELECT** command is executed, the rows that are selected by that command are collectively called an **active set**. If a **SELECT** command uniquely specifies a single row, that one row is the complete active set.

Once the select command has defined the **active set**, there is a **cursor** pointing at the first row in that set. To retrieve the record on which the cursor is positioned, the application must issue a **FILE *NEXT** command. That operation will take the row that the cursor is pointing to and copy it from the HP ALLBASE/SQL database into the HP ALLBASE/4GL file buffer.

When enough **FILE *NEXT** commands have been issued to move the cursor to the end of the current active set, an **end-of-file error** will be given if another **FILE *NEXT** is encountered.

Question

Can you think of an SQL select statement that would create the active set shown on the overhead transparency?

Answer:

```
SELECT :tablename FROM owner.tablename
WHERE student_number < 00054555
AND   student_number > 00054552;
```

SQL107 - SLIDE: SQL Logic Blocks IV

SQL INTERFACE

SQL Logic Blocks: Select Command

	number	name	sex	course
	00054551	Mary Kumar	F	LAW
SQL Cursor: GetStudent	00054552	Phil Anderson	M	SCIENCE
	00054553	Tricia Matsoukas	F	SCIENCE
	00054554	Michael O'Connor	M	MEDICINE

SQL Logic Block: GetStudent

```
SELECT :student_table FROM group.student_table
WHERE number > :V-start_number
FOR UPDATE OF number, name, sex, course;
```

When HP ALLBASE/4GL generates the SELECT command it converts it to a DECLARE CURSOR command; (Note that an OPEN CURSOR is automatically performed).

After the SQL logic block containing the SELECT command has been executed, the cursor points to the start of the ACTIVE SET. The first FILE *NEXT command will load the first row of the set into the file record buffer.

The cursor generated by the SELECT command is defined within the database with the same name as the SQL logic block that defines the SELECT command.

Australian Software Operation
sql107

HEWLETT PACKARD

Student Notes

SQL Logic Blocks IV

Cursor

Whenever HP ALLBASE/4GL generates a **SELECT** command, it is converted to a **DECLARE CURSOR** command. The cursor initially points to the beginning of the active set, and is advanced through the active set by the use of the **FILE *NEXT** operation.

Cursor Name

The name of the cursor is the same as the name of the SQL logic block that defined the cursor. In the example on the slide, the name of the logic block was **GetStudent**, and that is the name of the cursor as well. The developer must know the name of the cursor, as operations like deletion and modification of rows must know at which row the cursor is pointing.

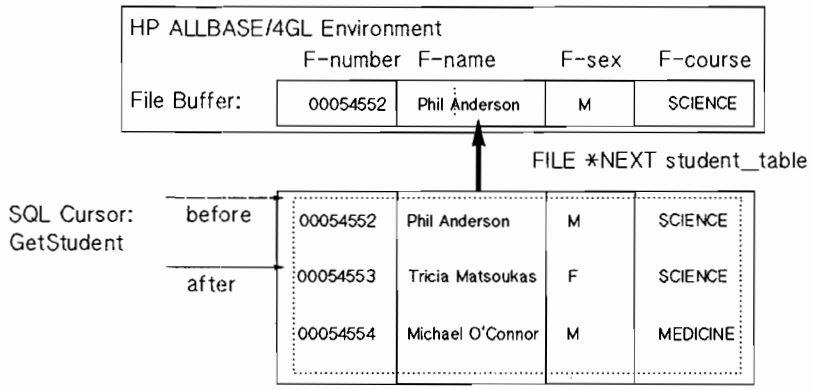
The Example

The example on the slide shows the select command that defines the active set shown on the table. You will notice that there is a **FOR UPDATE OF** clause at the end of the **SELECT** command. This informs HP ALLBASE/SQL that the intention is to modify the values of some of the rows in the active set. This applies a **write lock** on the table, and that later SQL logic blocks can use the **GetStudent** cursor for modification-type commands.

SQL108 - SLIDE: A Simple Application: SQL Read

SQL INTERFACE

SQL Logic Blocks: SQL Read



Once the SQL cursor is positioned on a row, the LOGIC command FILE *NEXT is used to transfer data into its associated file buffer.

The simplest read operation consists of a SELECT command within an SQL logic block, followed by a FILE *NEXT command.

Student Notes

The HP ALLBASE/4GL function **get_student** could contain the following:

```
01  SQL get_student
02  FILE *NEXT student
03  EXIT
```

and the SQL Logic Block **get_student** could contain the following:

```
SELECT :student FROM sqlgrp.student
WHERE student_number > 00054552 AND student_number < 00054554;
```

A Simple Application: SQL Read

To read in a row from an HP ALLBASE/SQL table, an SQL logic block must be executed. The **SELECT** statement in that block positions a cursor before the first row in the active set. A **FILE *NEXT** operation then retrieves the record from the table and places it in the file buffer, where other HP ALLBASE/4GL operations can be performed on the data.

This operation is the equivalent of a **FILE *READ** for an ISAM/KSAM file.

For example, consider the following function **get_student**:

```
01 SQL  get_student
02 FILE *NEXT  student
03 EXIT
```

and the SQL Logic Block **get_student** could contain the following:

```
SELECT :student FROM sqlgrp.student
WHERE student_number > 00054552
AND   student_number < 00054554;
```

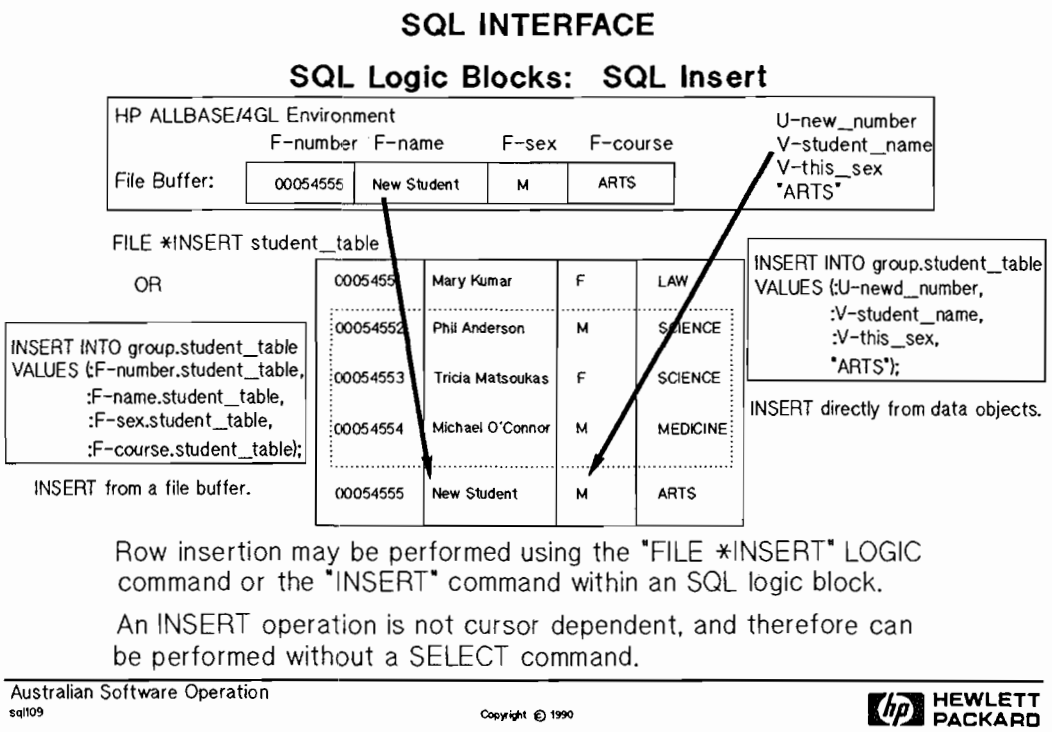
Question

What is the name of the **cursor** opened by the SQL Logic Block?

Answer:

The cursor has the same name as the SQL Logic Block; thus, it is called **get_student**.

SQL109 - SLIDE: A Simple Application: SQL Insert



Row insertion may be performed using the "FILE *INSERT" LOGIC command or the "INSERT" command within an SQL logic block.

An INSERT operation is not cursor dependent, and therefore can be performed without a SELECT command.

Student Notes

A Simple Application: SQL Insert

To insert a record into a table, the developer can use **FILE *INSERT**, or an SQL logic block can be used to enter an **INSERT** command.

Because HP ALLBASE/SQL tables are not stored physically in any particular key order, it does not matter where in the table a new record is inserted. Therefore the **INSERT** command is not required to use a cursor or an active set - there is no need to precede an **INSERT** command with a **SELECT** command.

If the insertion is performed using the **FILE *INSERT** syntax, the current contents of the appropriate file buffer will be inserted into the table. With the HP ALLBASE/SQL **INSERT** syntax, the developer must exactly specify the value for each column. Usually this is done by referring to file buffer elements as host variables, as is done in the left-hand box of the overhead slide. The developer is not restricted to just use file buffer elements; the data items to be inserted can also be literals, constants etc, as shown on the slide.

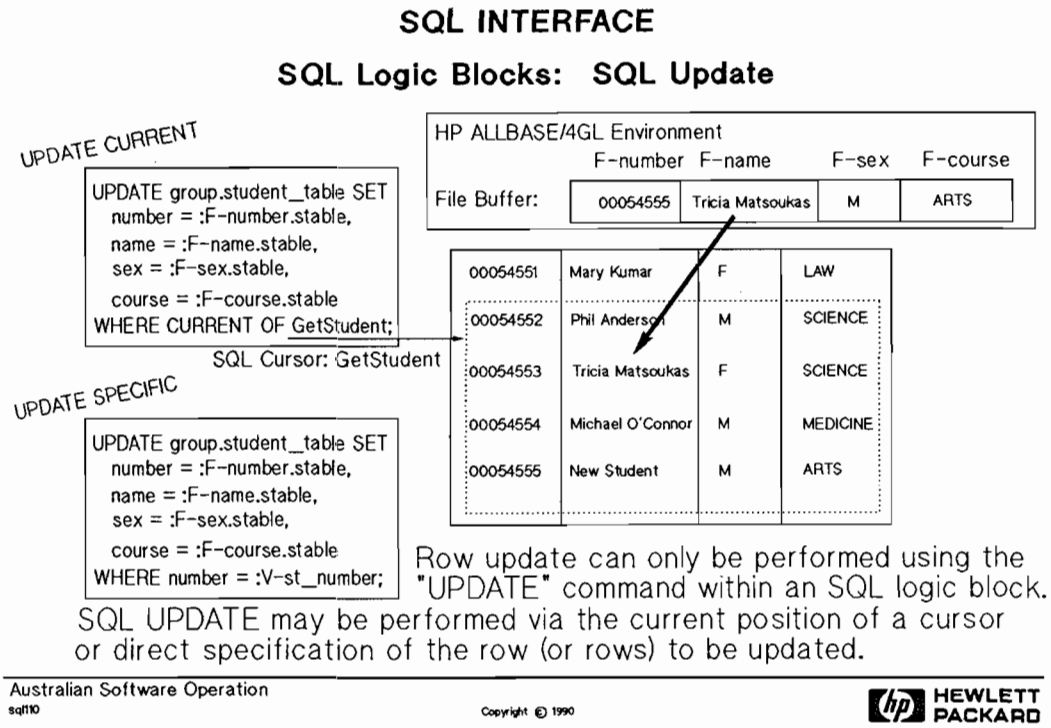
For example, consider the following function **insert_student**:

```
01 SQL  insert_student
02 SQL  commit_work
03 EXIT
```

The SQL Logic Block **insert_student** could contain the following:

```
INSERT INTO sqlgrp.student
VALUES (:F-number.student_table,
        :F-name.student_table,
        :F-sex.student_table,
        :F-course.student_table);
```

SQL110 - SLIDE: A Simple Application: SQL Update



Student Notes

A Simple Application: SQL Update

There is no **FILE** command equivalent to the HP ALLBASE/SQL **UPDATE** command because it is necessary to specify the record to be updated; to do this requires either a reference to the cursor which is pointing to the row of interest, or use of the **WHERE** clause syntax to specify a row.

On the slide, the two different methods are shown. The first logic block uses the cursor defined by the selection command in the **GetStudent** logic block; the second uses a **WHERE** clause to pick one particular row.

If the **WHERE** clause selects more than one row, the update operation will be performed on all the selected rows. This is one situation where HP ALLBASE/SQL is considerably more powerful than ISAM or KSAM.

For example, consider the following process **update_student**:

```
01 MODE *WRITE student_table
02 SQL  get_student
03 FILE *NEXT  student_table
04 SCREEN update_student
05 SQL update_student
06 SQL commit_work
07 EXIT
```

The SQL Logic Block **update_student** could contain the following:

```
UPDATE sqlgrp.student_table SET
       number = :F-number.student_table,
       name = :F-name.student_table,
       sex = :F-sex.student_table,
       course = :F-course.student_table
WHERE CURRENT OF get_student;
```

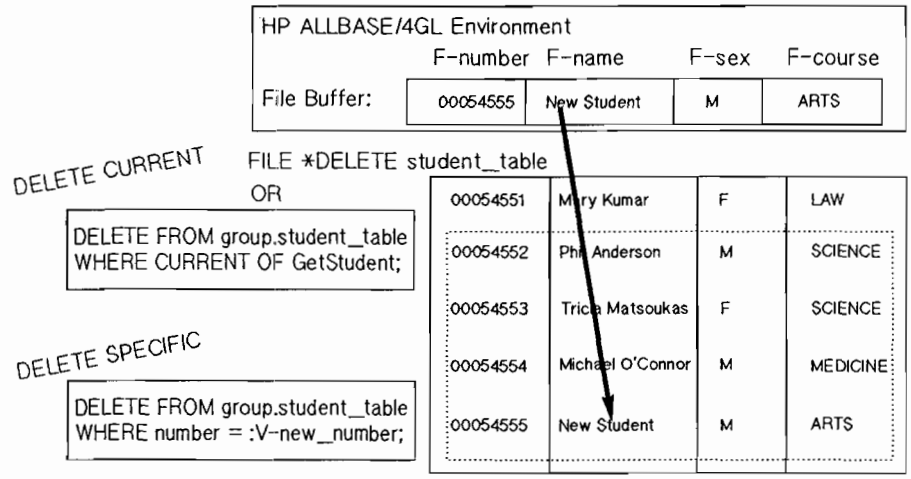
The SQL Logic Block containing the **UPDATE** command must be preceded, within the same HP ALLBASE/4GL process, by an SQL Logic Block containing a **SELECT** command to open the cursor. A **FILE *NEXT** command must follow the command to open the cursor. Why?

Normally, the various commands would be shared between the controlling process and a number of functions.

SQL111 - SLIDE: A Simple Application: SQL Delete

SQL INTERFACE

SQL Logic Blocks: SQL Delete



Row deletion may be performed using the "FILE *DELETE" LOGIC command or the "DELETE" command within an SQL logic block.

Student Notes

A Simple Application: SQL Delete

Three Methods of Deletion

Deletion from a table can be achieved by either using a **FILE *DELETE** command, or by using a **DELETE SQL** command in an SQL logic block. In this, case the deletion can be done via a **cursor** or by a **WHERE CLAUSE.**

If the **FILE** command is used, a **cursor** must have already been defined for the table, and a **FILE *NEXT** operation must have been performed to place the details of the record to be deleted into the file buffer. The **FILE *DELETE** will then delete the row.

When using the SQL syntax, the deletion can be done by a logic block which specifies which row (or rows) are to be deleted by using a **WHERE** clause. Alternatively the deletion can be done via a cursor, in which case there must have been a **SELECT** command performed to position the cursor appropriately.

Deletion Using FILE *DELETE

Consider the following process **delete_student**:

```
01 MODE *WRITE student_table
02 SQL  get_student
03 FILE *NEXT  student
04 SCREEN delete_student
05 FILE *DELETE student_table
06 SQL commit_work
07 EXIT
```

and the SQL Logic Block **get_student** could contain the following:

```
SELECT :student FROM sqlgrp.student
WHERE student_number > "00054552"
AND   student_number < "00054554";
```

Deletion Using a Cursor

Consider the following process **delete_student**:

```
01 MODE *WRITE student_table
02 SQL  get_student
03 FILE *NEXT  student
04 SCREEN delete_student
05 SQL delete_student
06 SQL commit_work
07 EXIT
```

The SQL Logic Block **delete_student** could contain the following:

```
DELETE FROM sqlgrp.student_table
WHERE CURRENT OF get_student;
```

Deletion Without a Cursor Using DELETE

Consider the following process **delete_student**:

```
01 MODE *WRITE student_table
02 SCREEN delete_student
03 SQL delete_student
04 SQL commit_work
05 EXIT
```

The SQL Logic Block **delete_student** could contain the following:

```
DELETE FROM sqlgrp.student_table
WHERE number = :V-new_number;
```

Intentionally blank

SQL112 - SLIDE: A Simple Application: The Four Operations

SQL INTERFACE

A Simple Application: The four simple operations.

SQL READ	**SQL INSERT**
FILE *NEXT tablename A cursor must have previously been defined using "SELECT" within an SQL logic block.	FILE *INSERT tablename OR SQL "INSERT INTO" command No cursor is required.
SQL UPDATE	**SQL DELETE**
SQL "UPDATE ..." command via cursor SQL "UPDATE ..." command by direct specification.	FILE *DELETE tablename OR SQL "DELETE FROM" command via cursor SQL "DELETE FROM" command by direct specification.

Australian Software Operation
sql112

HEWLETT PACKARD

Student Notes

A Simple Application: The Four Operations

Question

1. Can the SQL UPDATE command update more than one row at the same time?

   Answer:

   Yes, if more than 1 row is in the active set or are specified by the **WHERE** clause.

2. Is it necessary to execute an SQL logic block before issuing any sort of DELETE command?

   Answer:

   No, the **WHERE** clause version of the SQL **DELETE** command allows deletion without a **SELECT**.

SQL113 - SLIDE: SQL Logic Blocks: Error Handling

SQL INTERFACE

SQL Logic Blocks: Error Handling

SQL GetStudent ; MESSAGE FileError

If an error is encountered during execution of the SQL logic block the following actions occur.

* SQL logic block execution terminates
* *ERROR receives the text of the error message
* *IOSTATUS receives the error number plus 100,000
* The exception clause is executed if specified.
* If multiple errors are encountered, the additional error message(s) and number(s) are obtained using the SQL EXPLAIN command in an SQL logic block.

Student Notes

SQL Logic Blocks: Error Handling

If an HP ALLBASE/SQL error occurs during execution of an SQL logic block, execution of the logic block terminates. If the developer has specified an optional error condition, control passes to this when an error occurs.

The text of the error message is placed in the communication area field ***ERROR**, and the error number (+100000) is placed in ***IOSTATUS**.

In cases where a single HP ALLBASE/SQL statement has caused more than one error condition, the additional information can be retrieved by executing further logic blocks containing **SQLEXPLAIN** commands. The SQL logic block containing the **SQLEXPLAIN** will continue to return an error condition when executed, until all the errors have been reported.

For example, consider the sample fragment of a logic block:

```
.
.
20 SQL explain
21 IF *IOSTATUS = 0 THEN ENTER 10
22 MESSAGE FileError
23 ENTER 20
.
.
```

The SQL Logic Block **explain** just contains the HP ALLBASE/SQL command **SQLEXPLAIN**; and the message **FileError** displays the contents of ***IOSTATUS** and ***ERROR**.

HP ALLBASE/4GL Errors

In addition to errors generated by HP ALLBASE/SQL, HP ALLBASE/4GL may generate errors during HP ALLBASE/SQL transactions. For example, the database may be inaccessible. In these situations, the errors are in the **60000** range.

SQL114 - SLIDE: SQL Logic Blocks: Generation

SQL INTERFACE

SQL Logic Blocks: Error Handling

SQL logic block generation from the 'Logic Block Details Screen':

* Syntax check
* SELECT converted to DECLARE CURSOR and (implied) OPEN CURSOR
* Select lists and table record layouts replaced by dictionary item definition.
* Generated form stored in S-files.
* SQL commands are stored in the database as a section within the module:

  ownergroup.application_name

SQL logic block generation from the 'Generates Screen':

* Rebuilds the database module with newly generated sections.
* Rebuilds the database module definition file ('.dbm' file).

Student Notes

SQL Logic Blocks: Generation

Syntax

The SQL commands are checked for syntax when the logic block is generated.

If the syntax is incorrect, nothing will be written to the database.

Cursor

SELECT commands are automatically converted into **DECLARE CURSOR** commands, as the **SELECT** does not itself retrieve the data; it merely prepares the table for further operations that may retrieve or delete or modify.

Replace Table Names

Any named HP ALLBASE/4GL data items are replaced by their dictionary definitions, so that the HP ALLBASE/SQL dictionary has all the data it needs.

Generation

The generated form of the SQL logic block is stored in the S-files, and a executable stored section is placed in the database as well. When the section is stored in the database, the SQL ownername (as defined in the **Application Definition** screen) is prepended to the name of the stored section.

The HP ALLBASE/SQL commands **BEGIN WORK, COMMIT WORK, ROLLBACK WORK** and **SAVEPOINT** do not generate into stored database sections.

The Database Module File

If generated from the **Generates** screen in the developer environment, a **Database Module File** will be created.

On HP-UX, this will have the same name as the HP ALLBASE/4GL application but will have the extension **.dbm** file; it will be created in the S-file directory.

On MPE XL, it will have the same name as the HP ALLBASE/4GL application but will be created in the **HP4DBM** group.

The **Database Module File** is important when porting applications to the HP ALLBASE/4GL runtime environment. It contains the stored HP ALLBASE/SQL components of the application (generated sections from SQL Logic Blocks and SQL Table definitions) which can be loaded into the target database. Unlike the developer environment, the runtime environment does not have the ability to generate stored sections or create tables.

Module: The HP ALLBASE/SQL Interface II

Instructor Notes

SQL200 - SLIDE: Module Objectives

SQL INTERFACE

SQL Select Lists

Reports with SQL

Data Type Support

SQL Interface

OBJECTIVES:

Upon completion of this module, the student will be able to:

1. Describe the purpose, and definition of SQL Select Lists.
2. Explain how HP ALLBASE/4GL reports can be integrated into the HP ALLBASE/SQL relational environment.
3. Describe the HP ALLBASE/SQL data types support provided by HP ALLBASE/4GL.
4. List the operations supported by the HP ALLBASE/4GL SQL interface.

Student Notes

Module Objective

This module will introduce the students to the interface to HP ALLBASE/SQL provided by HP ALLBASE/4GL.

Student Objectives

At the end of this module, each student will be able to do the following:

- List the different components of the HP ALLBASE/4GL to HP ALLBASE/SQL interface.
- Describe each of the components of the interface.
- Build a simple database using **isql** and a command file.
- Describe the differences between the ISAM/KSAM file interface and the SQL interface.
- Use the interface to copy the data from an ISAM/KSAM file to an SQL table.

Resource Allocation

Equipment

The following items are required for this module:

- Instructor notes.
- Student notes.
- Slides.
- A laboratory worksheet.
- Access to an HP 3000 Series 900 machine or an HP 9000 Series 800 or HP 9000 Series 300 machine with HP ALLBASE/4GL installed.
- The sample database **PartsDBE**.
- The complete set of HP ALLBASE/4GL manuals.

Time

This module should take approximately 2 hours; allow 1 hour for the theory and 1 hour for the laboratory work.

Pacing

Read through the slides reasonably quickly.

Preparation

The demonstrations in this module require the sample database **PartsDBE**. This should exist within the **hp4sql** directory on HP-UX or the **HP4SQL** group on MPE XL.

On HP-UX, copy all of the files from the directory **/usr/lib/allbase/hpsql/sampledb** to the **./HP4GL/hp4sql** directory. Create the database by typing:

```
cd ./HP4GL/hp4sql
isql < creajob
```

On MPE XL, copy the file **CREASQL.SAMPLEDB.SYS** to the **HP4SQL** group in the current account. Edit the file and remove all lines from the line containing:

```
FCOPY FROM=READSQL.SAMPLEDB.SYS;TO=READSQL;NEW
```

down to, but not including, the last line. Create the database by typing:

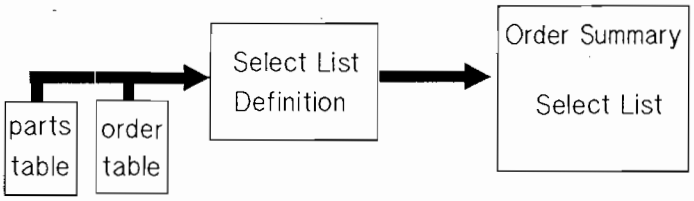
```
chgroup hp4sql
stream creasql
```

Intentionally blank

SQL201 - SLIDE: SQL Select Lists

SQL INTERFACE

SQL Select Lists



An HP ALLBASE/4GL Select List defines a virtual file that can be used within an HP ALLBASE/4GL application.

* Select List definition defines the field structures, record layout, and file definition for this virtual file.
* A Select List may define a subset of single SQL table, or it may join together columns from many SQL tables.
* A file record buffer is maintained for every Select List.
* A Select List is like an SQL View.
* Select Lists cannot be used with ISAM/KSAM files or other files.

Student Notes

SQL Select Lists

Demonstration

Ask the students to gather around the terminal while you demonstrate the use of **isql**.

Run HP ALLBASE/4GL and go into the administrator application. Press the function keys `System Keys`, `More Keys` `ISQL` to run **isql**. Connect to the sample database called **PartsDBE** by typing:

```
connect to 'PartsDBE';
```

or on MPE XL,

```
connect to 'PartsDBE.hp4sql';
```

Enter a **SELECT** command to display the entire contents of the table **system.table**, as follows:

```
select * from system.table;
```

Explain that * indicates to HP ALLBASE/SQL that we wish to select all of the columns.

Now select just two or three of the columns from the table by explicity listing them, as follows:

```
select name, owner from system.table;
```

The list of columns **name, owner** is essentially a **select list**.

If we call the list of columns **FOO**, the **SELECT** command would become:

```
select FOO from system.table;
```

This is precisely how **SQL Select Lists** are referenced within HP ALLBASE/4GL.

Exit **isql** by typing the following:

```
commit work;
exit;
```

This will return control to HP ALLBASE/4GL. Exit HP ALLBASE/4GL by pressing the **Exit** function key.

Virtual Table or View

SQL select lists are the HP ALLBASE/4GL construct for manipulating columns of HP ALLBASE/4GL tables. An **SQL Select List** is a **virtual table** or in HP ALLBASE/SQL parlance, a **view**.

SQL Select Lists are used within HP ALLBASE/4GL in **SQL Logic Blocks** and can replace the name of an **SQL Table** in a **SELECT** command.

A **SELECT** command to open a cursor for the table **SYSTEM.TABLE** could be the following:

```
SELECT :table FROM system.table;
```

where **:table** is the reference to the HP ALLBASE/4GL definition of the **SQL Table** called **table**. An **SQL Select List** called **foo** containing just two of the fields could replace the reference to the table, as follows:

```
SELECT :foo FROM system.table;
```

The **SQL Select List foo** could be defined as follows:

```
name,
owner
```

As you can see, the HP ALLBASE/4GL interface maintains consistency with standard SQL syntax.

Structure

An **SQL Select List** consists of HP ALLBASE/SQL table column names; these are implicitly associated with the HP ALLBASE/4GL field specifications corresponding to the column names, although the developer can override the default values.

Thus, an **SQL Select List** is similar to an HP ALLBASE/4GL record layout; basically it defines the structure of the **virtual file.**

Because of the similarity with HP ALLBASE/4GL files, an SQL Select List cannot have the same name as an HP ALLBASE/4GL file or record layout.

File Buffer

HP ALLBASE/4GL maintains a file buffer for every **SQL Select List.**

Subset of a Table

A select list can be created to make a subset of an existing table. This may be useful if the developer wants to restrict access to some columns that contain sensitive information or just wishes to display less information than is contained in the entire table.

A Join between Tables

A select list can be used to join existing tables together into a single table. This technique is often used for enquiry functions and greatly simplifies reporting on multiple tables.

Only with HP ALLBASE/SQL Tables

SQL Select Lists can only be referenced in **SQL Logic Blocks**; therefore they cannot be used with ISAM/KSAM files, serial files or HPTurboIMAGE data sets.

SQL202 - SLIDE: SQL Select Lists

SQL INTERFACE

SQL Select Lists

Definitions of Select List Columns can be:

* taken directly from a base table column
  or
* computed from an expression involving:
  - constants
  - specific HP ALLBASE/4GL variables
  - base table column values
  or
* computed from a group of base table column values with an aggregate function (AVG, MAX, MIN, SUM, and COUNT).

* Cannot contain duplicate column names
* Must contain a COMMON COLUMN when joining tables
* Must contain a fully qualified filename to avoid ambiguous column specifications when joining tables.

Australian Software Operation
SQL202

HEWLETT PACKARD

Student Notes

SQL Select Lists

An Expression

Constants and other column names can be included in the definition of a new column.

HP ALLBASE/4GL items can be referenced by using the : prefix. An example is shown in the previous slide.

Aggregate Functions

Calculated columns in a select list can use the built-in HP ALLBASE/SQL **aggregate** functions, such as **MAX, MIN, SUM, COUNT** and **AVG.** If **aggregate functions** are used, the select list may not normally contain normal columns because these functions return a single value, whereas a column will generally contain multiple values.

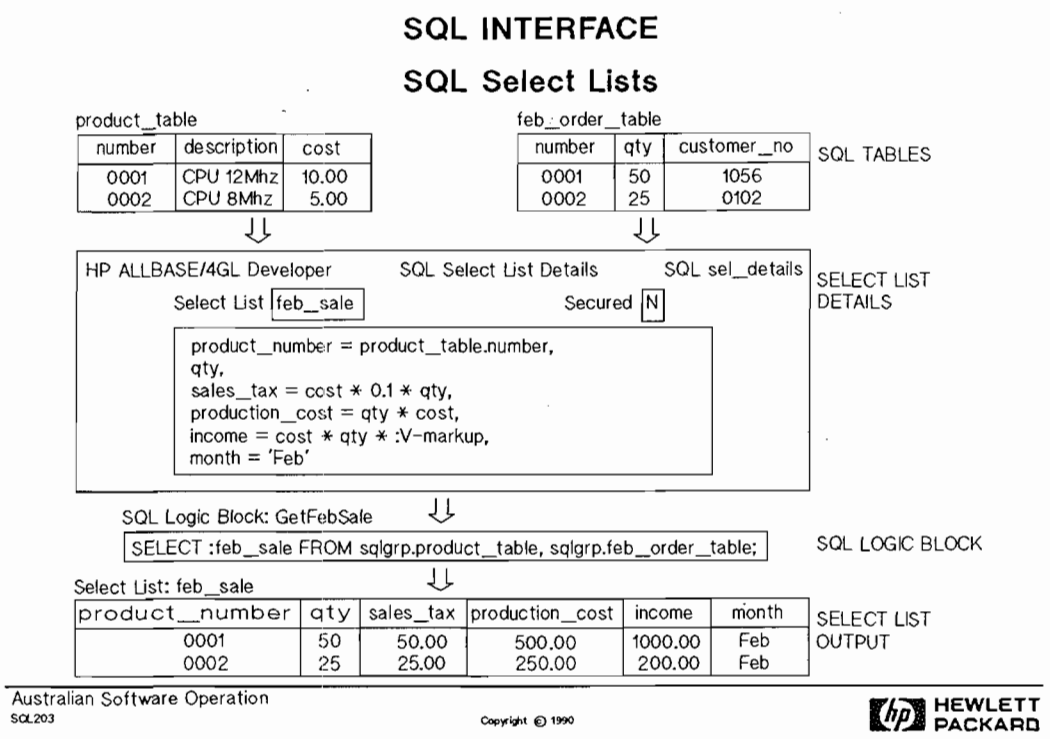
However, the **GROUP BY** clause does allow this.

Joining Tables

Tables to be joined together into a select list must have a common column. This requirement is identical to that for automatic file linkages from ISAM/KSAM files.

When joining tables, columns common to a number of tables must be fully qualified with the filename to avoid ambiguities. An example of qualification is shown in the previous slide with the reference **product_number = product_table.number.** The **product_table** qualification indicates that the **number** column is to be taken from the table **product_table.**

SQL203 - SLIDE: SQL Select Lists



Student Notes

```
select PURCHDB.PARTS.PARTNAME, PURCHDB.PARTS.SALESPRICE,
       PURCHDB.SUPPLYPRICE.PARTNUMBER, PURCHDB.SUPPLYPRICE.UNITPRICE
 from  PURCHDB.PARTS, PURCHDB.SUPPLYPRICE
where  (PURCHDB.PARTS.PARTNUMBER = PURCHDB.SUPPLYPRICE.PARTNUMBER);
```

SQL Select Lists

In this slide, the **SQL Tables** called **product_table** and **feb_order_table** are to be joined. They have a common column called **number**; this column will be used for the join.

The **SQL Select List** called **feb_sale** includes the following columns:

- **product_number**, taken from the **number** column in the table **product_table**.
- **qty**, taken from the table **feb_order_table**.
- **sales_tax**, which is computed from **cost** from the table **product_table** and **qty**.
- **production_cost**, which is taken from **qty** and **cost**.
- **income**, which is taken from **cost**, **qty** and the HP ALLBASE/4GL variable **V-markup**.
- **month**, which has the literal value **Feb**.

The select list **feb_sale** is referenced in the **SQL Logic Block** called **GetFebSale**, using the following syntax:

```
SELECT :feb_sale FROM sqlgrp.product_table,sqlgrp.feb_order_table;
```

Demonstration

Ask the students to gather around a terminal as you demonstrate the concept of **joins** using HP ALLBASE/QUERY.

Start up HP ALLBASE/4GL and go into the adminstrator application. Press `System Keys` `More Keys` `ALLBASE QUERY` to initiate HP ALLBASE/QUERY.

Connecting to the database

Connect to the **PartsDBE** database; on HP-UX, enter the name **./hp4sql/PartsDBE**; on MPE XL, enter the name **partsdbe.hp4sql**.

Selecting tables

Select the menu item **Define and View Report**. This will display a list of the tables in the database. Using the (Tab) key or arrow keys, highlight the tables **PURCHDB.PARTS** and **PURCHDB.SUPPLYPRICE** in turn; select each table by pressing (Return) while the name is highlighted. Note that on MPE XL, an extra (Return) is required after using the arrow keys to move around, just as with HP ALLBASE/4GL.

Selecting columns

Press `Next Step`. This will display all of the columns in the two tables. Select **PARTNAME** and **SALESPRICE** from **PURCHDB.PARTS**. Select **PARTNUMBER** and **UNITPRICE** from **PURCHDB.SUPPLYPRICE**.

Displaying the data

Press `Display Data` to display the report. Note that the columns are displayed in the order **PARTNAME**, **SALESPRICE**, **PARTNUMBER** and **UNITPRICE**.

Changing the layout and re-displaying the data

Press `Edit Keys` `Layout Editing` to bring up the **Edit Layout** screen. Change the display order of the columns by entering **2** against **PARTNAME**, **4** against **SALESPRICE**, **1** against **PARTNUMBER** and **3** against **UNITPRICE**. Press `Display Data` to re-display the report.

Changing to SQLACCESS mode

On HP-UX, type **Control-w**; on MPE XL, type **Control-w** (Return). This brings up the prompt **Enter HP ALLBASE/QUERY Command.**. Enter:

```
SET ACCESS SQLACCESS
```

The actual SQL command

Now press `Previous Step`. This will display the actual SQL command required to produce the report. The command should be:

```
select PURCHDB.PARTS.PARTNAME, PURCHDB.PARTS.SALESPRICE,
       PURCHDB.SUPPLYPRICE.PARTNUMBER, PURCHDB.SUPPLYPRICE.UNITPRICE
 from  PURCHDB.PARTS, PURCHDB.SUPPLYPRICE
where  (PURCHDB.PARTS.PARTNUMBER = PURCHDB.SUPPLYPRICE.PARTNUMBER);
```

Point out that both tables have a common column (PARTNUMBER) and that the join requires the condition that the value in the common column is identical for each table.

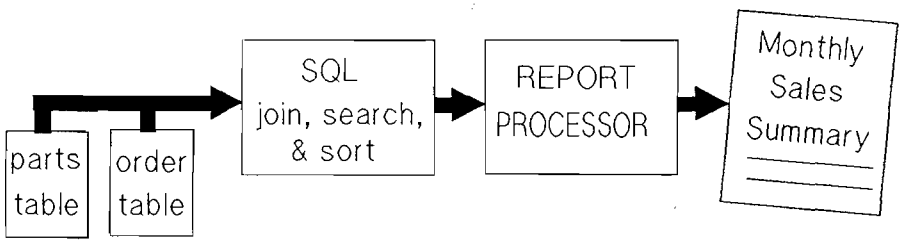
Exit HP ALLBASE/QUERY by pressing `System Keys` `More Keys` `Exit`. This will return you back to HP ALLBASE/4GL.

If time permits, press the function key `ISQL` to start **isql**. Connect to the **PartsDBE** database and manually enter the above command to observe the results. Exit HP ALLBASE/4GL in the usual way.

SQL204 - SLIDE: SQL with Reports I

SQL INTERFACE

SQL with Reports



SQL is an ideal tool for generating reports on large databases.

Although HP ALLBASE/4GL has powerful sorting and searching features built into it, the HP ALLBASE/SQL language provides a far more efficient method of delivering ordered data to the report formatting stage of the report processor.

Student Notes

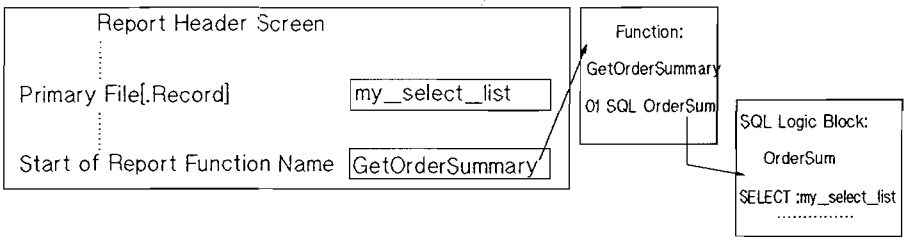
SQL with Reports I

When reporting on an HP ALLBASE/SQL database, SQL's sorting and searching capabilites should be used. HP ALLBASE/SQL has the ability to manipulate its own data more efficiently than HP ALLBASE/4GL can; in-built selection will minimise the amount of data passed back to HP ALLBASE/4GL, minimise the amount of data to be sorted and in-built sorting will also be faster.

SQL205 - SLIDE: SQL with Reports II

SQL INTERFACE

SQL with Reports



In order to report on data in your SQL database you must use a select statement to specify the required data from the relevant table(s), or views.

You must use the 'Start of Report' function to invoke an SQL logic block containing the SELECT statement above. This statement may also perform selection and sorting.

The name of the select list or base table is entered as the primary report file.

Student Notes

SQL with Reports II

Start of Report Function

HP ALLBASE/4GL can report on an HP ALLBASE/SQL table or an SQL select list. In either case, the **before report function** must contain an HP ALLBASE/4GL **SQL** command that executes an SQL logic block, which in turn peforms a **SELECT** command on the table or list.

Sorting & Searching

The **SELECT** command will perform all the necessary selection, and an **ORDER BY** clause can be added to perform any sorting. Thus the **SELECT** will define an **active set** of all the valid records on which the report will operate.

For example, consider the **SELECT** command in the **GetFebSale** SQL Logic Block as shown in the previous slide. Suppose we want a list of all values for products with numbers between **0005** and **0010** inclusive, sorted by quantity. Adding a **WHERE** clause for selection, and an **ORDER BY** clause for sorting, we get:

```
SELECT :feb_sale FROM sqlgrp.product_table,sqlgrp.feb_order_table
WHERE number > 0005 AND number < 0010
ORDER BY qty;
```

Primary File

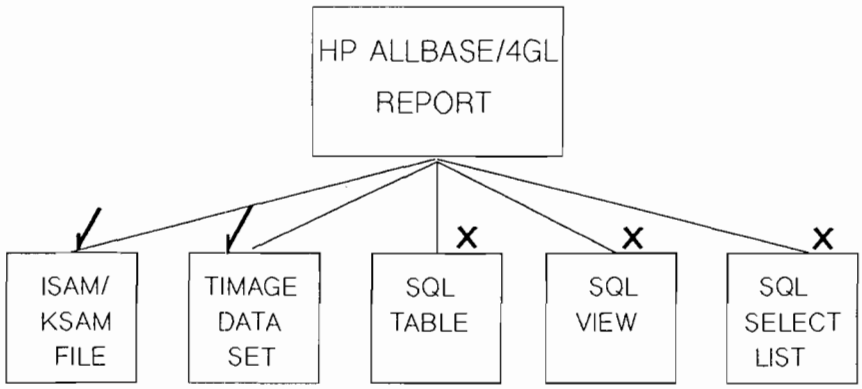
The name of the select list or base table is entered on the header screen as the name of the primary file. When the report is run, the report processor will issue an implicit **FILE *NEXT** operation, moving the cursor through the active set of records.

SQL206 - SLIDE: SQL with Reports III

SQL INTERFACE

SQL with Reports

It is not permitted to specify a file linkage to an SQL table, view, or select list.



Australian Software Operation
SQL206

HEWLETT PACKARD

Student Notes

SQL with Reports III

Automatic File Linkage

This automatic feature can only link ISAM/KSAM files and HP TurboIMAGE data-sets into reports. ISAM/KSAM or HP TurboIMAGE information can be linked into an HP ALLBASE/SQL-based report, but SQL tables or select lists can never be linked into reports using this feature.

Instead, use a join, implemented via an SQL Select List, to link information from other SQL tables into SQL reports.

In this way, the HP ALLBASE/4GL report processor does not even need to be aware that it is reporting on several tables; it only sees one large virtual table.

SQL207 - SLIDE: Data Type Support; Data Type Mapping

SQL INTERFACE

Data Type Support: Data Type Mapping

HP ALLBASE/4GL creates SQL tables that contain only CHAR and packed DECIMAL data types. However, all SQL data types can be accessed in existing tables.

Tables created by HP ALLBASE/4GL

HP ALLBASE/4GL	HP ALLBASE/SQL
X, A, U, K, Q →	CHAR(n)
N, S →	DECIMAL(p,s)
D, T →	CHAR(8)

Existing Tables

HP ALLBASE/SQL	HP ALLBASE/4GL
CHAR(n) VARCHAR(n) →	X
FLOAT DECIMAL(p,s) INTEGER SMALLINT →	S

Australian Software Operation
SQL207

HEWLETT PACKARD

Student Notes

Data Type Support: Data Type Mapping

Created from 4GL

If an HP ALLBASE/SQL table is created from HP ALLBASE/4GL, only two data types will be created, namely **CHAR** and **packed DECIMAL.**

Reading Existing Tables

Although all HP ALLBASE/4GL data items will be converted to these two types, all the other HP ALLBASE/SQL data types can be read in from existing HP ALLBASE/SQL tables.

The slide shows how HP ALLBASE/4GL data item edit codes map onto the data types within HP ALLBASE/SQL, and what data types should be used from HP ALLBASE/4GL to read existing HP ALLBASE/SQL tables.

Notice that all HP ALLBASE/SQL numeric types map onto the HP ALLBASE/4GL S or **Signed Numeric** type. Beware of setting up an **integer index** in HP ALLBASE/SQL (using a schema file) and defining an index for the table within HP ALLBASE/4GL. HP ALLBASE/4GL will treat the index as being **floating point**, leading to performance degradation. Instead, omit indexes from the HP ALLBASE/4GL definition.

Binary Data Types

Since the B.02 release on MPE XL 2.1, HP ALLBASE/4GL supports binary data types. This means that HP ALLBASE/4GL can now read and write binary data (integer, IEEE floating point, packed decimal) in SQL tables. Binary data types will be supported on HP-UX with the B.03 release in late 1990.

SQL208 - SLIDE: Data Type Support: NULL Value Support

SQL INTERFACE

Data Type Support: NULL Value Support

HP ALLBASE/4GL:

* Supports detecting, reading and writing null values in SQL tables.
* Maintains a null indicator variable with each field in a select list or record layout used with an HP ALLBASE/SQL table.
* Tests the status of the null indicator variable with the IF command.

```
EXAMPLE:    IF F-salary.salespeople *NULL
                THEN MESSAGE SalaryNotChecked
```

* Sets the indicator variable with the MOVE command.

```
EXAMPLE:    MOVE *NULL F-salary.salespeople
```

Australian Software Operation
SQL208

HEWLETT PACKARD

Student Notes

Data Type Support: NULL Value Support

What is NULL?

HP ALLBASE/SQL makes a distinction between a field being blank, and the field being **NULL**. Consider the case where a new row is added to an SQL table. The table has 5 columns, but the **INSERT** statement only specifies values for the first 4. It may be necessary to record that the 5th field has not been filled, so a blank value would not be appropriate. In this situation, the developer may wish that a **NULL** value be written to the 5th column of the table. Whether a column in a table is allowed to contain **NULL** values is determined when the definition of the table is created.

Null Support

The reading, writing and detection of **NULL** values is supported in HP ALLBASE/4GL. Any file buffer for an HP ALLBASE/SQL table or select lists has the capacity to store a field as **NULL**. To test if a a file buffer element is **NULL**, you can use an **IF** statement and use ***NULL** as the test operator.

Additionally you can **MOVE *NULL** into a field, usually done just prior to writing that field out to an HP ALLBASE/SQL database.

Table Creation

The issue of **NULL** values is yet another reason why tables in the database should be created via the schema file using **isql** rather than from within HP ALLBASE/4GL. To allow for **NULL** value support, HP ALLBASE/4GL automatically sets each column in the table to allow for **NULL** values. This may not be necessary.

By defining the table via the schema file, the developer can allow **NULL** value support for just the columns which need it, saving space in the database.

SQL209 - SLIDE: SQL Locking I

SQL INTERFACE

SQL Locking I

All SQL transactions initiated from HP ALLBASE/4GL are subject to to the normal SQL locking mechanisms.

Also: COMMIT AND ROLLBACK

- release all current locks
- close any defined cursors

TRANSACT *END => COMMIT WORK
TRANSACT *UNDO => ROLLBACK WORK
TRANSACT *BEGIN => BEGIN WORK

Student Notes

SQL Locking I

Locking with SQL is very different from locking for other data managers. SQL only allows **table locking** and **page-level locking; record locking** is not supported.

HP ALLBASE/4GL performs no special locking of HP ALLBASE/SQL tables but uses the facilities provided by HP ALLBASE/SQL.

If a **FOR UPDATE OF** clause is used with a **SELECT** command, the active set that is created by that select will be locked to that user.

When a **COMMIT** or **ROLLBACK** command is executed, all currently held locks will be released; additionally, any cursors that are defined will be lost. Note that the **TRANSACT *END** and the **TRANSACT *UNDO** commands are equivalent to the **COMMIT WORK** and **ROLLBACK** commands respectively.

Also note that the **TRANSACT *BEGIN** command is equivalent to an implicit **BEGIN WORK** command; any implicit **BEGIN WORK** issued by HP ALLBASE/4GL invokes the default **Repeatable Read** locking mode.

SQL210 - SLIDE: SQL Locking II

SQL INTERFACE

SQL Locking II

SQL only supports table locking and page-level locking. Record locking is not supported.

PAGE LEVEL LOCKING

At least ONE page will be locked; pages are 4096 bytes.

TABLE LOCKING

The HP ALLBASE/SQL command LOCK TABLE can be included in an SQL Logic Block; the entire table will be locked.

LOCKING MODES

The DEFAULT LOCKING MODE is specified when a table is first defined.

* Public - allows multiple read and write access.
* Publicread - multiple read access, exclusive write access.
* Private - exclusive read and write access.

Australian Software Operation
SQL210

HEWLETT PACKARD

Student Notes

SQL Locking II

As mentioned in the previous slide, SQL only allows **table locking** and **page-level locking.**

Page Level Locking

The minimum lock on an SQL database is **one page**; pages are **4096 bytes.**

More pages may be locked, depending on the size of the rows in the table. For example, if each row (or record) in the table contains 5000 bytes, at least two pages will be locked whenever a lock is applied. If the row size is small and the table contains few rows, one lock may lock the entire table.

Table Locks

HP ALLBASE/4GL also supports the HP ALLBASE/SQL command **LOCK TABLE**, which can be directly inserted into an SQL Logic Block. This command will lock the entire table.

Table Locking Modes

Locking is also affected by the **default locking mode** which is specified when the table is first defined. This can be one of three values:

- Public

  Allows multiple read access and multiple write access.

- Publicread

  Allows multiple read access, but requires exclusive access for writing.

- Private

  Requires exclusive access for reading and writing.

Publicread tables allow the highest amount of concurrency. **Public** is the most useful, and is provided by default in HP ALLBASE/4GL.

SQL211 - SLIDE: SQL Locking III - BEGIN WORK

SQL INTERFACE

SQL Locking III – BEGIN WORK Locking Modes

HP ALLBASE/4GL supports all of the BEGIN WORK locking modes.

* BEGIN WORK RR

  Repeatable Read locks all pages in the query access path

* BEGIN WORK CS

  Cursor Stability only locks the current page

* BEGIN WORK RC

  Read Committed only locks the current record while it is being accessed

* BEGIN WORK RU

  Read Uncommitted does not place any locks

Student Notes

SQL Locking III - BEGIN WORK locking modes

HP ALLBASE/4GL issues an implicit **BEGIN WORK** whenever an SQL Logic Block is executed or it encounters a **TRANSACT *BEGIN** command. All implicit **BEGIN WORK** commands use the default **repeatable read** mode. This is also true if a **BEGIN WORK** command is used in an SQL Logic Block.

HP ALLBASE/4GL supports the other modes supplied by HP ALLBASE/SQL which are **cursor stability, read committed** and **read uncommitted** by use of the **CS, RC** and **RU** options to the **BEGIN WORK** command.

Repeatable Read RR

Repeatable Read guarantees all query results to be internally consistent but locks all pages in the active set.

Cursor Stability CS

Cursor Stability guarantees that the record that was last accessed can be updated without fear of it changing, but other records may change. Locks are only held on the current page.

Read Committed RC

Read Committed guarantees that all data read has been committed. This mode locks only the current record while it is being accessed.

Read Uncommitted RU

Read Uncommitted will read uncommitted data and does not place any locks.

These isolation levels govern the level of locking a transaction applies to a database. Each level balances user concurrency against query consistency in a different way. In the list above, query consistency drops for each level down.

Newsletter Article

For further information, the students should read the article in Australian Software Operation Newsletter 15 entitled **Using HPSQL Locking Levels in HP ALLBASE products**, by Alex Nosiara.

SQL212 - SLIDE: SQL Locking IV - Locking Strategies

SQL INTERFACE

SQL Locking IV – Locking Strategies

1. In general, DON'T lock around data-entry screens!

   Don't leave the table locked while waiting for the data-entry operator to finish their coffee.

2. Use PRE-READS.

   Save the data when first read from the table and release the lock. When the screen is committed, read the table again to check for changes.

3. Implement logical record-locking in HP ALLBASE/4GL

   Add a 'flag' column to the table. Each '4GL application will check the flag to see if the row is already locked.

Australian Software Operation
SQL212

HEWLETT PACKARD

Student Notes

SQL Locking IV - Locking Strategies

Poor locking design can have disastrous consequences for a multi-user application.

For example, consider an OLTP application where the end-user is required to enter a search number and the application retrieves the appropriate information and displays it on the screen ready for updating.

Imagine the problems if a lock is applied to the table for the duration of the update of the row, in other words, the lock is applied all the time the data-entry screen remains uncommitted. If the end-user decides to take lunch with the screen half finished, the entire table could be locked for a long time and no other end-users could do any work.

The number one rule of locking is: **"Don't lock around data-entry screens!"**

Also beware of **query-type** messages and reports sent to the screen; they can cause similar problems.

Pre-reads

A much better strategy is to **pre-read** the row from the table. A lock would be applied while reading the row but it would be released immediately by using a **COMMIT WORK** command. The data is then saved away into a work-area. The end-user is then able to take as long as necessary with data-entry without impacting any other users of the database.

When the screen is committed, the row would be read once more and the contents compared against the contents read the first time. If they are the same, the row can be updated. If they differ, the user should be warned that somebody else has modified the row and the data must be re-entered.

This scheme provides for good concurrency in a multi-user environment. It could be very annoying for the end-user however, if they are frequently asked to re-enter transactions because somebody else has modified the same row. The degree of frustration could be minimised if the application is clever enough to re-use the data that has just been entered.

Logical Record Locking

Although HP ALLBASE/SQL does not support **record locking** or, in other words, just locking one row, it is possible to simulate this from HP ALLBASE/4GL.

This scheme requires that each table have **flag** in a column which indicates whether the row is locked or unlocked. Whenever an application reads a row, it actually performs a **read-for-update** and sets the flag to indicate that the record is now locked. During the period of the **read-for-update**, the row (and possibly many others) is locked by standard SQL page-level locking but the lock is released immediately the update is finished.

Any other application which tries to read the row will detect that the row is **logically locked** by checking the value of the flag. It will then return a message to the end-user indicating that the record is locked to another user. It may even be possible to store the name of the end-user who has the record locked, making it possible to report to the second end-user the identity of the end-user who has the lock.

This scheme makes the assumption that all applications reading the table will be **well-behaved** and will check the value of the flag. It also reverts to the notion of locking around a data-entry screen. This is not as disastrous as the previous scenario; previously, multiple rows would be locked until the screen is committed whereas using this scheme, only the row in question would be locked.

Intentionally blank

SQL213 - SLIDE: Implicit SQL Commands

SQL INTERFACE

Implicit SQL Commands

The following SQL commands are automatically performed by HP ALLBASE/4GL.

CONNECT
- Base table creation and deletion
- SQL logic block generation
- SQL command execution

RELEASE
- At end of session.

OPEN
- Execution of select command

CLOSE CURSOR
- COMMIT WORK, ROLLBACK WORK, TRANSACT END, TRANSACT UNDO

BEGIN WORK
- Execution of SQL logic block
- TRANSACT *BEGIN

COMMIT WORK
- TRANSACT *END

Australian Software Operation
SQL213

HEWLETT PACKARD

Student Notes

Implicit SQL Commands

Following the fourth generation mentality, HP ALLBASE/4GL performs some HP ALLBASE/SQL commands implicitly, whenever they are required.

For example, a **CONNECT** command will be issued when the developer creates or deletes a base table, generates an SQL logic block or first accesses the database.

Other commands are also executed automatically; see the slide for more information.

SQL214 - SLIDE: Invalid SQL Commands

SQL INTERFACE

Invalid SQL Commands

The following commands cannot be used in an SQL logic block.

BEGIN DECLARE SECTION,
BULK,
CLOSE (CURSOR NAME),
CONNECT TO (DBEnvironment name),
DECLARE (cursor name),
DESCRIBE (command name),
INTO (cursor name),
END DECLARE SECTION,
EXECUTE IMMEDIATE,
FETCH,
INCLUDE (SQLCA or SQLDA),
INTO (clause with SELECT command),
OPEN (cursor name),
PREPARE,
RELEASE,
START DBE,
STOP DBE,
WHENEVER

Student Notes

Invalid SQL Commands

Because some of the interaction between HP ALLBASE/4GL and HP ALLBASE/SQL interaction is performed automatically by HP ALLBASE/4GL, the developer is prevented from using certain HP ALLBASE/SQL commands, as these commands may disturb the interface.

The commands on the slide are either performed already by HP ALLBASE/4GL without the need for them to be specified explicitly, or they are too powerful to allow the developer to use at any point in an application. For example, you would not want to allow a developer to issue a **STOP DBE** command at any time.

SQL215 - SLIDE: The FILE Command

SQL INTERFACE

Logic Commands

FILE

FILE { *BUFFER | *CLOSE | *DELETE | *INSERT | *NEXT } file_ref[;command]

Student Notes

The FILE Command

1. The FILE command

The only options for the HP ALLBASE/4GL **FILE** command which can be used with HP ALLBASE/SQL tables are:

- *INSERT
- *NEXT
- *DELETE
- *CLOSE
- *BUFFER

These have all been covered in the previous slides.

SQL216 - SLIDE: Differences from the ISAM/KSAM Interface

SQL INTERFACE

Differences from the ISAM/KSAM interface

1. No reformat for SQL Tables.
2. SQL Select Lists and SQL Logic Blocks.
3. The FILE command – only a small number of arguments allowed with SQL.
4. Reports – no need for HP ALLBASE/4GL sorting, selection and file linkages.

Student Notes

Differences from the ISAM/KSAM Interface

There are a number of differences in the interface to HP ALLBASE/SQL from the interface to ISAM/KSAM.

1. Reformat

   HP ALLBASE/SQL tables cannot be reformatted. If the structure changes, the developer (or administrator) must unload the data using the **isql unload** command, delete the tables, re-create them using the new definitions, and re-load the data using the **isql load** command.

2. SQL Select Lists & SQL Logic Blocks

   ISAM/KSAM files cannot use these powerful HP ALLBASE/4GL items; similar functionality is provided by the HP ALLBASE/4GL **FILE** command.

3. The FILE command

   The only options for the HP ALLBASE/4GL **FILE** command which can be used with HP ALLBASE/SQL tables are ***INSERT**, ***NEXT**, ***DELETE**, ***CLOSE** and ***BUFFER**. Similar, or greater, functionality is provided by entry of HP ALLBASE/SQL commands into SQL Logic Blocks, in conjunction with SQL Select Lists.

4. Report Sorting, Selection and File Linkages

   Reports on an HP ALLBASE/SQL database should not use the sorting, selection and file linkage features of the HP ALLBASE/4GL report writer. Similar, or greater, functionality is provided by using the HP ALLBASE/SQL **SELECT** command in a **Start of Report** function.

SQL217: Creating an HP ALLBASE/SQL Database

Student Notes

The **schema** for the database for this training course is shown below:

```
START DBE 'st__base' MULTI NEW LANG=american,
TRANSACTION = 6,
DBEFILE0 DBEFILE sqldBE0
WITH PAGES = 50, NAME = 'st__F0',
LOG DBEFILE sqlDBElog1
WITH PAGES = 288,
NAME = 'sqllog__';

CREATE DBEFILESET RSCEFS;
CREATE DBEFILESET PERSFS;

CREATE DBEFILE persDBEfile WITH PAGES = 100,
NAME = 'persfl__', TYPE = MIXED;

CREATE DBEFILE rsceDBEfile WITH PAGES = 100,
NAME = 'rscefl__', TYPE = MIXED;

ADD DBEFILE persDBEfile to DBEFILESET PERSFS;
ADD DBEFILE rsceDBEfile to DBEFILESET RSCEFS;
commit work;
exit;
```

Replace all occurences of __ with your student number.

Creating an HP ALLBASE/SQL Database

After designing the structure of the HP ALLBASE/SQL database, the developer must create the database, using the utility **isql**.

It is possible to manually type in a sequence of HP ALLBASE/SQL command s each time a database is to be created. A more sensible method is to create a **schema** file using an editor, perhaps **vi** on HP-UX or **HPEDIT** on MPE XL, and then process the **schema** file using **isql**.

The **schema** for the database for this training course is shown below:

```
START DBE 'st00base' MULTI NEW LANG=american,
TRANSACTION = 6,
DBEFILE0 DBEFILE sqldBE0
WITH PAGES = 50, NAME = 'st00F0',
LOG DBEFILE sqlDBElog1
WITH PAGES = 288,
NAME = 'sqllog00';

CREATE DBEFILESET RSCEFS;
CREATE DBEFILESET PERSFS;

CREATE DBEFILE persDBEfile WITH PAGES = 100,
NAME = 'persfl00', TYPE = MIXED;

CREATE DBEFILE rsceDBEfile WITH PAGES = 100,
NAME = 'rscefl00', TYPE = MIXED;

ADD DBEFILE persDBEfile to DBEFILESET PERSFS;
ADD DBEFILE rsceDBEfile to DBEFILESET RSCEFS;
commit work;
exit;
```

The **schema** file, **schema00**, should be copied to the directory or group where the database is to be located; it is processed by running **isql** and using the **start** command, as follows:

```
isql=> start schema00;
```

Notice that **isql** will automatically exit when processing is finished because of the last line in the **schema**.

On HP-UX

On HP-UX, when calling **isql** HP ALLBASE/4GL will do a **cd** to the directory pointed to by the **HP4SQLPATH** environment variable, ensuring that the database is in the correct location.

So on HP-UX, it is possible to run **isql** from HP ALLBASE/4GL when creating the database to ensure that the database is created in the correct directory. Use the function keys `System Keys`, `More Keys` `ISQL` to run **isql**.

On MPE XL

MPE XL does not allow HP ALLBASE/4GL to change group from a new command shell as it invokes **ISQL**, so it is not possible to use HP ALLBASE/4GL to call **ISQL** when creating a database.

It is best simply for the developer to **chgroup** into the group pointed to by the **HP4SQLPATH** environment variable and run **ISQL** manually. In the standard installation, this will be the **HP4SQL** group.

The Application Definition Screen

Refer the students to the screen image at the end of this module.

As mentioned in a previous module, HP ALLBASE/4GL uses the **SQL Data Base Name** from the **Application Definition** screen to locate the HP ALLBASE/SQL database.

On HP-UX, the name has the value of the environment variable **HP4SQLPATH** prepended, unless the name begins with a /, indicating an absolute pathname.

On MPE XL, the name has the value of the environment variable **HP4SQLPATH** appended, unless the name is a fully qualified pathname of the form **FILE.GROUP.ACCOUNT**.

The **SQL Owner Group** name must be set correctly before the database is created. It must not be changed afterwards.

Creating an HP ALLBASE/SQL Database

Use the database schema shown previously to create your own HP ALLBASE/SQL database.

1. Using an editor, type the commands and save them in a file called **schema??**; replace the two question marks with the two digits of your student number.

2. If using HP-UX, run the HP ALLBASE/4GL administrator application and then **isql** using function keys. If using MPE XL, **chgroup** into the **HP4SQL** group and run **ISQL**.

3. Create the database by giving the **isql** command to process the schema file.

4. To check that the database is good, connect to it and browse through some of the tables using the syntax shown in this module.

5. Access the **Application Definition** screen within the HP ALLBASE/4GL administrator application. Change the name of the database from the previous setting; this time ensure that the database name is **st??base**, where the **??** characters represent the two digits of your student number.

6. Note the names and sizes of the files created.

Notes

Application Development

Transfer from ISAM/KSAM to SQL

In this exercise, the students will develop a process and screen to transfer the data stored in the ISAM/KSAM file **person_i** (developed in a previous lab) to the SQL table **person**. This lab only deals with the **Person Tracking** module; the **Book Tracking** module will be completed later using **Module Builder**.

1. Log in to your application as a developer.

2. Create the HP ALLBASE/SQL tables which you defined in a previous module using the **File/SQL Table Creation** screen. The tables are **person** and **books**.

3. Copy the screen **serial_transfer** to the screen **sql_transfer**.

4. Copy the process **serial_transfer** to the process **sql_transfer**. Modify the new process so that it will read data from the ISAM/KSAM file **person_i** and insert it into the HP ALLBASE/SQL table **person**. Do not use an SQL Logic Block to write to the table, but use the **FILE *INSERT** command to insert the data to the table.

A solution:

```
P-sql_transfer

 1 MODE *WRITE person
 2 MODE *READ person_i
 3 SCREEN sql_transfer
 4 MESSAGE pause
 5 FILE *NEXT person_i ; ENTER 10
 6 MOVE R-person_i R-person
 7 SCROLL F-person_number.person
        2 F-person_name.person
        2 F-address1.person
 8 FILE *INSERT person ; ENTER 12
 9 ENTER 5
10 IF *IOSTATUS = N-end_of_file
   THEN MESSAGE pause; SQL commit_work
   ELSE MESSAGE file_error
11 EXIT
12 MESSAGE file_error
13 EXIT
```

SQL data-entry

In this exercise, the students will create a data-entry module based on SQL. The items created for the ISAM/KSAM data-entry module will be copied and modified to cater for the SQL approach.

1. Copy the process **person_details_i** to the process **person_details**. Modify it to use SQL logic blocks to insert new records and update existing records; the SQL logic blocks should be called **person_insert** and **person_modify**.

   A variable called **mode** will be used to determine the mode of operation. Alphanumeric constants **insert** and **update** will be used to set the value of **V-mode**.

   A solution:

```
P-person_details

1 MODE *WRITE person
2 MOVE "1" P-last_person ;
  NOTE Set the person number to 1 initially
3 MOVE C-insert V-mode ;
  NOTE assume we are adding new records initially
4 SCREEN person_details
5 IF V-mode = C-insert
  THEN MESSAGE new_record ; SQL person_insert
  ELSE MESSAGE modify_record ; SQL person_modify
6 SQL commit_work
7 NOTE Automatically bump the person_number by 1.
       So, if they enter a record for person #5 and
       press [Commit Data], the number gets set
       automatically to 6.
8 MATH P-last_person + 1 = P-last_person ; ZIP
9 ENTER 4
```

2. Copy the screen **person_details_i** to the screen **person_details**. Modify all primary movement fields to refer to the SQL table rather than the ISAM/KSAM file. Change the name of the **after-function** on the first field to **person_read**.

3. Copy the function **person_read_i** to **person_read**. The function should attempt to read the row from the table. If the SQL logic block returns an SQL **end-of-file** error, the function should set **insert mode**, otherwise **update mode** should be set. You may wish to make the function operate as both a **prior-function** and as an **after-function**.

   A solution:

```
F-person_read

 1 IF *ENTERED *ON THEN ENTER 5
 2 MOVE P-last_person S-person_number.person_details
 3 SHOW 1
 4 ENTER 20 ; NOTE do a COMMIT WORK before exiting
 5 MOVE S-person_number.person_details P-last_person
 6 SQL person_cursor
 7 FILE *NEXT person ; ENTER 13
 8 NOTE No file error has occurred so we have found
       a row using the value in *S01
 9 MOVE C-modify V-mode
10 SHOW *REFRESH
11 ENTER 20 ; NOTE do a COMMIT WORK before exiting
12 NOTE The error condition has been exercised from
       the FILE *NEXT
13 IF *IOSTATUS <> N-sql_end_of_file
   THEN MESSAGE file_error ; ENTER 20
14 NOTE The row doesn't exist so set insert mode
15 FILE *BUFFER person
16 MOVE S-person_number.person_details
       F-person_number.person
17 SHOW *REFRESH
18 MOVE C-insert V-mode
19 NOTE Do a COMMIT WORK so that we don't have the
       database locked while the user is deciding
       to enter some information into this screen
20 SQL commit_work
```

4. Create the SQL Logic Block **person_cursor** to declare and open a cursor into the table using the value in the first field on the screen.

   A solution:

```
SQL Logic Block person_cursor

SELECT :person FROM sqlgrp.person
WHERE person_number = :S-person_number.person_details;
```

5. Create the SQL Logic Blocks **person_insert** and **person_modify**. They should use the **INSERT INTO** and **UPDATE** commands respectively; there is no need to use a cursor for the **UPDATE** command.

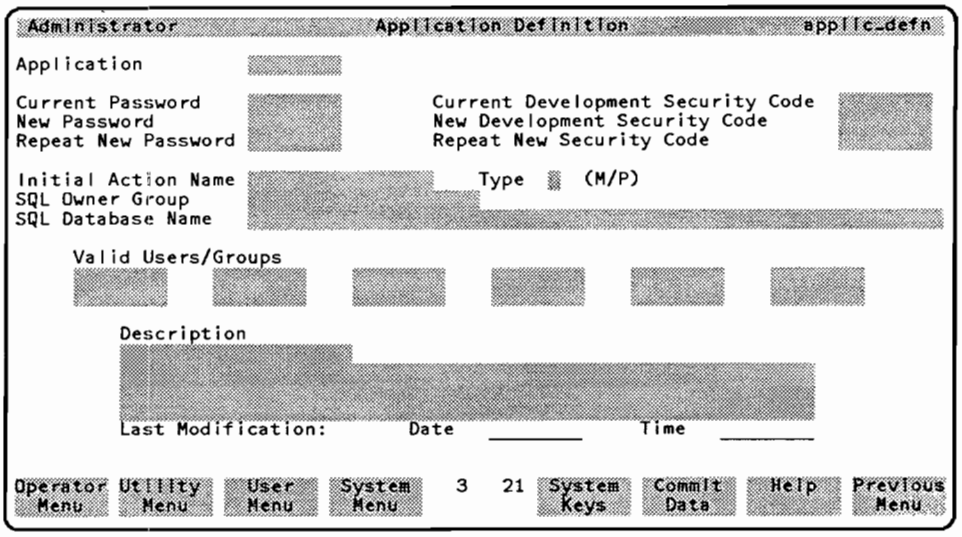
   Possible solutions:

```
SQL Logic Block person_insert

INSERT INTO sqlgrp.person
VALUES (:F-person_number.person,
        :F-person_name.person,
        :F-address1.person,
        :F-address2.person,
        :F-address3.person,
        :F-state.person,
        :F-zip_code.person);


SQL Logic Block person_modify

UPDATE sqlgrp.person SET
       person_number = :F-person_number.person,
       person_name   = :F-person_name.person,
       address1      = :F-address1.person,
       address2      = :F-address2.person,
       address3      = :F-address3.person,
       state         = :F-state.person,
       zip_code      = :F-zip_code.person
WHERE  person_number = :S-person_number.person_details;
```

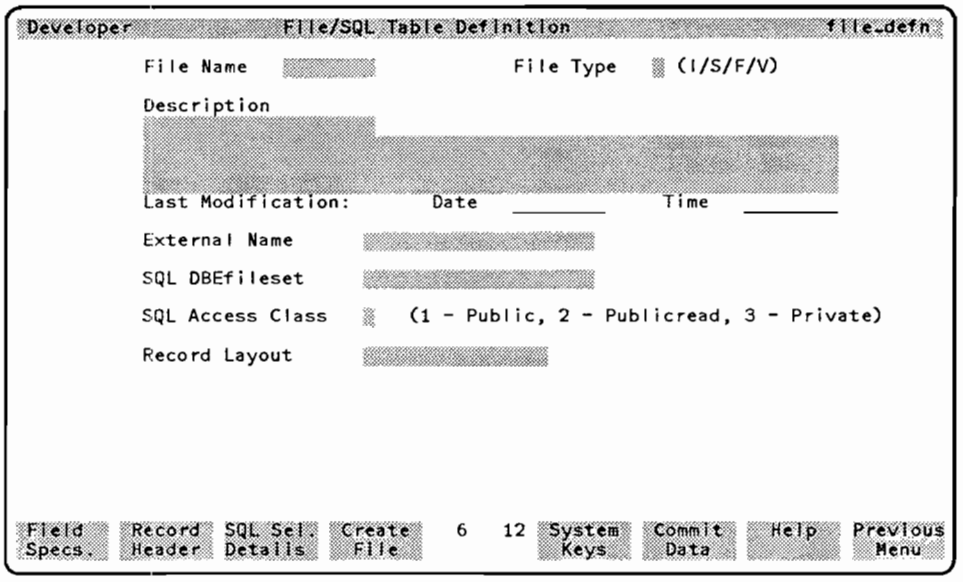
6. Run the module and add some more records to the table; confirm that it operates correctly when adding new records and when updating existing records.

Screen Images

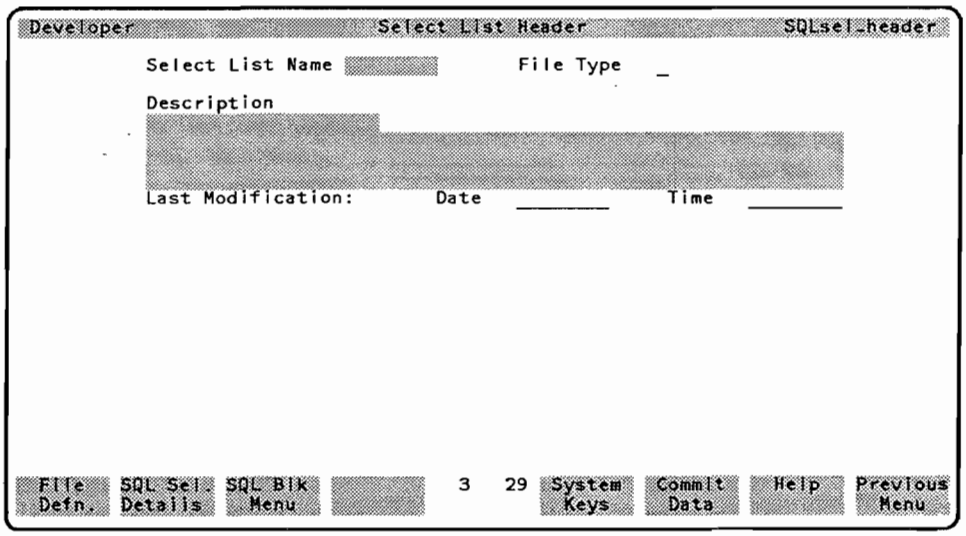
The **Application Definition** screen is shown below:



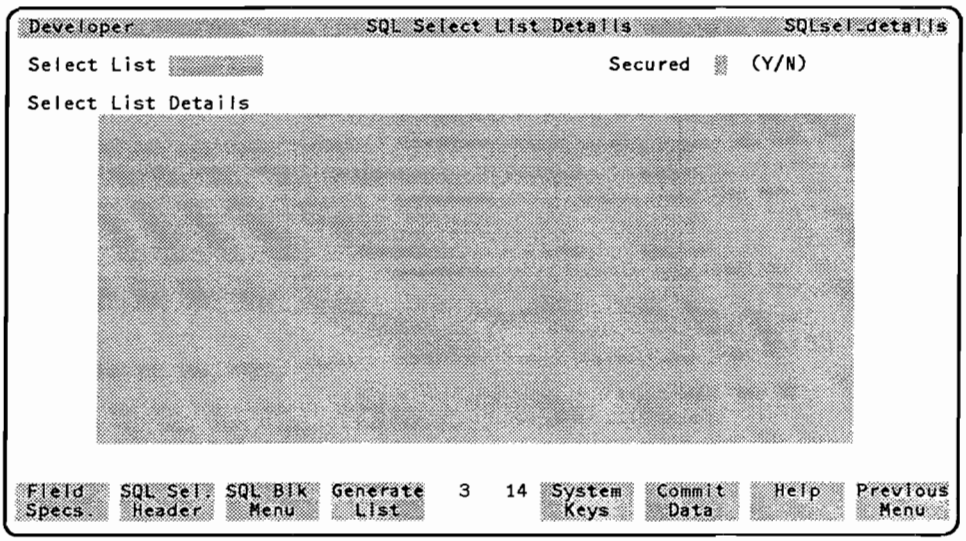
The SQL window on the **File/SQL Table Definition** screen is shown below:



The **Select List Header** screen is shown below:



The **Select List Detail** screen is shown below:



Module: HP ALLBASE/4GL Module Builder

Instructor Notes

MB00 - SLIDE: Module Objectives

MODULE BUILDER

Module Builder Basic Principles
Module Builder – How To!
Module Builder Templates
Developing with Module Builder

OBJECTIVES:

Upon completion of this module, the student will be able to:

1. Describe the basic principles of Module Builder's operation.
2. Describe how to use the Module Builder to quickly develop application modules.
3. Explain what Module Builder templates are and how they are used to generate customized modules with different features to the standard Module Builder modules.
4. Use Module Builder to develop a simple application module.

Australian Software Operation
mb00

HEWLETT PACKARD

Student Notes

Module Objectives

This module will introduce the students to the Module Builder, a facility within HP ALLBASE/4GL which can build a working module or program from a file definition.

Student Objectives

At the end of this module, each student will be able to do the following:

- List the steps required to use the Module Builder.
- List the type of modules created by the Module Builder.
- List the items created by the Module Builder.
- Describe the Module Builder templates and how they are used to generate customised modules with different features to the standard Module Builder modules.
- Use the Module Builder to build a section of their application.

Resource Allocation

Equipment

The following items are required for this module:

- Instructor notes
- Student notes
- Slides
- A laboratory worksheet
- Access to an HP 3000 Series 900 machine or an HP 9000 Series 800 or HP 9000 Series 300 machine with HP ALLBASE/4GL installed.
- The complete set of HP ALLBASE/4GL manuals

Time

This module should take approximately 2 hours, with 1 hours for theory and 1 hour for the practical sessions.

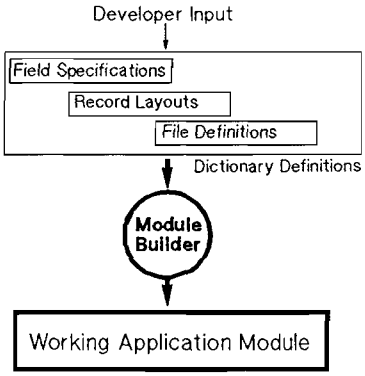
Pacing

Read through the slides reasonably quickly.

MB01 - SLIDE: Basic Principles I

MODULE BUILDER

Module Builder Basic Principles



Module Builder provides the developer with a tool for quickly producing working, fully-functional applicational modules.

With Module Builder, the developer just completes field specifications, record layouts, and file definitions, and then requests Module Builder to generate a screen, process and all function blocks for the module.

Australian Software Operation
mb01

HEWLETT PACKARD

Student Notes

Basic Principles

The HP ALLBASE/4GL Module Builder enables the developer to quickly produce a running application. From the definition of a file Module Builder will construct a **module** consisting of a data entry screen, a main process, a number of associated functions and various other dictionary items required to support the module.

The structure of the module is determined by a set of **templates**; the standard templates can be customised by the developer to produce modules more appropriate for the end-user 's needs.

Development Methodology

As with application design, development using Module Builder begins with the HP ALLBASE/4GL dictionary.

The first requirement is for the developer to define a set of field specifications to match the data requirements. The field specifications should then be grouped into a record layout; in the final stage of preparation, the developer must define the file upon which the module is to be based.

The file can be an ISAM/KSAM file, a serial file (fixed length or variable length), an HP ALLBASE/SQL table or an HP TurboIMAGE data set.

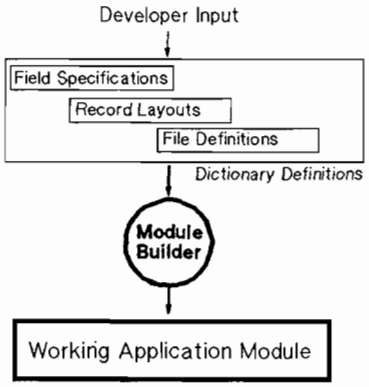
The developer then invokes the Module Builder screen to actually **build the module.**

The last task for the developer is to link the module in to a menu or function key set by specifying the name of the controlling process. The name of the process would normally be tied to a menu item although it could be the action for a function key.

MB02 - SLIDE: Basic Principles II

MODULE BUILDER

Module Builder Basic Principles



Module Builder works from a template application module supplied with the Developer package, or created at the customer site.

Template dictionary references are replaced by those for the defined file(s) and the result is standard HP ALLBASE/4GL screens, logic, etc.

The new module is automatically generated, but all source is available for modification.

Student Notes

Basic Principles II

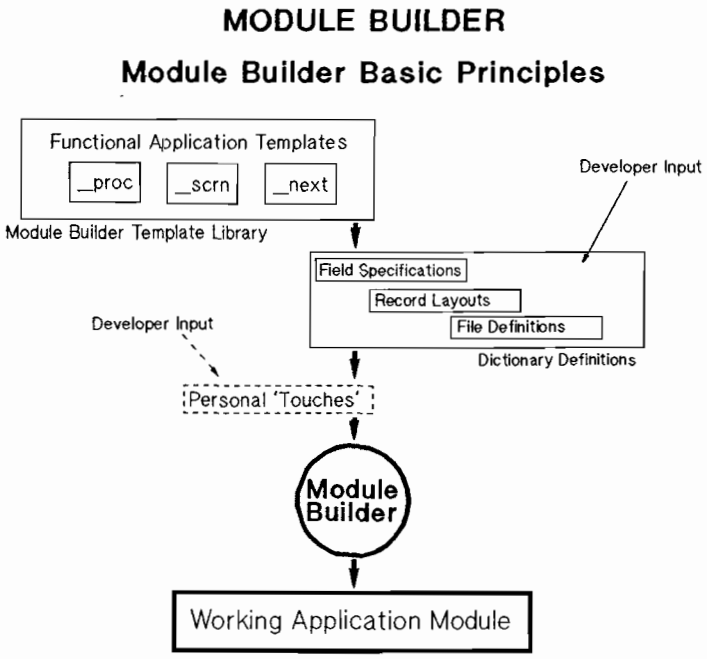
The module created by Module Builder is based on a template supplied with the developer package. Standard HP ALLBASE/4GL logic macros (based on the **DEFINE** command) are expanded to refer to the specific fields defined for the file and then Module Builder creates a standard HP ALLBASE/4GL screen together with standard HP ALLBASE/4GL logic in the process and associated functions.

Module Builder automatically generates the screen and logic blocks. The developer is then free to call up each of the items and make any modifications; the source code is available for acceses.

Naming Convention

All HP ALLBASE/4GL items created by Module Builder carry the prefix **mb_**, indicating that they originated with Module Builder. The name of the module is then added to the end. For example, if the module is called **part**, the main process and screen will be called **mb_part**. The associated functions will have further strings appended according to their function. For example, the function responsible for finding the next record in the file will be called **mb_part_next**.

MB03 - SLIDE: Basic Principles III



Student Notes

Basic Principles II

After defining the module but before actually starting the **build phase**, the developer can add some personal touches.

For example, the developer can decide to omit one or more of the fields present in the record layout from the resultant screen. Fields can also be defined as **display only**, linkages to other files can be established and it is also possible to change the text of the screen labels from the default.

Of course, even greater changes can be made to the module source after the **build phase.**

MB04 - SLIDE: How To!

MODULE BUILDER

Module Builder - How To!

The steps to building an application module with Module Builder are very straight forward:

1. **Create the Dictionary Items for the module.**
   - field specifications, record layouts, file definition and creation.
2. **Fill in the Module Builder Screen.**
   - module name, module type (maintenance or inquiry).
   - name of the main file for the module (see 1.)
   - record layout and index for the main file.
   - include all file fields in screen?
3. **Complete Module Details Screen.**
   - which fields are to be included.
   - which fields are required fields.
   - which fields are display only fields.
   - which fields should provide links to secondary files.
4. **Generate the Module.**
5. **Join the module to a Menu Screen.**

Student Notes

How To!

After defining the minimal set of dictionary items, the developer is ready to let Module Builder loose.

Interacting with Module Builder involves working with two screens, namely the **Module Builder** screen and the **Module Details** screen.

The Module Builder Screen

Refer to the **Screen Images** section to see the **Module Builder** screen.

Module Name

Module Builder uses this to name the items which will be created. If the name already exists, Module Builder prompts the developer to check whether the existing module is to be overwritten.

Be careful in this situation. Any manual enhancements to the existing module will be lost if the developer proceeds.

Module Type

Module Builder can produce two types of modules, namely **Inquiry Modules**, type I, and **Maintenance Modules**, type M.

Maintenance modules allow the end-user to retrieve, add, delete or modify data, whereas **inquiry modules** only permit data retrieval.

Main Access: File

This is the name of the primary file for the module. If a file was recently created, Module Builder supplies the name as the default.

The entry can be the name of an ISAM/KSAM file, a serial file, an HP ALLBASE/SQL table or an HP TurboIMAGE data set.

Record

By default, Module Builder supplies the name of the primary record for the file named in the previous field. This field is bypassed for HP ALLBASE/SQL tables as only one record layout is supported.

Index

By default, Module Builder supplies the name of the primary key field for the record named in the previous field.

For HP ALLBASE/SQL tables, the name of any field in the record layout or the number of any key field can be entered.

For ISAM/KSAM files, only key fields can be specified by name of number.

For serial files, this field is bypassed.

For HP TurboIMAGE data sets, only key fields can be specified by name of number.

Include All Fields

Module Builder allows the developer to exclude some fields from the resultant screen. If the developer enters **Y**, the screen can be committed to move on to the **build** phase. After committing the screen, the function keys change and the developer is required to press `Generate Module` to build the module or `Specify Details`; the latter option allows the developer to specify field details for the resultant screen on a field-by-field basis.

If the developer enters **N**, only the first field is automatically placed on the screen and the **Module Details** screen must be used to specify the other fields to be included on the module's screen.

The Module Details Screen

Refer to the **Screen Images** section to see the **Module Details** screen.

The first group of fields are display only, showing the information already entered on the **Module Builder** screen.

Sequence Number

This number indicates the position of the field in the **tabbing sequence** on the resultant screen; it is not necessary for the fields to appear on the resultant screen in the same order as they are listed in the record layout.

Action

The allowable actions **A**, **C**, **D** and **I** would all be familiar to anybody who has used HP ALLBASE/4GL. The **L** action lists the screen field details.

File

This field first defaults to the name of the main file for the module; if all the fields from the main file record layout have been positioned on the screen, it defaults to the most recently specified secondary file.

Field Spec. Name

Module Builder uses this name to complete the definition of the primary data movement field for the screen field.

Two function keys are displayed when this field is active, namely `Previous Field` and `Next Field`; these keys allow the developer to scan through all the fields defined in the record layout.

A number indicating **occurrence** can be entered into the ( ) field for fields in the record layout which have multiple occurrences defined; the field is bypassed otherwise.

On Screen Label

The contents of this field will be placed on the screen next to the data field. This field defaults to the text in the **Short Description** field from the **Field Specifications** screen, so it is always a good idea to enter a properly capitalised short description when defining field specifications; in fact, it is good practice with all HP ALLBASE/4GL items.

Type

Enter **I** to make the field an **input field** or **D** to make it a **display-only** field.

This field is bypassed if it is dealing with the first field on a screen (always input), a later field for an inquiry module (always display-only) or a field retrieving data from a secondary file (again display-only).

The remaining fields on the **Module Details** screen are bypassed if the **Type** field contains **D**, indicating that the current field under consideration is **display-only**.

Required

Entering **Y** means that the end-user must enter data into the field; it cannot be left blank.

Link

Enter **Y** if the screen field is to be linked to a secondary file. Fields linked to secondary files are called **link fields** and must be display-only.

Validate

Setting this field to **Y** means that the **link field** is validated by data in the index field of the secondary file in the same way a validation range or table operates.

Link to: File, Record, Index

The next three fields allow the developer to specify the details for the secondary file.

A serial file cannot be used as the secondary file.

A file cannot be used more than once in a module; thus, the main faile cannot be used as a secondary file and a secondary file can only be linked to one screen field.

Module Generation

Module generation is actually a two stage process.

First Module Builder resolves references to all HP ALLBASE/4GL data items and builds the source for each of the items in the module.

Upon successful completion, Module Builder will then automatically invoke **generates** to convert the module into an executable form.

Occasionally Module Builder will find problems with the module. Most problems are due to the developer using a record layout with too many fields; consequently Module Builder complains that the screen is too cramped!

Connecting to the Application

The module must be connected to the rest of the application in some way; this is normally done by making the main process **mb_name** the action for a menu item.

MB05 - SLIDE: Module Builder Capabilities

MODULE BUILDER

Module Builder Capabilities

Capabilities	Module Type	HP ALLBASE/SQL		KSAM/ISAM		Serial		HP TurboIMAGE	
		Unique Key	Duplicate Key	Unique Key	Duplicate Key	Fixed Length	Variable Length	Unique Key	Duplicate Key
File Scanning	I and M	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Full-Key Read	I and M	Yes	Yes	Yes	Yes	No	No	Yes	Yes
Partial-Key Read	I and M	Yes	Yes	Yes	Yes	No	No	Yes	Yes
Add Records	M only	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Insert Records	M only	No	Yes	No	Yes	No	No	No	Yes
Modify Records	M only	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Delete Records	M only	Yes	Yes	Yes	Yes	No	No	Yes	Yes

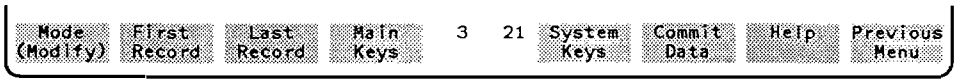
Module Builder can automatically generate two kinds of module:

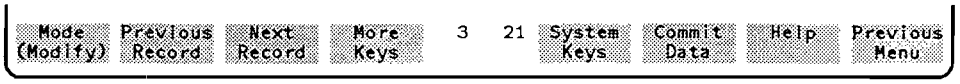
Inquiry and Maintenance.

The standard Module Builder modules provide a high level of functionality in the resulting application modules, however this is highly dependant on the file type being accessed.

Student Notes

The function keys for the resultant screen are shown below:





Module Builder Capabilities

Modules created by Module Builder provide sophisticated functionality for the end-user; the HP ALLBASE/4GL logic produced for the process and functions is comparable with that produced by an experienced developer, certainly beyond the capabilities of a novice. Yet novices can produce very powerful applications using this feature.

Screen Painter Touch-up

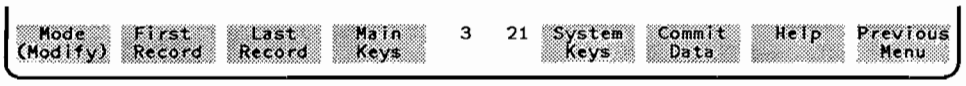
In most cases, the developer will want to **touch-up** the resultant screen after having a first look. Typically screen fields and associated text can be re-positioned (maintaining tabbing sequence) and special text, such as line drawing, can be added to improve the screen's appearance.

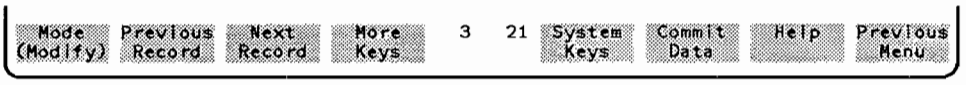
If careful choices were made when entering **short descriptions** or **on-screen labels**, no changes need to be made to the data field labels.

The developer will quickly learn how to do things to save time and effort.

File Access

File accessing is one of the features of the resultant module. The resultant screen provides the end-user with the following sets of function keys.





These keys allow the end-user to scan forwards and backwards through the data file. The end-user can also jump to the first record or the last record in the file. When trying to match an entry, the screen can find a match based on either a **full-key read** or a **partial-key read**. A **partial-key read** means that an entry of Smi will find the first record in the file which starts with those three letters, providing a very powerful way of interrogating the database.

Maintenance Modules

With Maintenance Modules, if the entry in the first field of the screen does not match any records in the file, a new record will be created; however, if a match is found, the existing record is retrieved, ready to be updated.

The screen operates in a number of modes, namely **Add**, **Insert**, **Modify** and **Delete**; the mode will change automatically depending on the operation currently being performed and the end-user will always be informed of the current mode of operation.

Data Manager Limitations

Data access capabilities are highly dependant upon the underlying data manager as indicated by the type of file. Obviously modules based on ISAM/KSAM, HP ALLBASE/SQL and HP TurboIMAGE support greater functionality than those based on serial files.

For example, serial files do not support **keyed reads** because serial files do not have keys! Also, the **file scanning** capabilities of serial files do not match those of other data managers; this will be discussed in the next slide.

The table shown on the overhead projector summarises the capabilities provided.

Intentionally blank

MB06 - SLIDE: Module Builder Capabilities - File Scanning

MODULE BUILDER

Module Builder Capabilities

Scan Capability	HP ALLBASE/SQL		KSAM/ISAM		Serial		HP TurboIMAGE	
	Unique Key	Duplicate Key	Unique Key	Duplicate Key	Fixed Length	Variable Length	Unique Key	Duplicate Key
First Record	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Next Record	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Previous Record	Yes	No	Yes	Yes	Yes	No	Yes	Yes
Previous Key Value	No	Yes	No	No	No	No	No	No
Last Record	Yes	No	Yes	Yes	Yes	No	Yes	Yes
Last Key Value	No	Yes	No	No	No	No	No	No

Scanning facilities are provided in both Inquiry and Maintenance module types. Inquiry modules give full access to scanning facilities from the main function key set displayed; for Maintenance modules, this set is accessed via [More Keys].

Scanning facilities reflect the differences in file types.

Student Notes

Module Builder Capabilities - File Scanning

Serial files do not support **key value scanning** because they do not have keys.

Fixed length serial files support all of the other scanning capabilites but variable length serial files only support **First Record** and **Next Record** scanning.

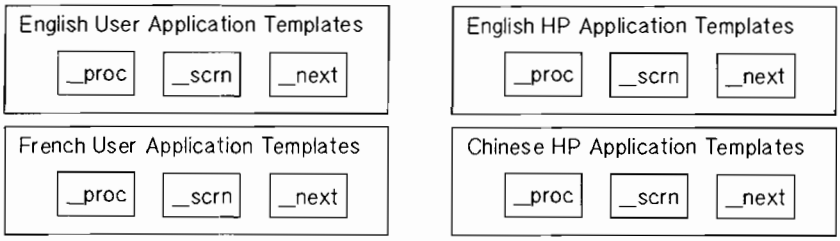
The basic problem with variable length serial files is that it is very difficult to scan backwards through the file as the record length is unknown.

HP ALLBASE/SQL tables do not support **previous key value** scanning for duplicate keys as HP ALLBASE/SQL does not support backwards searching.

MB07 - SLIDE: Module Builder Templates

MODULE BUILDER

Module Builder Templates



Module Builder Template Library

If the supplied Module Builder templates are not exactly what you need, it is possible to develop custom versions for use in your environment.

* All the facilities and ease of use of Module Builder.
* Your custom application templates - designed for your needs.

Student Notes

Module Builder Templates

The standard Module Builder environment provides developers with a very powerful and flexible development system, but it still may not suit everybody's needs.

For example, a French language developer would quickly become frustrated if they found that they had to translate the text of the function keys for every module produced by Module Builder. Why can't I have my own set of function keys in the French language? Sacre Bluer!!

Another developer, after thinking long and hard, decides that she can do a better job with the HP ALLBASE/4GL logic to be produced by Module Builder than is provided in the standard templates.

For this reason, HP ALLBASE/4GL allows the developer to customise templates to more closely match the developer's individual requirements. This allows developers to have a completely localised set of templates, able to produce modules suited to the native language of the end-users.

The ability to tailor to individual requirements is a very powerful feature.

MB08 - SLIDE: Module Builder Templates - Contents

MODULE BUILDER

Module Builder Templates

Custom template modules may contain the following HP ALLBASE/4GL items:

* A Process.
* A Screen.
* Function Key Sets.
* Functions.
* SQL Logic Blocks.
* Constants.
* Variables.
* Messages.
* Help Screens.

Student Notes

Module Builder Templates - Contents

The Module Builder constructs a number of HP ALLBASE/4GL items to create a module. The Module Builder creates items by copying templates, which are stored in library applications, into the current application and making the necessary modifications.

Module Builder templates contain the following:

- A Process
- A Function
- Function Key Sets
- SQL Logic Blocks
- Constants
- Variables
- Messages
- Help Screens

and these are also the items that can be created during the **build phase.**

The developer may modify the templates but no extra item types can be included.

The following HP ALLBASE/4GL item types are not supported:

- Application titles
- Calculated items
- Decision tables
- Master titles
- More than one resultant screen
- Reports
- Scratch pad field names
- SQL select lists
- Validation ranges
- Validation tables
- Work areas

MB09 - SLIDE: Module Builder Templates - Macros

MODULE BUILDER

Module Builder Templates

Template modules take advantage of the following macro substitutions:

@FILE@ -> Primary or link file name.

@RECORD@ -> Primary or link file record name.

@INDEX@ -> Index field for main or link file.

@LINK@ -> Name of SQL logic block for link function.

@SCREEN@ -> Name of module screen.

also for SQL logic blocks:

@FIELD@ -> Current Screen field name.

@TABLE@ -> Name of current SQL table.

@COLUMNS@ -> List of columns to UPDATE.

@VALUES@ -> List of screen fields, corresponding to table columns.

@UPDATECLAUSE@ -> List of fields to update.

Student Notes

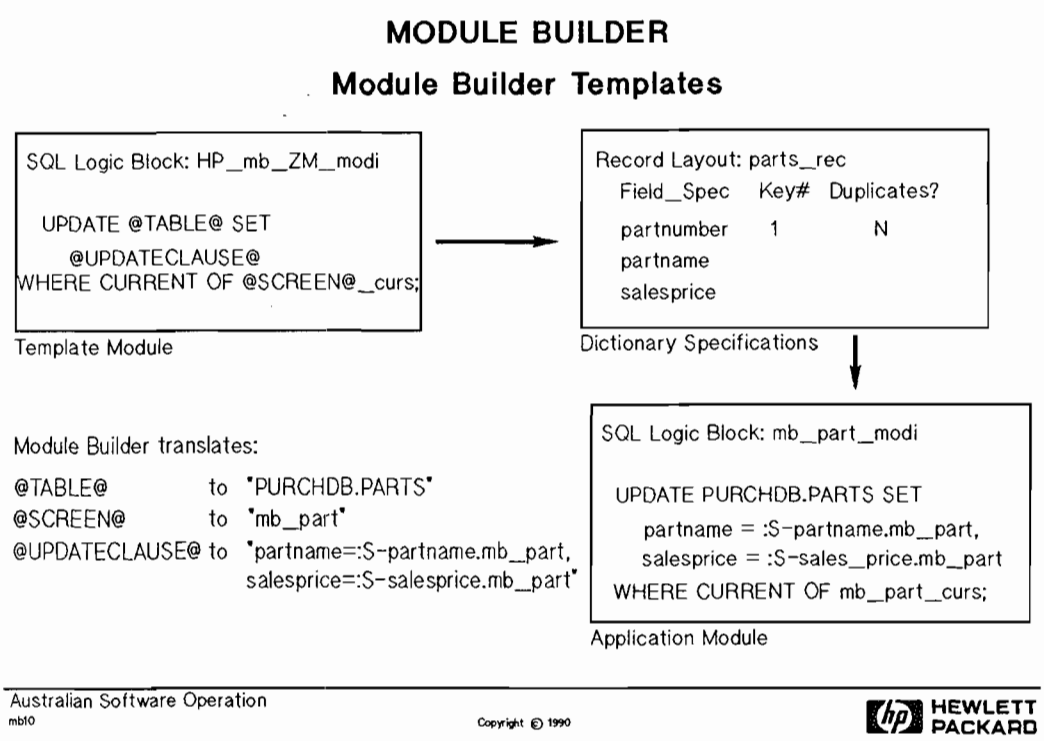
Module Builder Templates - Macros

Within application items there are cross references to fields, record layouts, files, SQL tables, and other application items. The names of all these are unique for each module. To ensure that all references are resolved, templates contain macros and template names which are substituted with unique names during the **build phase**. Macros refer to files and their components, and SQL table items, while templates refer to other application items.

The macros, and their substitutions, are shown on the slide.

The value substituted for **@FILE@** is context dependent. If the context deals with the main file, the main file name or index name is used. If the context deals with the link file, the link file name or index name is used.

MB10 - SLIDE: Module Builder Templates - Example



Australian Software Operation
mb10

HEWLETT PACKARD

Student Notes

Module Builder Templates - Example

For each HP ALLBASE/4GL item type, a library application contains a number of templates.

For example, there is a separate template corresponding to each type of operation to be performed by SQL logic blocks. Thus there are templates for **selecting a record with the current key for inquiry or update, inserting a new record in the table, updating the currently displayed record**, to name a few.

The slide shows a sample SQL Logic Block template module and the end result after macro substitutions have occurred.

The original template for the SQL logic block is called **HP_mb_ZM_modi**. All SQL logic block templates have the prefix **HP_mb_ZM**; the **_modi** suffix indicates that the template is for an SQL logic block which can **update the currently displayed record.**

In the example on the slide, the record layout is called **parts_rec** and the HP ALLBASE/SQL table is called **part**. Thus, the name of the resultant SQL logic block is **mb_part_modi**.

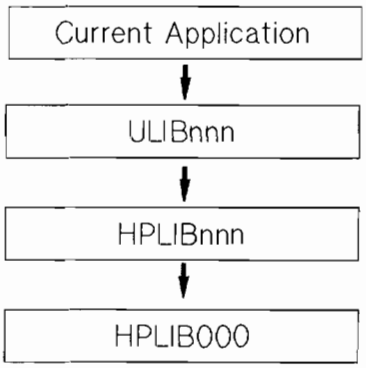
The macro **@TABLE@** translates to **PURCHDB.PARTS**; the other macros translations are shown on the slide.

MB11 - SLIDE: Module Builder Templates - Hierarchy

MODULE BUILDER

Module Builder Templates

Module Builder searches an application list looking for a required module template:



nnn – current NLS language.

Student Notes

Module Builder Templates - Hierarchy

The Module Builder searches for a required template through a hierarchy of four applications.

First in the hierarchy is the current application, followed by the three library applicatons. When the search for a template fails at one level, the Module Builder searches the next level in turn.

Note

The library applications are only used to store templates; they cannot be run as stand-along applications.

Current Application

Templates in the current application can specifically cater for the requirements of that application; these are **local templates.**

Site Customisable Template Library

This user library is called **ULIBnnn**, where **nnn** is the three digit code referring to the current language identifier. For example, if the current language is **american**, the library is **ULIB001.**

This library should be used to store **developer-created** templates for use in several applications.

Native Language Template Library

A localisation center may have prepared local language versions of the Module Builder templates. If so, the templates will be stored in an application called **HPLIBnnn**, where **nnn** is the three digit language code.

Standard Template Library

By default, all required templates exist in an application called **HPLIB000**; this is supplied as part of the HP ALLBASE/4GL product.

The contents of the **HPLIB000** application can be printed using the developer printing utilities.

Note

Do not modify the templates in **HPLIB000**. If you wish to make modifications, copy them to the **ULIBnnn** application and modify them there.

Application Development

In this laboratory exercise, the students will use Module Builder to build two modules for **book tracking**. They will build similar modules to the module for **people tracking** which they have already developed in the course - this will contrast application development with the standard development method, and application development with Module Builder.

The first module will provide a **maintenance module** for the **books** base table, and the other an **inquiry module** for the **books** base table.

Creating the Maintenance Module

From the main menu of the **developr** select and activate the **Module Builder** menu item. You are requested to enter the name of the module you will build. For the first module, enter the module name **books**.

You are then requested to specify whether the module will be a (M)aintenance or (I)nquiry screen. Enter **M** to indicate a **maintenance module**.

The module will access the file **books**; use the default primary index. You are then asked if you wish to include all fields. Read the help for this field; enter Y and press the `Commit Data` function key.

You are prompted to press `Specify Details` or `Generate Module`. `Specify Details` provides the opportunity to tailor the application module before generation. For now, press `Generate Module`. Your module should generate correctly. Note that the module name **books** is prefixed with **mb_** to provide the module's main process name.

You must now set up access to the process just created. Press `Screens` and then go into the screen painter to modify the **books** menu. Add the new action item to call the process **mb_books**. Choose your own label or use the default.

Now try out the new maintenance module. Add about 10 records to the table.

Questions

1. Does Module Builder create any source for the generated module? Use the **Catalog Display** screen to list Processes, Functions, SQL Logic Blocks, and Screens.

   Answer:

   Yes, all items created by Module Builder contain source code.

2. What does Module Builder build?

   Answer:

   A process, a screen, some function key sets, some functions, some help screens, some messages, some constants, etc.

Creating the Inquiry Module

Now build an inquiry module to access the **books** file. Go into Module Builder and enter the name of the inquiry module. Enter the module name **inquiry**.

You are then requested to enter if the module will be a (M)aintenance or (I)nquiry screen. Enter **I** to indicate an **inquiry module**.

The module will access the file **books**; use the default primary index for the file. You are then asked if you wish to include all fields. Enter Y and press the `Commit Data` function key.

You are asked if you want to press `Specify Details` or `Generate Module`. This time use `Specify Details` to tailor the **On-Screen Labels** that will be placed in your screen; make some changes to the labels and press `Commit Data`.

With **tailoring** done, press `Generate Module`. The inquiry module should generate correctly.

You must now set up access to the process just created. Press `Screens` and go into the screen painter to modify the **books** menu. Add the new action item to call the process **mb_inquiry**. Choose your own label or use the default.

Try out your new inquiry module.

1. What is the effect of using the `Specify Details` to tailor on-screen labels?

   Answer:

   The text associated with the data fields now differs from the default.

2. What is the difference between a **M** type module and an **I** type module?

   Answer:

   A Maintenance Module allows the user to view and update the data whereas an Inquiry Module only allows the user to view the data.

Templates

Return to the sign-on screen, and log in to the library application **HPLIB000.**

Review the SQL Logic Block **HP_mb_SM_modi**. Module Builder used this SQL logic block template as the basis for creating the **mb_books_modi** SQL logic block in the **books** module.

Compare **HP_mb_SM_modi** in **HPLIB000** with **mb_books_modi** in your application.

1. What is the **@TABLE@** template macro in **HP_mb_SM_modi** mapped onto in **mb_books_modi**?

   Answer:

   It is mapped onto **SQLGRP.RESOURCE.**

2. What about the **@SCREEN@** macro?

   Answer:

   It is mapped onto **mb_books.**

3. What about the **@UPDATECLAUSE@** macro? List the individual fields.

Allocation of Resources

You should now plan a module to be used to track the allocation of resource items (books) to the various people. The module should be based on the SQL table **loans** which contains the columns **person_number**, **book_number**, **loan_date** and **due_date**.

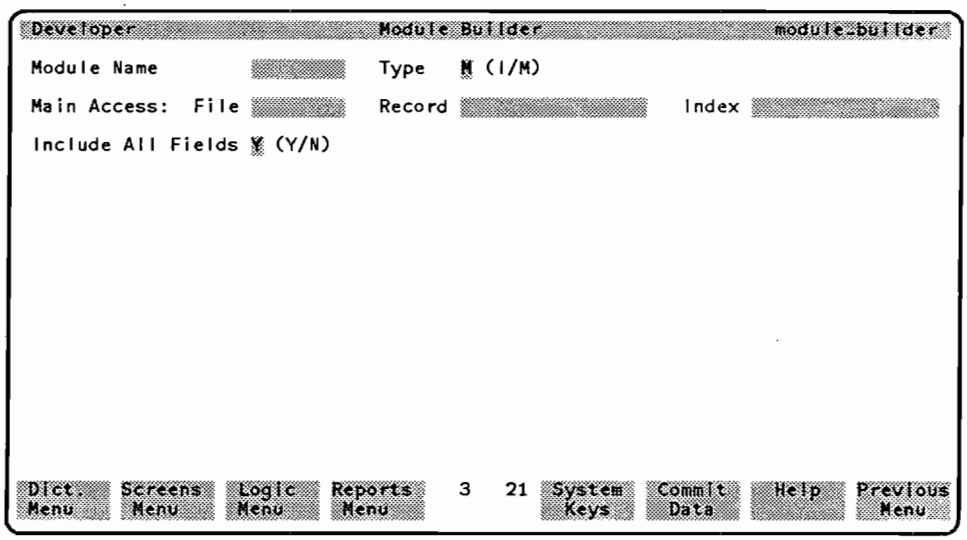
The module should be able to operate in three modes, namely **book loans**, **book returns** and **loan updates** (extensions of borrowing time).

Think about how you would modify a **maintenance module** for the **loans** table, created by Module Builder, to retrieve information from both the **person** table and the **books** table. Some of the fields on the main screen should be display only; others should become display only when processing updates and returns.

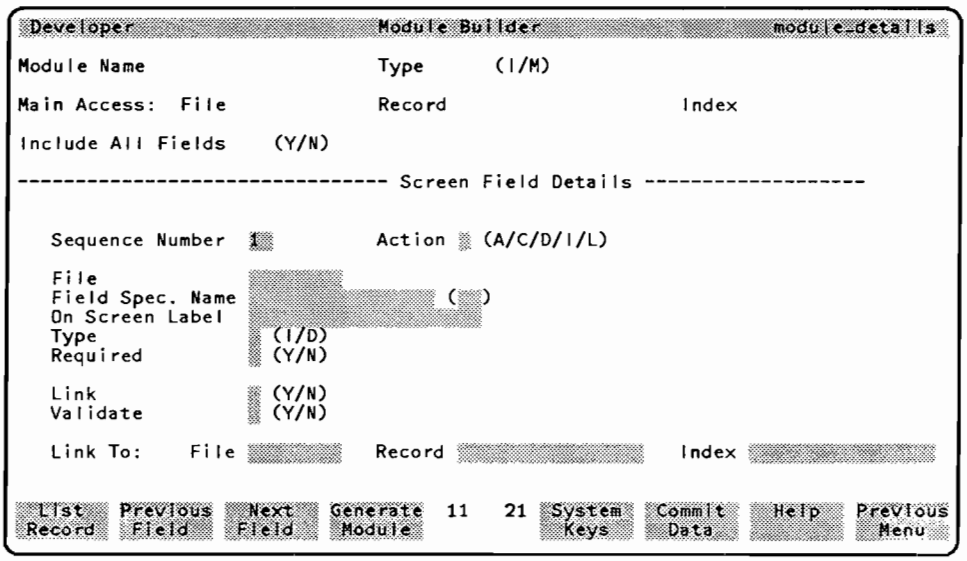
Do not worry if you don't finish the planning or are not able to implement this module by the end of the training course. It will provide you with an on-going task once you return to your normal work environment!

Screen Images

The **Module Builder** screen:



The **Module Details** screen:



Module: The HP TurboIMAGE Interface

Instructor Notes

TIMAGE00 - SLIDE: Objectives

HP TurboIMAGE INTERFACE

Module Objective

At the end of this module, the student will be able to:

- Describe the HP TurboIMAGE interface.
- List the logic commands for accessing HP TurboIMAGE.
- Describe the access modes and locking facilities.
- Describe and use the external utilities provided.
- Use the interface to transfer data from HP TurboIMAGE to HP ALLBASE/SQL.

Australian Software Operation
timage00

HEWLETT PACKARD

Student Notes

Module Objective

This module will introduce the students to the interface to HP TurboIMAGE provided by HP ALLBASE/4GL on MPE XL. It will be optional, as HP TurboIMAGE is not available on HP-UX and may not be of interest to customers or SEs who work solely with HP-UX.

Student Objectives

At the end of this module, each student will be able to do the following:

- List and describe each part of the HP ALLBASE/4GL interface to HP TurboIMAGE.
- List and describe the logic commands for working with HP TurboIMAGE.
- List the possible modes for HP TurboIMAGE dataset access.
- List the operations provided by the FILE command.
- Describe the locking facilities available with HP TurboIMAGE datasets.
- List and describe the utilities provided for uploading and downloading definitions between HP ALLBASE/4GL and an HP TurboIMAGE database.
- Use the interface to develop a logic block which would transfer data from an HP TurboIMAGE dataset to an SQL table.

Resource Allocation

Equipment

The following items are required for this module:

- Instructor notes.
- Student notes.
- Slides.
- A laboratory worksheet.
- Access to HP 3000 Series 900 machine with HP TurboIMAGE.
- An HP TurboIMAGE database installed on the machine.

Time

This module should take approximately 3 hours, with 2 hours for theory and 1 hour for the 3 practical sessions.

Pacing

Read through the slides reasonably quickly. Much of the detail should be familiar to the students as the modules on **ISAM/KSAM**, **SQL** and **serial** files have already been completed.

For full detail, refer the students to the Developer Reference Manual.

Allow the full time for the laboratory exercises.

Intentionally blank

TIMAGE01 - SLIDE: Overview - Interface Components

HP TurboIMAGE INTERFACE

Interface components

The HP ALLBASE/4GL interface to HP TurboIMAGE includes new and enhanced:

- Administrator screens.
- Developer screens.
- Logic commands.
- External utilities.

Student Notes

Overview - Interface Components

The HP ALLBASE/4GL interface to HP TurboIMAGE involves both the **administrator** and **developer** applications.

In the **administrator** application, the enhancement involves the following:

- The new **Database Definition** screen.
- The new **Database Attributes** screen.
- New items on the main menu.
- A new item on the **Administrator Reports** menu.
- An extra item on the **Administrator Deletions** screen.

In the **developer** application, the enhancement involves the following:

- A new field on the **Field Specification** screen.
- The new **Data Mgr. Specific Attributes** screen.
- A new window for the **File Definition** screen.
- A new logic command.
- Changes to some existing logic commands.

Two new function key sets are included and some sets have been modified.

Two utilities are also supplied for working with HP TurboIMAGE databases and two have been enhanced.

TIMAGE02 - SLIDE: The Administrator Application

HP TurboIMAGE INTERFACE

The Administrator Application

- The HP ALLBASE/4GL administrator contains two new screens:
  - * The Database Definition Screen.
  - * The Database Access Parameters screen.
- Plus two new items in the Main Menu.
- The Administrator Deletions screen has also been changed.

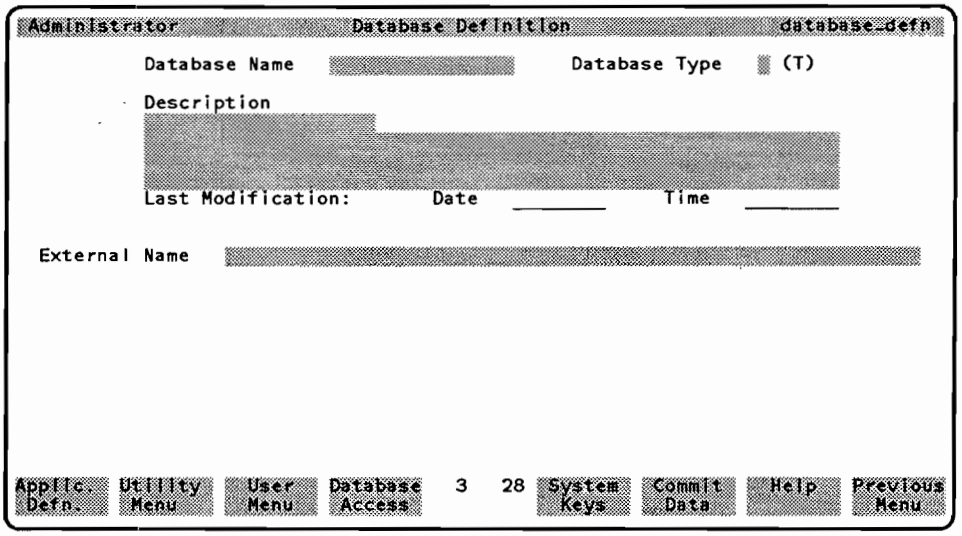
Student Notes

The Administrator Application

The HP ALLBASE/4GL administrator application allows the administrator to define HP TurboIMAGE databases and to assign HP TurboIMAGE databases to HP ALLBASE/4GL applications. More than one database can be accessed from an application. The administrator is also able to specify the access parameters for each database, to print database details and to delete database definitions.

New screens have been added, new items have been added to the main menu and some new options have been added to some existing screens.

TIMAGE03: The Administrator Application - Database Definition



Student Notes

If **HP4TIPATH** is **HP4TI.HP4GL**, write down what happens to the external name:

DEBTS becomes:

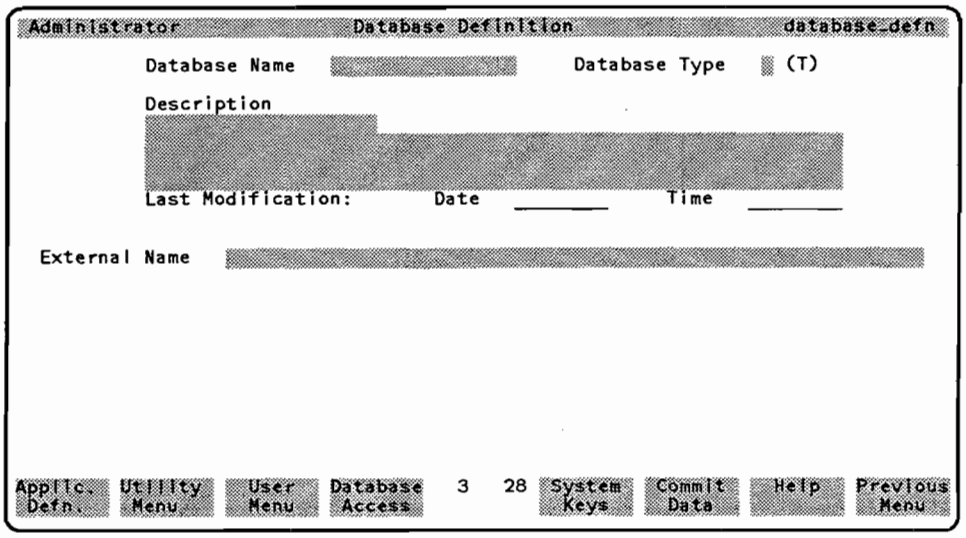
DEBTS.PUB becomes:

DEBTS.PUB.SYS becomes:

The Administrator Application - Database Definition

The **Database Definition** screen is used to define the HP TurboIMAGE databases to be accessed by HP ALLBASE/4GL applications.

The screen is shown below:



The administrator specifies the name of the database and its type; currently only HP TurboIMAGE (T) is supported. A description of the database is then entered, followed by the external name of the database. It is anticipated that a future version of HP ALLBASE/4GL will allow HP ALLBASE/SQL databases to be defined using this screen.

The value of the system variable **HP4TIPATH** is appended to the external name for a HP TurboIMAGE database unless the external name is fully qualified, i.e. of the form **FILE.GROUP.ACCOUNT**.

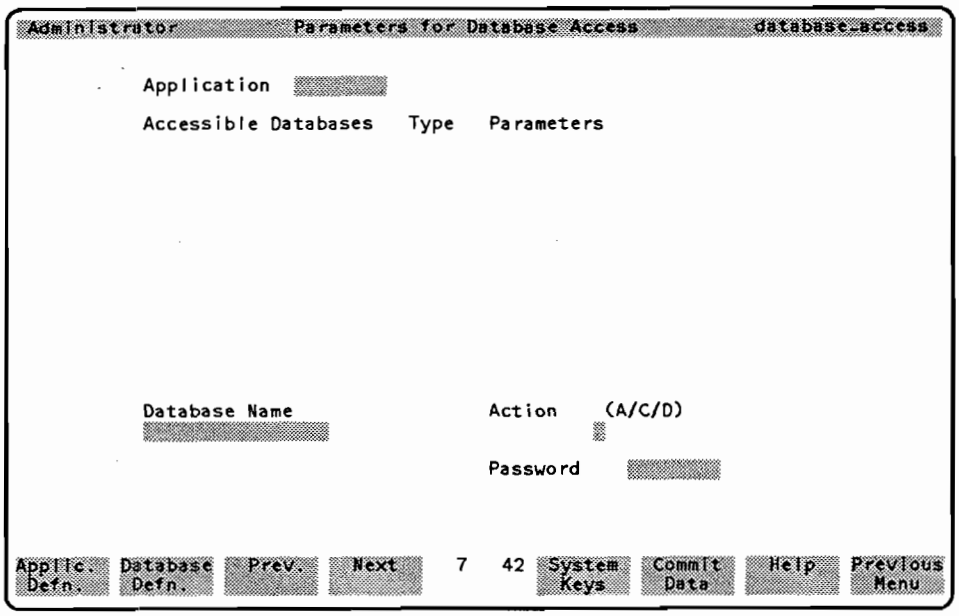
With **HP4TIPATH** set to **HP4TI.HP4GL** the external name:

DEBTS becomes **DEBTS.HP4TI.HP4GL**;

the name **DEBTS.PUB.SYS** is unchanged;

the name **DEBTS.PUB** becomes **DEBTS.HP4TI.HP4GL**.

TIMAGE04: Administrator Application - Database Access Parameters

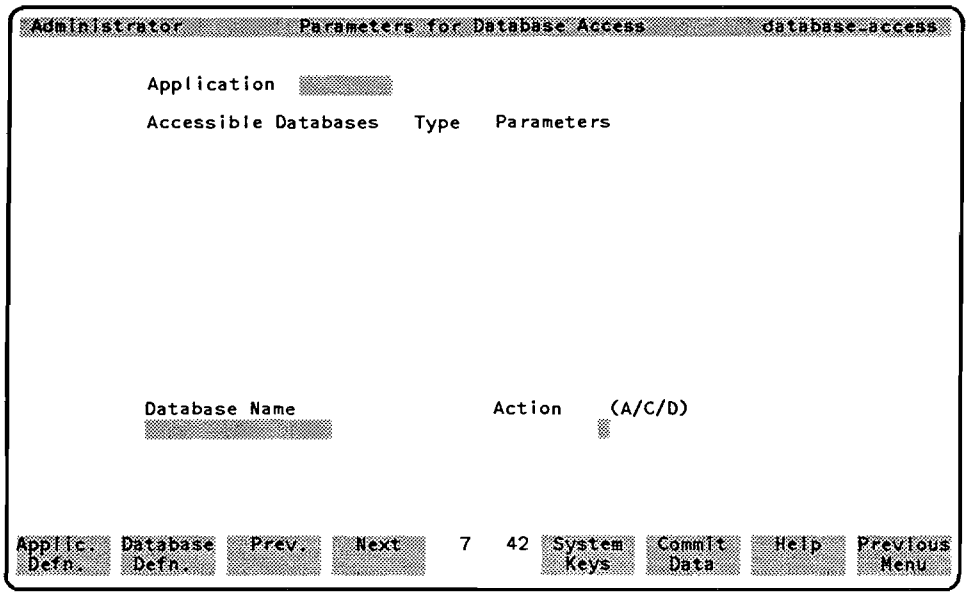


Student Notes

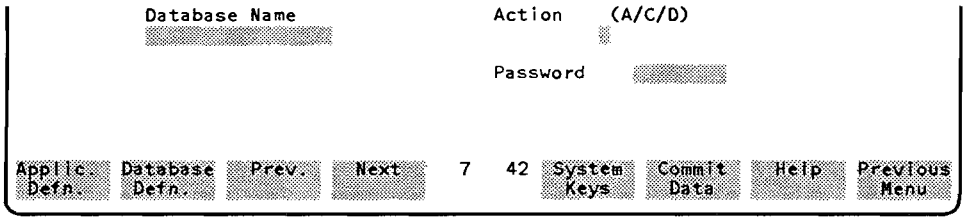
The Administrator Application - Database Access Parameters

The **Parameters for Database Access** screen is used to associate HP TurboIMAGE databases with an application and to specify the access password. Note that multiple HP TurboIMAGE databases can be accessed from an HP ALLBASE/4GL application.

The screen is shown below:

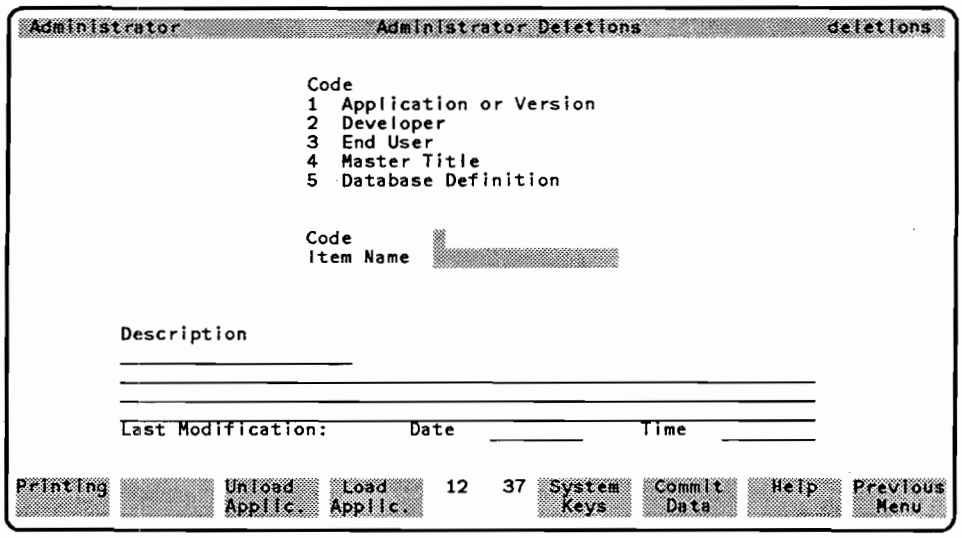


When the name of a HP TurboIMAGE database is entered, a window is displayed prompting the administrator for the application password. The window is shown below:



TIMAGE05: The Administrator Application - Deletions and Reports

Administrator Deletions



Student Notes

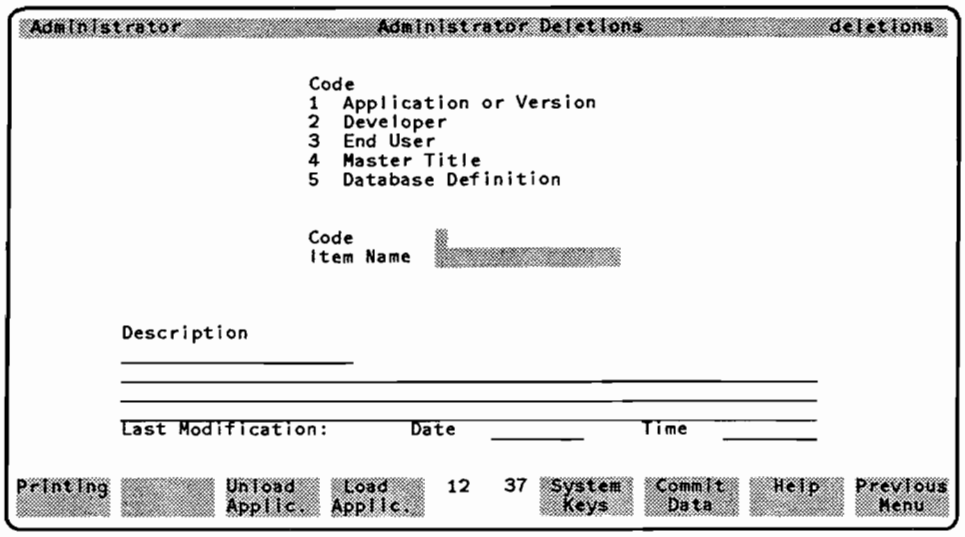
Administrator Documentation

The **Administrator Documentation** menu now allows the administrator to print details of HP TurboIMAGE databases; refer to the **Screen Images** section at the end of the module.

The Administrator Application - Deletions and Reports

Administrator Deletions

The **Administrator Deletions** screen now allows the administrator to delete database definitions. The screen is shown below:



The new item is **5** for deleting **Database Definitions**.

Administrator Documentation

The **Administrator Documentation** menu now allows the administrator to print details of HP TurboIMAGE databases. Refer the students to the **Screen Images** at the end of the module.

TIMAGE06 - SLIDE: The Developer Application - Field Specifications

HP TurboIMAGE INTERFACE

Field Specifications Screen

The HP ALLBASE/4GL Field Specification Screen now supports the following Storage types:

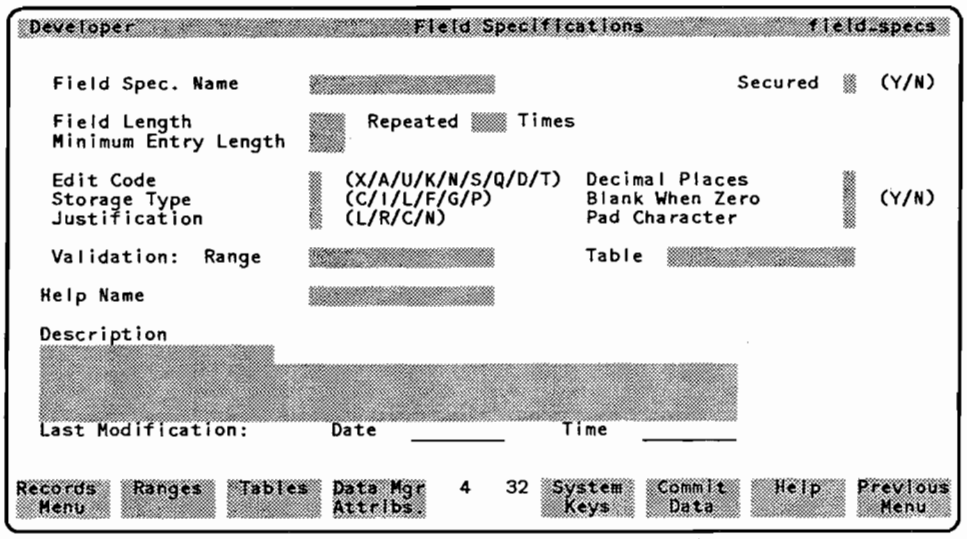
- C – Character
- I – Integer
- L – Long Integer
- F – Single precision floating point
- G – Double precision floating point
- P – Packed decimal

Student Notes

The Developer Application - Field Specifications

The **Field Specifications** screen has been modified as HP ALLBASE/4GL now supports new numeric formats using the **Storage Type** field.

The new screen is shown below:



The new field is for **Storage Type**; the allowable values are:

- C - Character type
- I - Integer (two byte)
- L - Long integer (4 byte)
- F - Floating point (4 byte IEEE, single precision)
- G - Floating point (8 byte IEEE, double precision)
- P - Packed decimal (BCD)

An entry is only required in this field if the edit code is N or S. The default value for this field is **C** for **character type**; this should be used for a **non-numeric** edit code.

The tabbing sequence is now ..., **Edit Code**, **Storage Type**, **Justification**, **Decimal Places**, **Blank When Zero**, **Pad Character**, ...

TIMAGE07 - SLIDE: The Developer Application - Extended fields

HP TurboIMAGE INTERFACE

Data Mgr. Specific Field Attributes screen

The Data Mgr. Specific Field Attributes screen allows the developer to specify the:

- Data Manager for a Data Item
  (TurboIMAGE is the only supported Data Mgr.)
- Data Item name
- Sub-Item count
- Storage Item Type designator

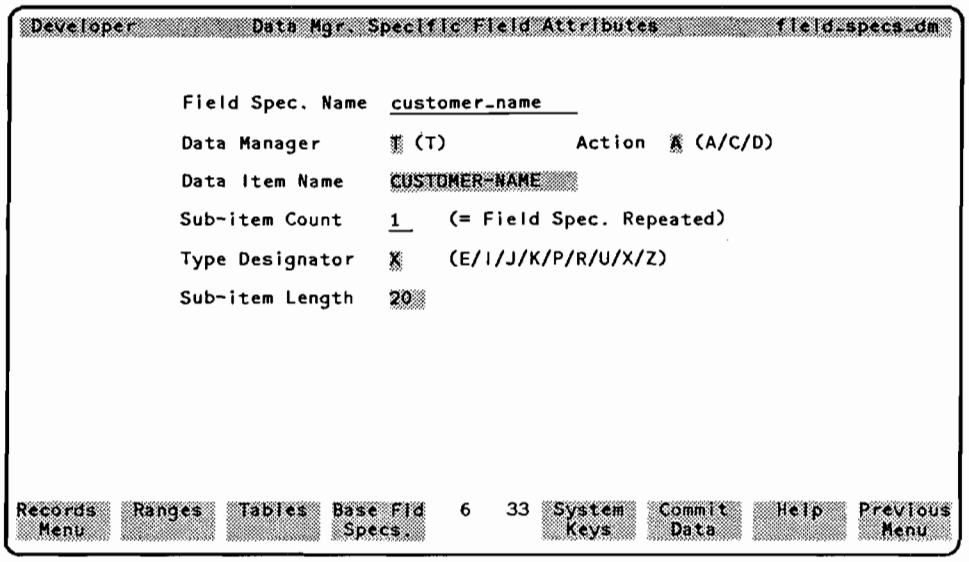
Student Notes

The Developer Application - Extended Field Specification

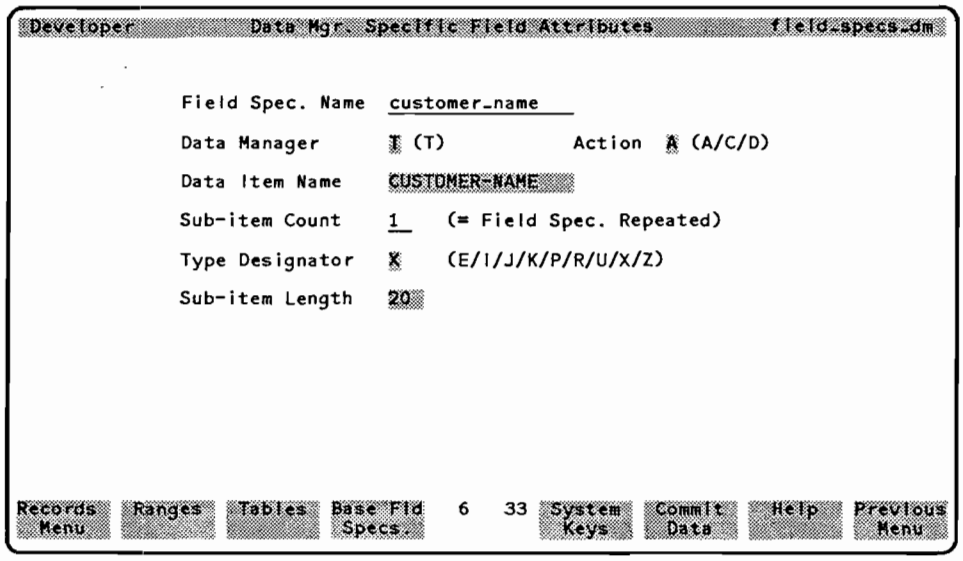
The **Data Mgr. Specific Field Attributes** screen allows the entry and validation of attributes required by specific data managers. It can only be called from the `Data Mgr Attribs.` function key on the **Field Specifications** screen and it should only be called once the **Base Field Specifications** screen has been completed.

The developer specifies the data manager type and the appropriate window is displayed. Currently only HP TurboIMAGE is supported.

The screen with the HP TurboIMAGE window displayed is shown below:



TIMAGE08: The Developer Application - Extended Field Specification



Student Notes

Field Spec. Name:

Data Manager:

Data Item Name:

Sub-item Count:

Type Designator:

Sub-item Length:

The Developer Application - Extended Field Specification

The relevant fields on the screen are:

- Field Spec. Name

  A read-only field containing the name on the original screen.

- Data Manager

  Only HP TurboIMAGE is supported currently.

- Data Item Name

  The HP TurboIMAGE data item name; the field specification name is upshifted and underscore characters are changed to hyphens.

- Sub-item Count

  Equal to the **Repeated Field** of the base field specification.

- Type Designator

  The HP TurboIMAGE type designator; the default value is derived from the **Storage Type** and **Edit Code** of the base field. Refer to the table in the following slide.

  The allowable values are **E**, **I**, **J**, **K**, **P**, **R**, **U**, **X** and **Z**.

- Sub-item Length

  The length of the base field specification.

When the screen is committed, a number of HP TurboIMAGE requirements are checked. The checks ensure that:

- *Data Item Name* is from 1 to 16 valid characters, starting with an alphabetic character.

- *Sub-item Count * Sub-item Length * type designator base length* is a multiple of 2.

TIMAGE09: The Developer Application - Extended Field Specification

The default value for the **Type Designator** field is derived from the following table:

HP ALLBASE/4GL		HP TurboIMAGE	
Edit Code	Storage Type	Data Type	Sub-item Length
X,A or K	-	X	Field Length + pad
U	-	U	Field Length + pad
Q	-	X	2
D or T	-	X	8
N or S	C	X	Field Length + pad
N or S	F	K	2
N or S	G	K	4
N or S	I	I	1
N or S	L	I	2
N or S	P	P	Field Length + pad

The allowable values for the **Type Designator** field are:

Code	Meaning
E	IEEE real number
I	2's complement (integer)
J	Same as I, except with COBOL range restrictions
K	Absolute binary quantity
P	Packed decimal (BCD, or Binary Coded Decimal)
R	'Classic' HP 3000 real number
U	Uppercase alphabetic character
X	Any printable character
Z	A zoned decimal format number

Student Notes

The Developer Application - Extended Field Specification

The default value for the **Type Designator** field is derived from the following table:

HP ALLBASE/4GL		HP TurboIMAGE	
Edit Code	Storage Type	Data Type	Sub-item Length
X,A or K	-	X	Field Length + pad
U	-	U	Field Length + pad
Q	-	X	2
D or T	-	X	8
N or S	C	X	Field Length + pad
N or S	F	K	2
N or S	G	K	4
N or S	I	I	1
N or S	L	I	2
N or S	P	P	Field Length + pad

In the above table, **pad** is the smallest integer that can be used so that the sub-item length is a multiple of two. If the default value is not suitable, the developer can overtype to enter the desired type.

The allowable values for the **Type Designator** are:

Code	Meaning
E	IEEE real number
I	2's complement (integer)
J	Same as I, except with COBOL range restrictions
K	Absolute binary quantity
P	Packed decimal (BCD, or Binary Coded Decimal)
R	'Classic' HP 3000 real number
U	Uppercase alphabetic character
X	Any printable character
Z	A zoned decimal format number

Notice that **E**, **J** and **R** are not put in as default values by HP ALLBASE/4GL.

TIMAGE10 - SLIDE: The Developer Application - File Definition

HP TurboIMAGE INTERFACE

File Definition Screen

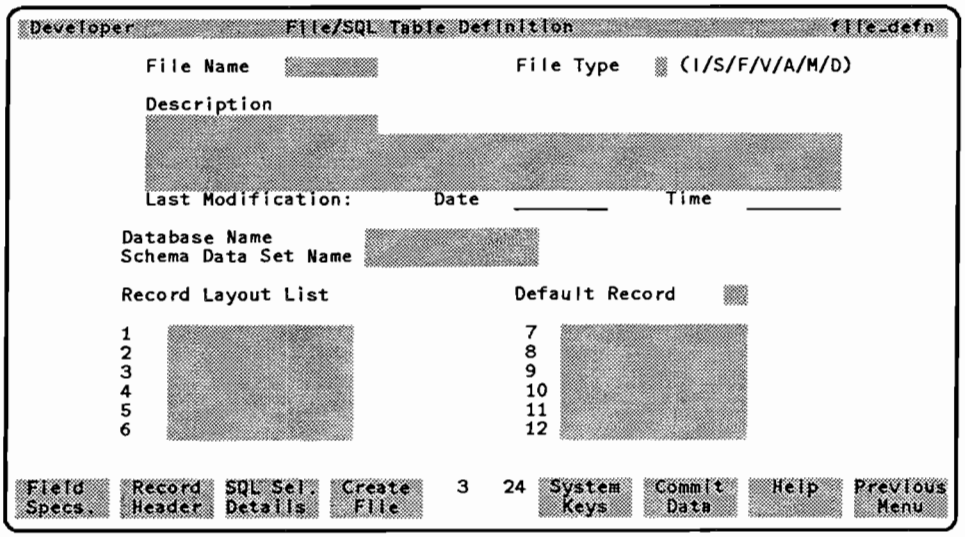
- The File Definition Screen allows the developer to define a data set within HP ALLBASE/4GL.

- HP TurboIMAGE data sets cannot be created within HP ALLBASE/4GL but are created via the schema file.

Student Notes

The Developer Application - The File Definition screen

On the **File Definition** screen, the developer enters the name of the file/table/dataset and then specifies the type. If the developer enters **A** (Automatic Master), **M** (Manual Master) or **D** (Detail), the HP TurboIMAGE window is displayed, as shown below:



The **Database Name** field must contain the HP ALLBASE/4GL name of the HP TurboIMAGE database, **NOT** the external name of the database.

The **Schema Data Set Name** field must contain the name of the data-set as it is known to the HP TurboIMAGE database. The name is case sensitive and all lowercase characters are automatically converted to uppercase. The field may contain any of the special characters (+, -, *, /, ?, ', #, %, &, @) allowed by HP TurboIMAGE.

The fields for the **Default Record** and **Record Layout List** are the same as for other data managers.

TIMAGE11 - SLIDE: The Developer Application - Logic Commands

HP TurboIMAGE INTERFACE

Logic Commands

The following commands can be used with HP TurboIMAGE data sets:

- DM IMAGE
- FILE
- MODE
- TRANSACT
- UPDATE

Student Notes

The Developer Application - Logic Commands

The following logic commands can be used when accessing an HP TurboIMAGE database:

- DM IMAGE
- FILE
- MODE
- TRANSACT
- UPDATE

The new command added for the HP TurboIMAGE interface is **DM IMAGE**.

TIMAGE12 - SLIDE: The Developer Application - Logic Commands

The DM IMAGE Command

HP TurboIMAGE INTERFACE

Logic Commands

DM IMAGE

The DM IMAGE command has the following arguments:

*CLOSE
*LOCK
*MODE
  *MODLOCK *READLOCK
  *UPDSHAR *READMOD
  *MODEXCL *READEXCL
  *MODREAD *READSHAR
*UNLOCK

Student Notes

The Developer Application - Logic Commands

The DM IMAGE Command

DM is used as a prefix to logic commands specific to individual data managers. Currently, HP ALLBASE/4GL only offers data manager specific commands for the HP TurboIMAGE data manager by means of arguments to the **DM IMAGE** command. The commands are:

DM IMAGE *CLOSE

Closes a whole database.

DM IMAGE *LOCK

Allows multiple predicate locks to be requested explicitly against an HP TurboIMAGE database.

DM IMAGE *MODE

Set the access mode for the entire database. The allowable modes are:

- *MODLOCK - shared modify with enforced locking
- *UPDSHAR - update, allow concurrent update
- *MODEXCL - modify, exclusive
- *MODREAD - modify, allow concurrent read
- *READLOCK - read, allow concurrent modify with enforced locking
- *READMOD - read, allow concurrent modify
- *READEXCL - read, exclusive
- *READSHAR - read, allow concurrent read

The access permissions requested in a **DM IMAGE *MODE** command for a database must be a superset of those requested in **MODE** commands for any of its datasets.

The default mode is ***READLOCK** (HP TurboIMAGE mode 6).

DM IMAGE *UNLOCK

Release all locks held on this database.

TIMAGE13 - SLIDE: The Developer Application - Logic Commands

The MODE Command

HP TurboIMAGE INTERFACE

Logic Commands

MODE

The MODE Command specifies the ways files are used in the current process.

*READ
*WRITE
*LOCK
*UNLOCK

Student Notes

The Developer Application - Logic Commands

The MODE Command

The MODE command specifies the way files are used in the current process. The following arguments are provided for access to HP TurboIMAGE databases:

MODE *READ

Opens a data set in *read* mode. The data can be read but not modified or updated.

MODE *WRITE

Opens a data set in *read/write* mode; the data can be read, modified or updated.

MODE *LOCK

Opens a data set in *read/write* mode; the data can be read, modified or updated.

MODE *UNLOCK

Opens a data set in *read/write* mode; the data can be read, modified or updated.

Locking

The modes ***WRITE**, ***LOCK** and ***UNLOCK** differ in the locking which they allow. For a more complete discussion of locking, refer to the section entitled **Locking HP TurboIMAGE Data Sets.**

TIMAGE14 - SLIDE: The Developer Application - Logic Commands

The FILE Command

HP TurboIMAGE INTERFACE

Logic Commands

FILE

FILE { *BUFFER, *CLOSE, *MODIFY, *UNLOCK }	file_ref [;command]
FILE { *NEXT, *PREVIOUS }	[*NOLOCK] file_ref [*INDEX=index_name][;command]
FILE { *DELETE, *FIND, *FIRST, *INSERT, *LAST, *READ [*REC = recno], *WRITE }	[*NOLOCK] file_ref [*INDEX=index_name] [*KEY=key][;command]

Australian Software Operation
timage14

HEWLETT PACKARD

Student Notes

The Developer Application - Logic Commands

The FILE Command

The interface to HP TurboIMAGE is similar to the interface to KSAM as most accesses will use the **FILE** command; this provides the following operations on **automatic master sets, manual master data sets** and **detail data sets:**

*BUFFER	*LAST
*CLOSE	*MODIFY
*DELETE	*NEXT
*FIND	*PREVIOUS
*FIRST	*READ
*FIND	*UNLOCK
*INSERT	*WRITE

Most of the arguments support the ***INDEX=index_name** option; this allows searching using other than the default index.

Some of the arguments support the ***KEY=key** option. The ***READ** argument also supports the ***RECNO=recno** option (seen previously for fixed length serial files).

As with ISAM/KSAM files, serial files and SQL tables, an optional error command is allowed for each file operation. The error command will be performed if ***IOSTATUS** contains a **non-zero** value.

TIMAGE15: The Developer Application - Logic Commands

The FILE command

***BUFFER - clears the specified buffer of all data**

***CLOSE - closes the nominated data set**

***DELETE - deletes a record from the data set**

Detail data sets:
Master data sets:
Secondary record migration:
Automatic master data set records:

***FIND - finds first record with key given by *KEY=key**

Master data sets:
Detail data sets:
Incomplete keys:
Error status:

***FIRST - reads the first record for the current index**

Serial reads:

***INSERT - adds a new record**

Detail data sets:
Automatic master data sets:
Manual master data sets:
Master data sets:

***LAST - reads the last record for the current index**

Serial reads:

The Developer Application - Logic Commands

The FILE command

*BUFFER

Clears the specified buffer of all data. Unless the data set has been opened using **MODE *UNLOCK**, clearing the buffer also unlocks the current record.

*CLOSE

Closes the nominated data set.

Data sets are automatically closed when a process exits and also when the entire database is closed with the **DM IMAGE *CLOSE** command.

*DELETE

Deletes a record from the data set.

Detail data set records must be read before they can be deleted.

Master data sets act as indexes to details data sets, so master data set records cannot be deleted if any detail data set records contain the primary key value of the master data record.

Deletion of a record from a manual master data set may cause a secondary record to migrate. In this case, HP ALLBASE/4GL ensures that the next serial operation re-establishes the data pointer on the same record.

Automatic master data set records are automatically deleted if all of the detail data set records are deleted.

*FIND

Finds the first record whose key is equal to the value of *key* .

For master data sets, the records are read in serial order.

For detail data sets, this operation is used to place the data set pointer at the beginning of a chain of detail data set records with the same key value; ***NEXT** and ***PREVIOUS** operations should then be used to move the pointer along the chain.

***FIND** does not place data into the record buffer; the ***NEXT** command should be used to do that.

Records can be found from incomplete keys, in which case the first record beginning with the value will be found.

The error status **19111** is returned if a matching record is not found.

*FIRST

Reads the first record in the data set for the current index.

The data set can be read serially if the ***KEY** option is omitted.

*INSERT

Adds a new record to the data set.

This operation adds new records to detail data sets.

A new record may be automatically added to any automatic master data sets linked to the detail data set.

An error will occur if any manual master data sets linked to the detail data set do not contain a record with the key value of the detail data set.

Master data sets do not permit duplicate primary key values.

*LAST

Reads the last record in the data set for the current index.

The data set can be read serially if the ***KEY** option is omitted.

Intentionally blank

TIMAGE16: The Developer Application - Logic Commands

The FILE command

*MODIFY - changes an existing record in the data set

Automatic master data sets:

Master data set key values:

*NEXT - serially reads the next record in the data set

Chain reads:

Migrating secondaries:

*PREVIOUS - serially reads the previous record in the data set

Chain reads:

Migrating secondaries:

*READ - reads the record with key given by *KEY=key

*REC=recno:

Migrating secondaries:

*UNLOCK - releases all locks on a data set

MODE *READ:

Caution

A ***UNLOCK** operation to unlock a data set will release **ALL** locks held in the current process for the same database.

*WRITE - writes a new record or modifies an existing record

Automatic master data sets:

Detail data sets:

Manual master data sets:

The Developer Application - Logic Commands

The FILE Command

*MODIFY

Changes an existing record in the data set; the record must be read before it can be modified.

Automatic master data sets that are linked to a detail data set that is modified are automatically updated.

Master data set key values can only be modified if no detail sets linked to the master data set have records containing the key value. This operation may cause a secondary record to migrate.

*NEXT

Serially reads the next record in the data set.

For detail data sets it is possible to perform a **chain read**; the *NEXT must follow either a *FIND or a *READ *KEY= operation.

For master data sets, a migrating secondary may cause a *NEXT operation to re-read the current record.

*PREVIOUS

Serially reads the previous record in the data set.

For detail data sets it is possible to perform a **chain read**; the *PREVIOUS must follow either a *FIND or a *READ *KEY= operation.

For master data sets, a migrating secondary may cause a *PREVIOUS operation to re-read the current record.

*READ

Reads the record in the data set whose key value, for the current index, exactly matches the value specified by ***KEY=key**. If the ***KEY=** is not used, HP ALLBASE/4GL uses the current value in the record buffer field for the current index.

The ***REC=recno** argument causes the ***READ** operation to read the record specified by *recno*.

For master data sets, a migrating secondary may cause a ***READ** operation to re-read the current record.

*UNLOCK

Releases all locks on a data set that has been accessed under **MODE *LOCK, MODE *WRITE** or **MODE *UNLOCK**. No action is taken if the data set is accessed under **MODE *READ**.

Caution

A ***UNLOCK** operation to unlock a data set will release **ALL** locks held in the current process for the same database.

---

*WRITE

Writes a new record or modifies an existing record within a manual master data set.

***WRITE** cannot be used with automatic master data sets because they are automatically maintained by HP TurboIMAGE.

For detail data sets, the ***INSERT** operation should be used instead of ***WRITE**.

For manual master data sets:

- The data set record is modified if the already record exists.
- A new record is added if the record does not exist.

Intentionally blank

TIMAGE17 - SLIDE: The Developer Application - Logic Commands

The FILE Command - Error Codes and Error Conditions

HP TurboIMAGE INTERFACE

Logic Commands

FILE – Error Handling

19000	MPE file access error
19013	Access permission denied to file
19100	Duplicate Primary Key error
19110	Beginning or end of file reached during *NEXT or *PREVIOUS
19111	Record not found
19113	Exclusive locking error
19115	Beginning or end of detail data set chain reached during *NEXT or *PREVIOUS

Student Notes

The Developer Application - Logic Commands

The FILE Command - Error Codes and Error Conditions

The following table summarises the most common file error values returned to the communication area field *IOSTATUS.

19000	MPE XL file access error detected
19013	MPE XL error. Access permission denied to file.
19100	Duplicate primary key error.
19107	Record locked error.
19110	Beginning or end of file reached during *NEXT or *PREVIOUS
19111	Record not found
19112	Record not read
19113	Exclusive locking error
19115	Beginning or end of detail data set chain reached during *NEXT or *PREVIOUS
19130	File locking error.

In addition, any errors returned by the HP TurboIMAGE data manager are recorded in the ***IMSTATUS** communication area field, and any errors returned by any data managers other than HP ALLBASE/4GL are written to the ***IMSTATUS** field. The value in these fields is zero if no error occurs. If errors do occur, these fields contain a number corresponding to an HP TurboIMAGE status array, which represents status information about the last HP TurboIMAGE library procedure performed.

The optional error command performed if ***IOSTATUS** is **non-zero** can be any of the following HP ALLBASE/4GL logic commands:

- ENTER
- EXIT
- EXTERNAL
- MESSAGE
- PROCEED
- SERIES
- TOP
- VISIT
- ZIP

Laboratory Exercise

The students can now do the first of the laboratory exercises for this module.

TIMAGE18 - SLIDE: The Developer Application - Logic Commands

HP TurboIMAGE INTERFACE

Logic Commands

UPDATE

Writes all modified file buffers to disk

TRANSACT

TRANSACT *MEMO - Is used to include comments in an HP TurboIMAGE log record.

TRANSACT *UNDO - Has no effect on HP TurboIMAGE databases.

Student Notes

The Developer Application - Logic Commands

The UPDATE Command

The update command for HP TurboIMAGE file buffers operates the same as for KSAM and serial file buffers; only modified buffers are written to disk.

The TRANSACT Command

The **TRANSACT** command can be used to trigger appropriate HP TurboIMAGE transaction logging operations.

The arguments are:

- *BEGIN
- *END
- *MEMO
- *UNDO

The new command **TRANSACT *MEMO** is used to include comments in an HP TurboIMAGE log record; it has no effect on non-HP TurboIMAGE files.

TRANSACT *UNDO has no effect on HP TurboIMAGE databases.

TIMAGE19 - SLIDE: The Developer Application - Locking

HP TurboIMAGE INTERFACE

Locking Data Sets

HP ALLBASE/4GL provides three levels of locking with HP TurboIMAGE:

* Database locking
* Data Set locking
* Record locking

Locking is determined by the DM IMAGE and MODE commands.

If DM IMAGE is not used, Implicit Locking is used as determined by the MODE command. Implicit Locking can be disabled.

Student Notes

The Developer Application - Locking HP TurboIMAGE Data Sets

Data set locking is determined by the **DM IMAGE** and **MODE** commands. If the **DM IMAGE** command is not used, HP ALLBASE/4GL uses **implicit locking** which is determined by the **MODE** command. It is best to use the **DM IMAGE** command to make best use of HP TurboIMAGE locking capabilities.

HP TurboIMAGE provides three levels of locking; database locking, data set locking and record locking. All three levels of locking are available within HP ALLBASE/4GL using the **DM IMAGE *LOCK** command.

Implicit Locking

Implicit data set locking in HP ALLBASE/4GL depends on both the data set operation used with the **FILE** command, and the data set access mode specified in the **MODE** command.

The following locking modes are provided by HP ALLBASE/4GL for HP TurboIMAGE data sets:

MODE *READ

No locking performed.

MODE *WRITE

Apply a set level lock to the data set concerned unless a lock is already in place on behalf of the same record. If a lock is already in place because a different record was accessed through the same file buffer, that lock is released first. Locks are released under the same conditions as for HP-UX ISAM.

MODE *UNLOCK

Apply a set level lock to the data set concerned unless a lock is already in place on behalf of the same record. Locks are only released under the same conditions as for HP-UX ISAM.

Integrity is guaranteed as another process or user cannot lock the same data set while a lock is current.

MODE *LOCK

The same locking applies as for ISAM.

Releasing locks on a data set

During use of any of the modes above, whenever locks held on one data set are released, all locks on all other data sets in the same database are released.

This is an immutable HP TurboIMAGE constraint. It means that use of **MODE *LOCK** on a data set will be subverted by use of ***WRITE** or ***UNLOCK** on any other data set in the same database. A similar issue exists with **MODE *UNLOCK**.

Disabling Implicit Locking

If a **DM IMAGE *LOCK** command is used, all implicit locking of that database will be disabled until a matching **DM *IMAGE *UNLOCK** or **FILE *UNLOCK**.

Whenever a database is opened with ***MODEXCL** (mode 3), all implicit locking of that database is disabled.

The ***NOLOCK** parameter will also suppress implicit locking for any **FILE** command.

Intentionally blank

TIMAGE20 - SLIDE: The Developer - Reports and Module Builder

HP TurboIMAGE INTERFACE

Reports / Module Builder

Reports: HP TurboIMAGE data sets can be used as the primary file for HP ALLBASE/4GL reports. They can also be used as link files in automatic file linkages.

Module Builder: HP TurboIMAGE data sets can be used as the main files for modules created by Module Builder. They can also act as secondary or link files.

Student Notes

The Developer Application - Reports and Module Builder

Reports

HP TurboIMAGE data sets can be used as the primary file for HP ALLBASE/4GL reports. File linkages are supported from HP TurboIMAGE data sets to other HP TurboIMAGE data sets and to KSAM files. HP TurboIMAGE data sets can also be used as the link files for reports based on KSAM files.

Module Builder

HP TurboIMAGE data sets can be used as the primary file for HP ALLBASE/4GL reports. Obviously it does not make any sense to use automatic master data sets (file type **A**) as the primary file for a module.

HP TurboIMAGE data sets can also be used as secondary or link files.

TIMAGE21 - SLIDE: The Developer Application - Database Access

HP TurboIMAGE INTERFACE

Database Access

HP TurboIMAGE databases and data-sets cannot be created within HP ALLBASE/4GL; use DBSCHEMA and DBUTIL. These can be accessed via function key.

The HP TurboIMAGE database will automatically be OPENED the first time one of its data sets is READ, WRITTEN or LOCKED; the data set will also be opened.

Student Notes

Miscellaneous - Database Access

Creating and Defining an HP TurboIMAGE Database or Data Set

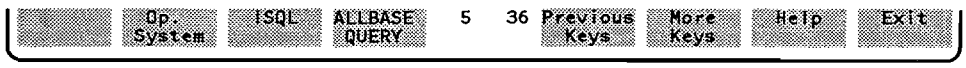
HP ALLBASE/4GL provides no means for creating HP TurboIMAGE databases or data sets; these tasks are performed using the HP TurboIMAGE utilities **DBSCHEMA** and **DBUTIL.**

For use in an application, HP ALLBASE/4GL only requires that HP TurboIMAGE databases and data sets have been completely defined using the **Database Definition, Database Access Parameters** and **File Definition** screens.

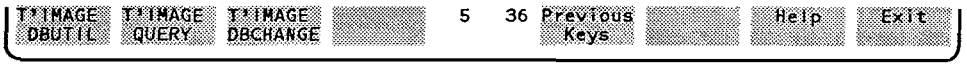
Function Keys

HP ALLBASE/4GL provides developer and administrator access to **TurboIMAGE/DBUTIL, TurboIMAGE/QUERY** and **TurboIMAGE/DBCHANGE** via function keys.

The existing **developr_keys_2** and **admin_keys_2** have been changed to:



The new sets **developr_keys_3** and **admin_keys_3** are:



Opening an HP TurboIMAGE Database or Data Set

HP ALLBASE/4GL does not provide a means for explicitly opening an HP TurboIMAGE database or data set. The HP TurboIMAGE data set and its associated database are automatically opened the first time they data set is accessed.

TIMAGE22 - SLIDE: The Developer Application - Database Access

HP TurboIMAGE INTERFACE

Database Access - Closing databases & data-sets

Any open data-sets will be closed automatically when an HP ALLBASE/4GL process exits.

Any open databases and data-sets will be closed automatically when an HP ALLBASE/4GL application exits.

An open HP TurboIMAGE database can be explicitly CLOSED by the DM IMAGE *CLOSE command.

An open HP TurboIMAGE data set can be explicitly CLOSED by using the FILE *CLOSE command.

Student Notes

The Developer Application - Database Access

Closing an HP TurboIMAGE Database or Data Set

HP ALLBASE/4GL provides automatic and developer-controlled means for closing HP TurboIMAGE databases and individual HP TurboIMAGE data sets.

Automatic operations

Any open HP TurboIMAGE data-sets are automatically closed whenever an HP ALLBASE/4GL process exits.

When an HP TurboIMAGE database is closed, any open data sets are automatically closed. The database is closed automatically when the HP ALLBASE/4GL application terminates, thus closing any open data sets.

Manual operations

As well as the automatic operations, HP ALLBASE/4GL gives the developer manual control over closing an HP TurboIMAGE data set or database.

An HP TurboIMAGE database can be closed at any time using the **DM IMAGE *CLOSE** command.

HP TurboIMAGE data sets can be closed at any time using the **FILE *CLOSE** command.

TIMAGE23 - SLIDE: HP ALLBASE/4GL Utilities

HP TurboIMAGE INTERFACE

Utilities

HP4STOA and HP4ATOS have been enhanced
to support HP TurboIMAGE.

A new utility, HP4TUPLD, can be used to load definitions
from the schema file into HP ALLBASE/4GL.

Student Notes

HP ALLBASE/4GL Utilities

The utilities **HP4STOA** and **HP4ATOS** have been enhanced to support the new and enhanced HP ALLBASE/4GL entities required for the HP TurboIMAGE interface.

HP ALLBASE/4GL also provides a new utility called **HP4TUPLD** for working with HP TurboIMAGE databases. **HP4TUPLD** is used in the process of loading schema **data-item** and **data-set** details into the HP ALLBASE/4GL dictionary. It is also possible to load **database** details from the schema file into the HP ALLBASE/4GL administrator application.

It is used in conjunction with the utilities **HP4ATOS**.

TIMAGE24 - SLIDE: Utilities - HP4STOA

HP TurboIMAGE INTERFACE

Utilities

The **HP4STOA** utility is used to *unload* an HP ALLBASE/4GL application to an ASCII file.

hp4stoa "-a transfer -u developr > transfer.asf"

Student Notes

Utilities - HP4STOA

HP4STOA is used to produce the contents of an HP ALLBASE/4GL application in ASCII format.

The syntax is:

```
hp4stoa "-a application_name:password -u user_name:password [option ...]"
```

HP4STOA has been enhanced to be able to unload **Storage Type** information from the **Field Specifications** screen, and to unload the **Data Mgr. Specific Field Attributes** information.

The new object options to support HP TurboIMAGE are:

xb Database (from new database definition screen)

xp Database access parameters (from database access parameters screen)

HP4STOA normally directs output to the screen but the output can be directed to a file using the following syntax:

```
hp4stoa "-a application_name:password -u user_name:password [option ...] >asf_file"
```

Note

An **ASF** file is an ASCII file produced by **HP4STOA** containing an HP ALLBASE/4GL application. HP ALLBASE/4GL applications are normally stored in the **S-files**, thus the ASCII versions are called **ASCII S-Files** or **ASF** files.

On HP-UX, ASF files are normally given the extension **.asf**. Thus, the **HP4STOA** output for the application **test** would be saved to the file **test.asf**.

On MPE XL, it is usual to create a group called **ASF**; thus, the **HP4STOA** output for the application **test** would be saved to the file **TEST.ASF**, the file **TEST** in the group **ASF**.

TIMAGE25 - SLIDE: Utilities - HP4ATOS

HP TurboIMAGE INTERFACE

Utilities

The **HP4ATOS** utility is used to *upload* an HP ALLBASE/4GL application from an ASCII file.

hp4atos "-a transfer -u developr < transfer.asf"

Student Notes

Utilities - HP4ATOS

HP4ATOS is used to write the contents of an ASCII file (in correct format) into an application in the HP ALLBASE/4GL S-files.

The syntax is:

```
hp4atos "[option ...] < asf_file"
```

HP4ATOS has been enhanced to support **Storage Type**, **Data Mgr. Field Specific Attributes** information for the developer application. It also supports **Database Details** and **Database Access Parameters** for the administrator application.

There are no syntax changes.

For a complete explanation of **HP4STOA** and **HP4ATOS**, refer to the module **Utilities and the External World.**

Note

An **ASF** file is an ASCII file produced by **HP4STOA** containing an HP ALLBASE/4GL application. HP ALLBASE/4GL applications are normally stored in the **S-files**, thus the ASCII versions are called **ASCII S-Files** or **ASF** files.

On HP-UX, ASF files are normally given the extension **.asf**. Thus, the **HP4STOA** output for the application **test** would be saved to the file **test.asf**.

On MPE XL, it is usual to create a group called **ASF**; thus, the **HP4STOA** output for the application **test** would be saved to the file **TEST.ASF**, the file **TEST** in the group **ASF**.

TIMAGE26 - SLIDE: Utilities - HP4UPLD

HP TurboIMAGE INTERFACE

Utilities

The **HP4TUPLD** utility is used to *upload* the HP TurboIMAGE schema file to HP ALLBASE/4GL dictionary definitions.

hp4tupld "-a transfer -o d schema > transfer.asf"

hp4atos "-a transfer -u developr < transfer.asf"

Student Notes

Utilities - HP4TUPLD

HP4TUPLD can be used in the process to *upload* definitions to the HP ALLBASE/4GL dictionary from an HP TurboIMAGE schema file.

HP4TUPLD will read the schema file and output the dictionary definitions in a format which can be written into the HP ALLBASE/4GL application using the utility **HP4ATOS**.

The syntax is:

```
hp4tupld "-a <application> -o <object_type> [schema_file]"
```

where **object_type** is **x** for database definitions and access parameters and **d** for dictionary definitions.

Output of **HP4TUPLD** is normally directed to the screen; to direct the output to a file able to be processed by **HP4ATOS**, use the syntax:

```
hp4tupld "-a <application> -o <object_type> [schema_file] > asf_file"
```

For example, to *upload* dictionary items from the database **TRANSFER**, for which the schema file is **SCHEMA**, into the file **TRANSFER.ASF**, the syntax is:

```
hp4tupld "-a transfer -o d schema > transfer.asf"
```

To write the dictionary definitions into the application **transfer** using **HP4ATOS**, the syntax is:

```
hp4atos "-a transfer -u developr < transfer.asf"
```

Laboratory Exercises

The students should now do the final laboratory exercise.

Database Setup

The HP TurboIMAGE Database 'CTC??B'

For the lab. exercises, you will build an HP TurboIMAGE database called **CTC??B**. Replace the **??** with your two digit student number. From now on in these instructions, the digits **00** will be used.

The Schema File

The schema for the HP TurboIMAGE database **CTC00B** is included below. Create it using **HPEDIT** or, if you are brave, you can use **EDITOR**. The schema file should be created in the **HP4TI** group with the name **PROD00B**.

```
BEGIN DATA BASE CTC00B, LANGUAGE:AMERICAN;
PASSWORDS: 10 DEVELOP;

ITEMS:

SUPPLIER-NO, I (/10);
PRODUCT-NO, X6 (/10);
DESCRIPTION, X30 (/10);
LEAD-TIME, I (/10);

SETS:

NAME: PRODUCT, MANUAL (/10);
ENTRY: PRODUCT-NO (0),
DESCRIPTION,
SUPPLIER-NO,
LEAD-TIME;
CAPACITY: 200;

END.
```

Creating the Root File

The database **root file** is created by running the utility **DBSCHEMA**, as follows:

```
FILE DBSTEXT=PRODOOB
FILE DBSLIST=OUTFILE
DBSCHEMA.PUB.SYS;PARM=3
```

Creating the Root File and the database files

To create the remainder of the database, run the utility **DBUTIL** and use the **CREATE** command, as follows:

```
DBUTIL.PUB.SYS
CREATE CTCOOB
EXIT
```

Additions to the HP ALLBASE/SQL Database

In this exercise, you are to add another **DBEFILESET** and **DBEFILE** to your existing HP ALLBASE/SQL database **st??base**. You can enter the commands directly from within **isql** or you can use an editor to create a new script **st??add** which can be run from within **isql**. To make the additions to the database, run **ISQL** and from the prompt, as shown, type:

```
isql => start st00add;
```

The 'st00add' file

```
connect to 'st00base';
create dbefileset prodFS;

create dbefile prodDBEfile with pages = 100,
name = 'prodfile', type = mixed;

add dbefile prodDBEfile to dbefileset prodFS;
commit work;
exit;
```

Uploading Definitions to HP ALLBASE/4GL

In this exercise, you will extract definitions from the HP TurboIMAGE schema file **PROD00B** and load them into your application. In the following discussion, the name **ctc92440** will be used for the application. Replace that with the name of your own application.

Follow the steps below:

1. Run HP ALLBASE/4GL and log-in to the administrator application.
2. Press `System Keys` `More Keys` `Op. System` to obtain a new command line.
3. Use **showvar** to check that the system variables **HP4SPATH** and **HP4TIPATH** are set appropriately.
4. Create the file **CTC00D.ASF** by typing:

   ```
   hp4tupld "-a ctc92440 -od prod00b.hp4ti  > ctc00d.asf"
   ```

   This will produce dictionary definitions.
5. Examine the contents of the file **ctc00d.asf** using **print**.
6. Load the dictionary definitions into the **ctc92440** application by typing:

   ```
   hp4atos "-a ctc92440 -u developr < ctc00d.asf"
   ```

7. Create the file **CTC00X.ASF** by typing:

   ```
   hp4tupld "-a ctc92440 -ox prod00b.hp4ti  > ctc00x.asf"
   ```

   This will produce database definitions, to be written into the HP ALLBASE/4GL administrator.
8. Examine the contents of the file **ctc00x.asf** using **print**.
9. Load the database definitions into the **ctc92440** application by typing:

   ```
   hp4atos "-u administ < ctc00x.asf"
   ```

10. Return to HP ALLBASE/4GL by typing **exit**.
11. Go to the **Database Definition** screen and check that the database **CTC00B** has been correctly defined. Add a meaningful description in the description fields.

12. Go to the **Paramters for Database Access** screen and check that the database **CTC00B** has been assigned to the application **ctc92440**. Add the password **DEVELOP**; ensure it is in upper case to match the HP TurboIMAGE definition.
13. Now log-in to the **ctc92440** application.
14. Inspect the new field specifications and the record layout and file details.
15. Generate the record layouts for the application to ensure that the new record layouts can be used.
16. Go into the **Module Builder** screen and build a module based on the file **PRODUCT**; call the module **product**.
17. Use the **screen painter** to modify the main menu and create an item to call the HP TurboIMAGE menu. Create a new menu to handle HP TurboIMAGE; create an item to call the process **mb_product**.
18. Run the new module and add about 10 records into the HP TurboIMAGE data-set.
19. Return to the developer level and examine the code produced by **Module Builder** in the process **mb_product** and the various functions **mb_product_...**

Transferring Data Between Databases

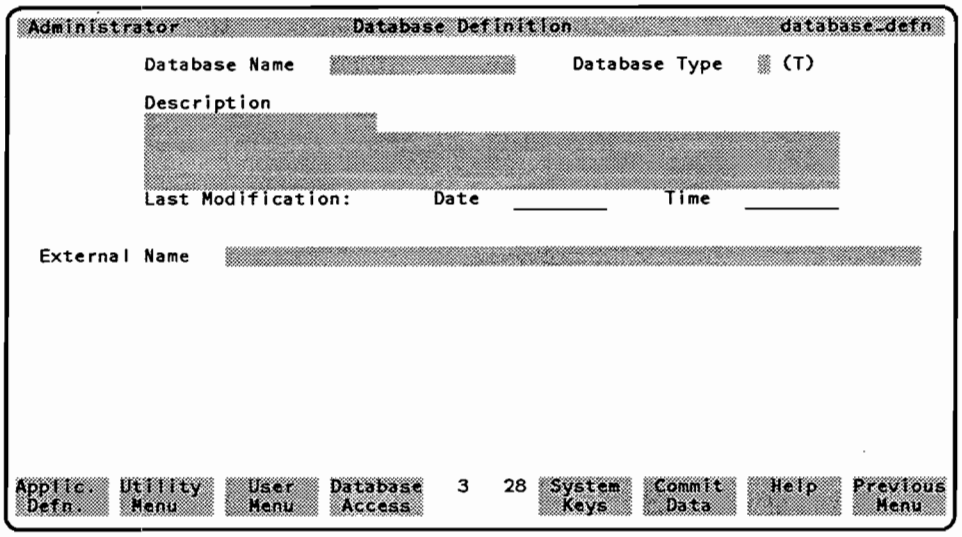
In this exercise, you will develop an HP ALLBASE/4GL logic block which will read data from the **PRODUCT** dataset in the HP TurboIMAGE database **CTC00B** and write it to the table **product** in the SQL database **st00base**.

Follow the steps below:

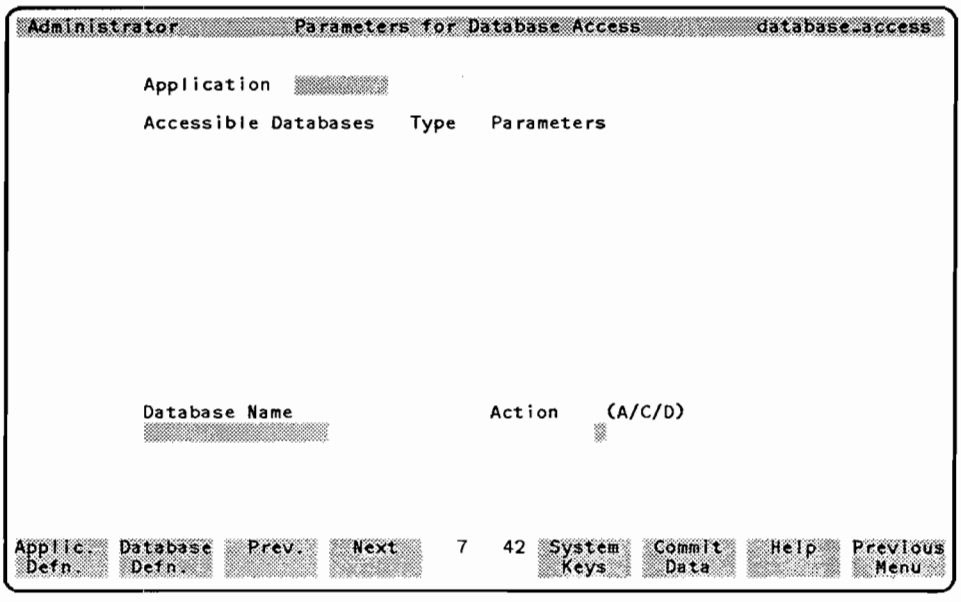
1. Define the SQL Table **product**. Use the record layout **PRODUCT**, the same record layout used in the HP TurboIMAGE data-set **PRODUCT**. Ensure that it will be in the **PRODFS** fileset. Create the table.
2. Write a process **ti_sql_transfer** to read data from the file **PRODUCT** using FILE *NEXT and insert it into the file **product** using FILE *INSERT. This should be almost identical to the process used for transferring data from the KSAM file to the SQL table.
3. Copy the screen used for the KSAM to SQL transfer and customise it.
4. Tie the process **ti_sql_transfer** to the HP TurboIMAGE sub-menu.
5. Generate and run the process.
6. Use HP ALLBASE/QUERY to produce a printed report showing the data in the table **product**.

Screen Images

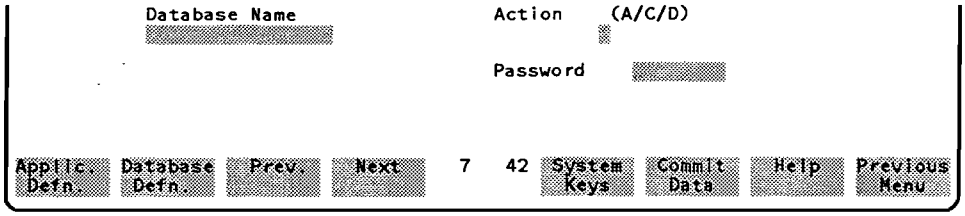
The **Database Definitions** screen is shown below:



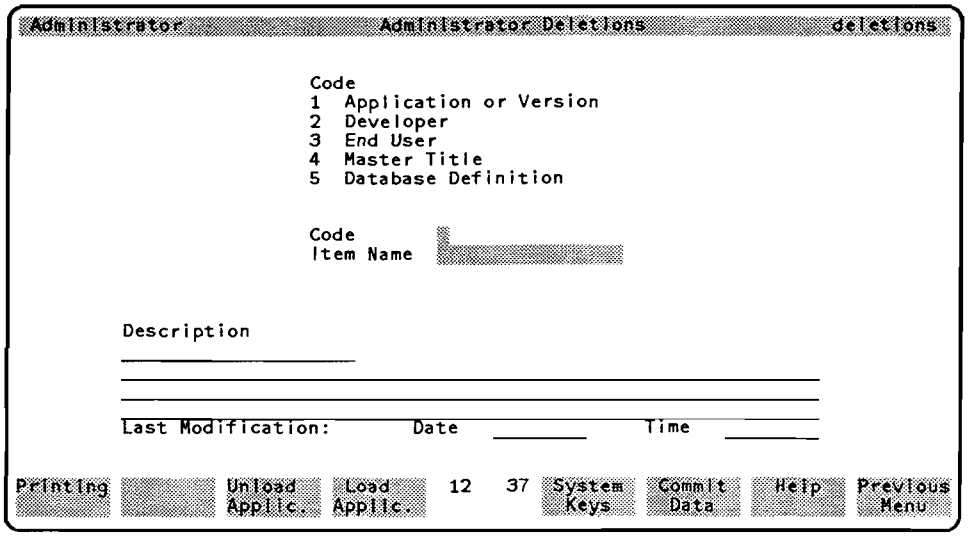
The **Database Access Parameters** screen is shown below:



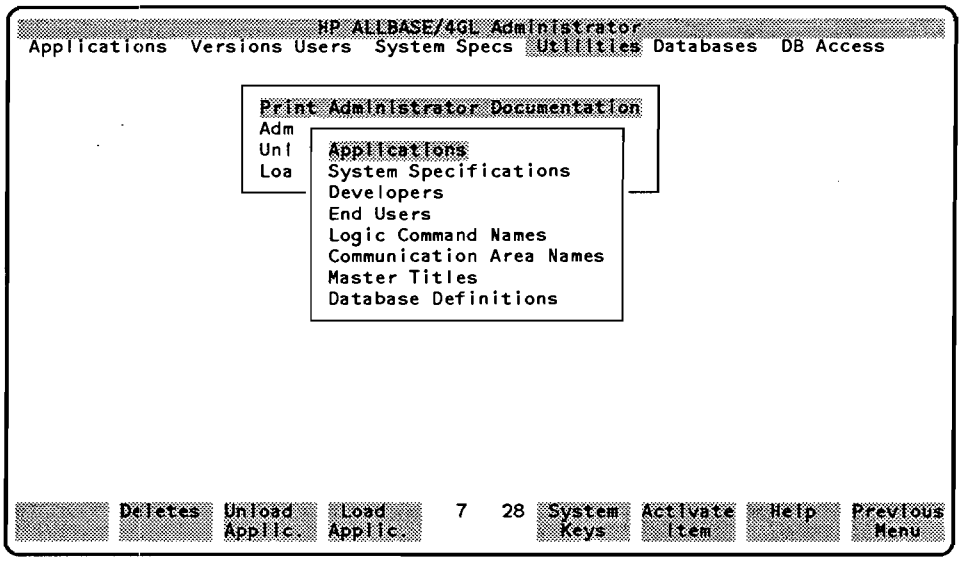
The HP TurboIMAGE window for the **Database Access Parameters** screen is shown below:



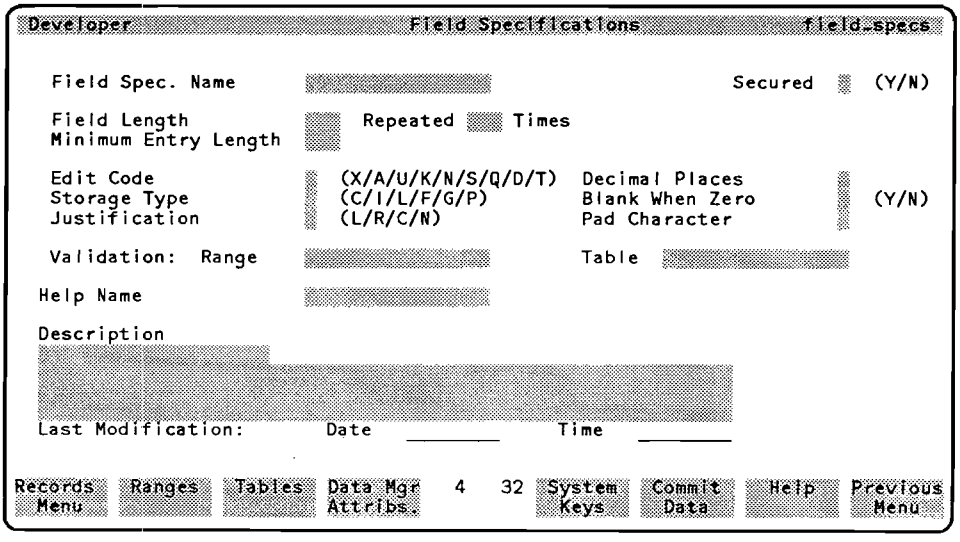
The **Administrator Deletions** screen is shown below:



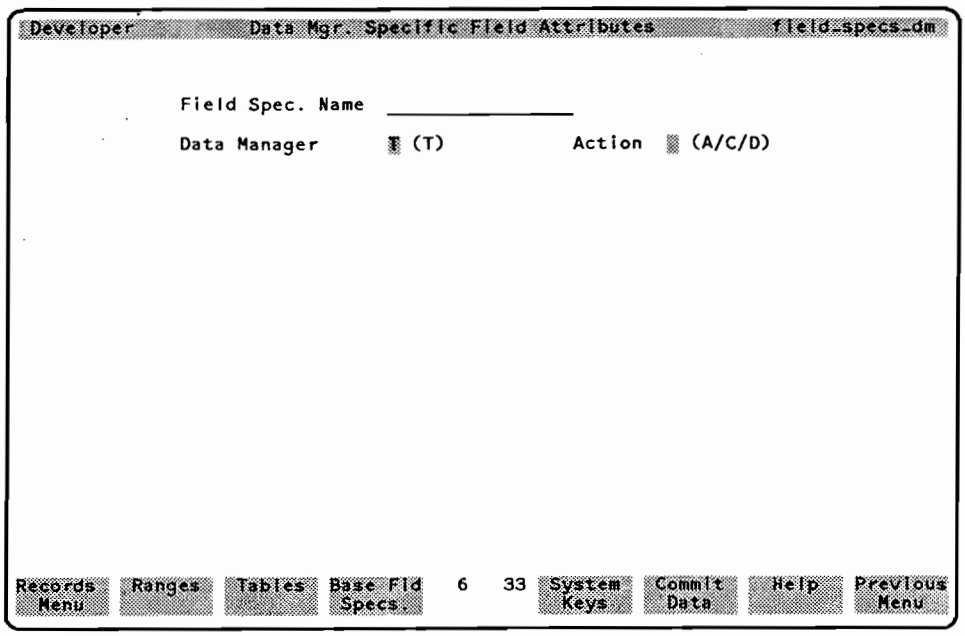
The MPE XL version of the **Administrator Documentation** screen is shown below:



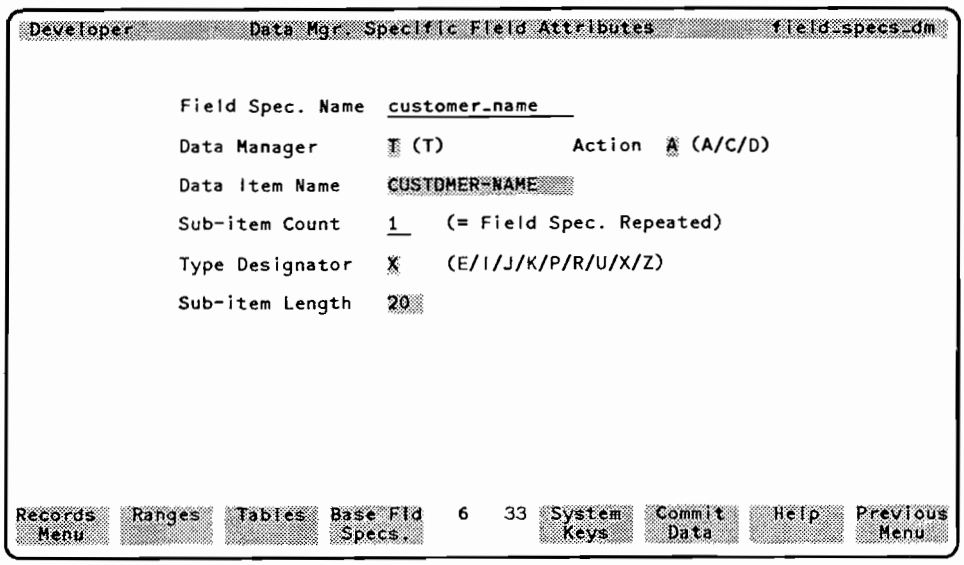
The **Field Specifications** screen is shown below:



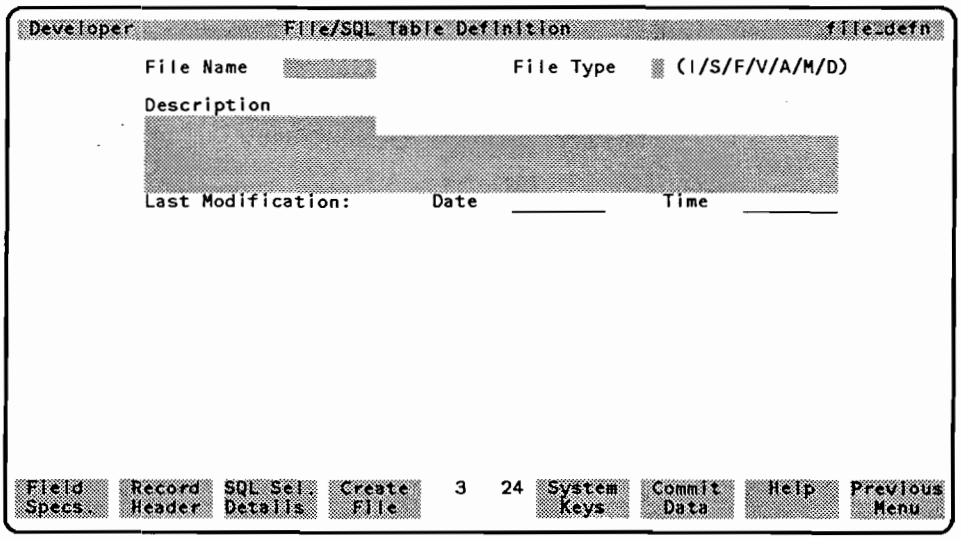
The **Data Mgr. Specific Attributes** screen is shown below:



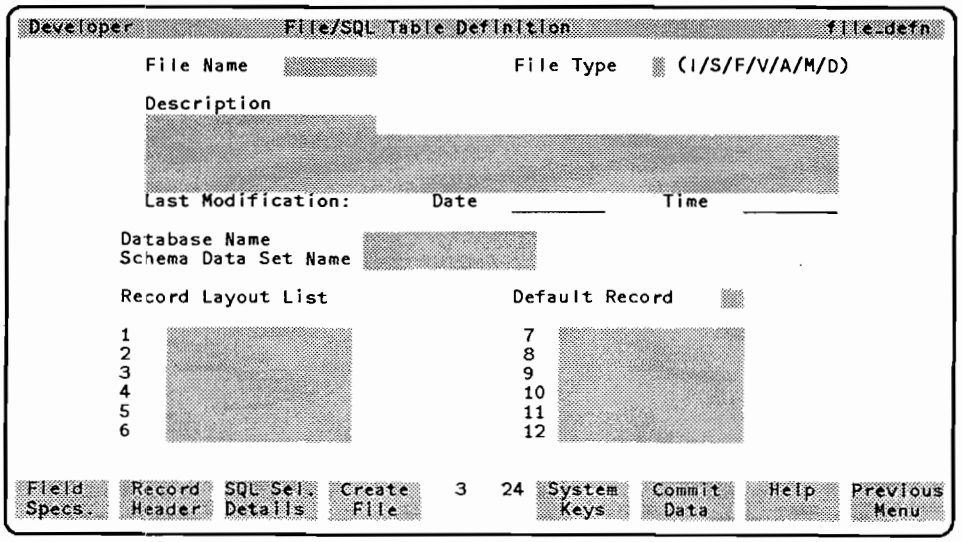
The **Data Mgr. Specific Attributes** screen with the HP TurboIMAGE window is shown below:



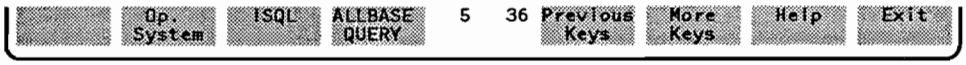
The **File/SQL Table Definition** screen is shown below:



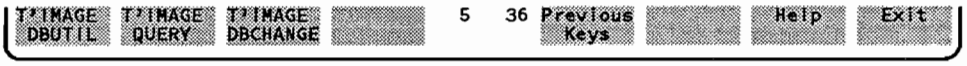
The **File/SQL Table Definition** screen with the HP TurboIMAGE window is shown below:



The **developer_keys_2** function key set is shown below:



The **developer_keys_2** function key set is shown below:



Module: HP ALLBASE/4GL Reports

Instructor Notes

REP00 - SLIDE: Module Objectives

REPORTS

Overview

Report Development

Totalling Communication Areas

A Simple Report

OBJECTIVES:

Upon completion of this module, the student will be able to:

1. Broadly describe how the HP ALLBASE/4GL report processor functions and how this relates to the database and the rest of the application.
2. List the steps involved in creating a report.
3. List some of the line types provided and how they are processed.
4. List the communication areas used for automatic totalling.
5. Develop some simple reports.

Student Notes

Module Objectives

This module will introduce the students to the HP ALLBASE/4GL reporting system.

Student Objectives

At the end of this module, each student will be able to:

- List the sections of the HP ALLBASE/4GL reporting system.
- List the considerations when using HP ALLBASE/4GL to develop a report from an ISAM/KSAM file, serial file, HP TurboIMAGE or HP ALLBASE/SQL database.
- Use the reporting system to develop a number of reports for their application.

Resource Allocation

Equipment

The following items are required for this module:

- Instructor notes
- Student notes
- Slides
- A laboratory worksheet
- Access to an HP 3000 Series 900 machine or an HP 9000 Series 800 or HP 9000 Series 300 machine with HP ALLBASE/4GL installed.
- The complete set of HP ALLBASE/4GL manuals.

Time

This module should take approximately 2 hours, with 1 hour for theory and 1 hour for the 2 practical sessions.

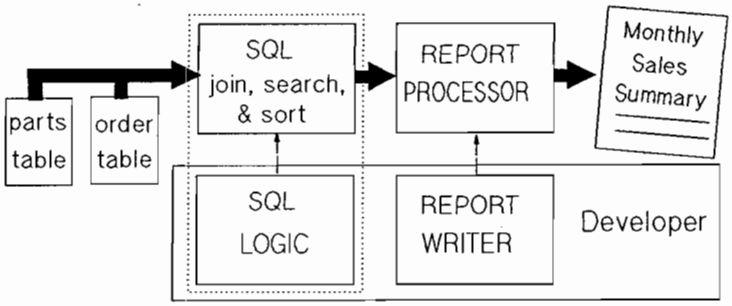
Pacing

Read through the slides reasonably quickly.

REP01 - SLIDE: Overview

REPORTS

Overview



The Report subsystem within HP ALLBASE/4GL consists of the Report Writer for defining your report, and the Report Processor which reads your selected files and processes the data according to the report definition.

As well as HP ALLBASE/4GL's automatic report processing, the SQL language provides a powerful and efficient way to search, sort and manage SQL data.

Australian Software Operation
rep01

HEWLETT PACKARD

Student Notes

Overview

The HP ALLBASE/4GL reporting system consists of two main parts, namely the **Report Writer** and the **Report Processor**. The reporting system allows the developer to produce very powerful reports without any programming, although procedural logic can be associated with various items of the report if required.

Report Writer

The developer uses the **report writer** to define a report that **report processor** will later run. There are several distinct sections to the report writer, where report parameters, selection, sorting, line groups and the report layout are defined.

Report Processor

This is the part which actually runs the report.

Data Managers

The HP ALLBASE/4GL report system can produce reports based on a number of data managers. The primary file for the report can be an ISAM/KSAM file, an SQL table, a serial file or an HP TurboIMAGE data set. Some of the data managers are very powerful, others offer minimal facilities, yet HP ALLBASE/4GL provides a consistent interface to each. The developer can also use any features made available by the data manager.

Reports based on HP ALLBASE/SQL

If an application uses an HP ALLBASE/SQL database, SQL's inbuilt selection and sorting power should be used rather than the inbuilt selection and sorting provided by HP ALLBASE/4GL. This is because HP ALLBASE/SQL will always be faster at selecting and sorting records internally in its own environment than HP ALLBASE/4GL can be; HP ALLBASE/SQL has indexes defined for the data whereas HP ALLBASE/4GL must sort and select the data without any indexes.

The inbuilt selection and sorting facilities of HP ALLBASE/4GL are really provided for reports based on the other data managers.

REP02 - SLIDE: Defining a Report

REPORTS

Defining a Report

Creating an HP ALLBASE/4GL report involves defining its attributes through the following screens:-

REPORT WRITER

- The report header screen.
- The report line header screen.
- The report painter.
- The report sorting screen.
- The selection criteria screen.
- The file linkages screen.

} Optional

The first three screens are essential for all reports.

The last three are optional and usually only used with ISAM/KSAM and HP TurboIMAGE based reports.

Student Notes

Defining a Report

To create a report the developer must always complete the **Report Header** screen, create some line groups in the **Report Line Header** screen, and must paint the report using the report painter.

Refer the students to the screen image of the **Reports Menu** at the end of this module.

Report Menu

Each choice on this menu is a separate area of the HP ALLBASE/4GL report writer. All reports must have a report header; the header specifies parameters such as the report destination, how many copies are required, and what **Before-report** and **After-report** functions should be executed.

The selection and sorting areas are both optional. Some reports may not require any selection and sorting, as you may want to report on every record in a file in the order that they exist in the file. Selection and sorting are not required when using an HP ALLBASE/SQL application.

The file linkages area is also optional. All of the data may be in the primary file or the report may use HP ALLBASE/SQL which makes it very easy to use information from a number of tables using SQL Select Lists.

REP03 - SLIDE: The Report Header Screen

REPORTS

Defining a Report

The Report Header Screen

The report header screen defines the operational environment of the report. The developer:

- defines the name of the report,
- defines the output file for the report,
- defines the primary file for the report,
- on HP-UX, specifies which printer is to be used for the report,
- on MPE XL, specifies the formal file designator,
- specifies the physical parameters for the printed output,
- names any start-of-report or end-of-report functions.

Student Notes

The Report Header Screen

Read through the slide and then ask the students to refer to the screen image of the **Report Header** screen in the **Screen Image** section at the end of this module.

Name

This is the internal HP ALLBASE/4GL name of the report.

Output File

The name of the **temporary report file** created by the HP ALLBASE/4GL **report processor** while it is running the report. It will be preserved if the developer specifies **N** for the field **Delete file after print Y/N.**

Primary File

This is the file on which the report will operate. It can be an ISAM/KSAM file, an HP ALLBASE/SQL base table or view, an HP ALLBASE/4GL select list, a serial file or an HP TurboIMAGE data set.

To operate on additional files, the file linkage area should be used. The file linkage area cannot be used for HP ALLBASE/SQL items however, and in fact there is no need to do so. In order to use data from several HP ALLBASE/SQL tables, the developer can specify a view as the primary file for the report; alternatively, an HP ALLBASE/4GL select list can be used, where the select list references fields from a number of HP ALLBASE/SQL tables.

Printer

HP ALLBASE/4GL allows the developer to specify the destination of the report. No provision is made for the end-user to re-direct a report although there are ways to do this on both HP-UX and MPE XL.

On HP-UX, a report can be sent to one of the four printers defined for the HP ALLBASE/4GL environment, or to the user's screen or to a local printer connected to the user's terminal.

End-user re-direction of reports on HP-UX is described in the article by David Williams in **ASO Support Newsletter #12** entitled **REPORTS: More flexible Developr and Administ reporting (for HP-UX only).**

On MPE XL, the developer must specify the formal file designator to which the report is to be directed. A file **equation** is normally set up before the HP ALLBASE/4GL application is run; this could be done by inserting a line into the **hp4gl** command file used to run HP ALLBASE/4GL. The file equation can direct the report to any printer connected to the system, specifying any parameters required; the report can also be sent to a file by this method.

It is also possible to allow the end-user to specify the report destination by using the **EXTERNAL** command to call an MPE XL **command file** which sets the file equation according to the user's input.

End-user re-direction of reports on MPE XL is described in the article by Simon Hiscox in **ASO Support Newsletter #18** entitled **REPORTS: End-user re-direction of HP ALLBASE/4GL reports on MPE XL.**

Physical Parameters

The number of physical and logical lines per report can be specified in the header.

The width of the report in number of characters is also declared. HP ALLBASE/4GL uses this number in the report painter to set the maximum width of a line. For screen reports, the width should be limited to 80 characters; HP ALLBASE/4GL will still show reports wider than 80 characters on the screen but a warning message will be displayed and each line will be truncated.

Functions

The developer may declare that a function is to executed before the report begins, and that a function may be executed afterwards as well. For reports which use HP ALLBASE/SQL, the **Before-function** must be used to execute an SQL logic block that will select records from the database for the report to use. An **After-function** may be used to do some special calculations, or perhaps to update a file that uses totalling information from the report.

REP04 - SLIDE: The Report Line Header Screen

REPORTS

Defining a Report

The Report Line Header Screen

The report line header screen defines the characteristics of the lines that will make up the report.

A line group is a logical entity that the report processor prints or processes at a certain stage when running the report.

Thus, by defining what lines we will include in a report, we define the way the report will be processed.

The various line types and groups are:-

Line Type	Group Numbers	Number of Lines	Description
P	1	1 to 99	Top of page headings
C	1	1 to 3	Column headings
B	1	1 to 9	Bottom of page lines
D	1	1 to 99	Detail lines
D	2 to 9	1 to 99	Link detail lines
E	1 to 9	1 to 99	Extra lines
H	1 to 8	1 to 99	Subheading lines
T	1 to 8	1 to 99	Subtotal lines
TF	1	1 to 99	Final total lines

Australian Software Operation
rep04

HEWLETT PACKARD

Student Notes

The Report Line Header Screen

Line Groups

At least one line group must be defined for any report. A line group can have more than one physical line in it, and all those lines will be printed as one logical unit.

Line groups cannot be defined in the **report painter**, so they must be defined before it is called. As with the **screen painter**, fields within a line are created in the **report painter**.

Different line types are used for different purposes and will appear in different parts of the report:

- Type **P** or **Page** type lines appear at the top of every page in the report. There can be 99 lines in this group, numbered **P.01** to **P.99**.

- Type **C** or **Column** type lines are like P type lines; they cannot contain variable references but only literal strings. **Don't bother using them.** There can be 3 lines in this group, numbered **C.01** to **C.03**.

- Type **B** or **Bottom** type lines are like **P** type lines but they appear at the bottom of pages. They are sometimes used to report bottom-of-page totals. There can be 9 lines in this group, numbered **B.01** to **B.03**.

- Type **D** or **Data** type lines are the most commonly used line type. There are 9 lines of type D, namely **D1** to **D9**.

  The **D1** line group is for detail lines; there can be 99 **D1** detail lines, numbered **D1.01** to **D1.99**.

  The D1 line group is printed once for every record selected from the primary file. e.g. consider a file containing 200 records, of which the report selects 75. Therefore, the **D1** line group will be printed 75 times. If the **D1** line group was 4 lines long (**D1.01** to **D1.04**), there would be at least 300 physical lines in the report.

  Lines **D2** to **D9** are for **link details**, lines read from **link files**. Each of these groups allow 99 lines, namely **D2.01** to **D9.99**.

- Type **E** or **Extra** type lines can only be printed from HP ALLBASE/4GL logic using the **PRINT** command. They do not have a preset place in which to appear in a report. Functions can be executed before or after any physical line in a report, and it is usually in one of these functions that you would print an **E** line, using the **PRINT** logic command. There can be 9 groups and 99 lines in each group, namely **E1.01** to **E1.99**.

- Type **H** or **Header** type lines are printed at the start of any control break. There can be 8 **H** line groups, **H1** to **H8**, one for each possible level of control break. An explanation of control breaks follows later. There can also be 99 lines for each group, numbered **H1.01** to **H8.99**.

- Type **T** or **Total** type lines are printed after a control break has occurred. Again, there can be 8 **T** line groups, **T1** to **T8**, one for each possible level of control break. There can be 99 lines for each group, numbered **T1.01** to **T8.99**.

- Type **TF** or **Final Total** type lines are printed at the very end of the whole report. They are usually used to print a grand total for the report. There can be 99 lines, numbered **TF.01** to **TF.99**.

Functions

For any physical print line, the developer may specify a function to be executed before the line is printed, or a function to be executed after the line has been printed.

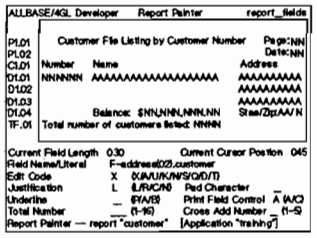
Intentionally blank

REP05 - SLIDE: The Report Painter

REPORTS

Defining a Report

The Report Painter



The report painter allows the developer to define the image of the report by painting the data objects into the report lines.

Report lines may contain the following data types:

- literals,
- application titles,
- master titles,
- constants,
- variables or calculated items,
- screen field references,
- scratch-pad field references,
- communication area fields.

Note that 'C' type lines can only contain literals.

Student Notes

The Report Painter

Report Lines

As was mentioned before, lines in a report must be defined in the **Report Line Header** screen before they can be accessed in the painter.

Fields in report lines can contain any HP ALLBASE/4GL data item; To paint the items, the developer can either use dictionary specifications, or paint the fields manually.

Report Size

The painter provides a **window** of width 78 characters through which a report page, up to 255 characters wide, can be viewed, enabling the developer to create reports that are much larger than the size of the terminal screen. Remember that the width of the report is specified in the **Report Header** screen, and the **report painter** uses that figure to set the **maximum paint width** of the report.

Similarly, reports can be longer than the window provided. The painter provides methods of horizontal and vertical scrolling to allow the developer to position the window at the appropriate position.

Field Details

Once the developer has painted the report image of a field, the details for that field must be entered. Just like screen fields, every report field has a set of **details** which specify the data to be printed in that field, and how the data will appear. Thus, attributes such as field justification (left, right, center) must be specified.

Three different characters can be used to paint screen fields. Alpha-numeric fields are painted using **A** characters but most numeric fields are painted with **N** characters. Numeric fields may also be painted using **Z** characters to specify that the field should remain blank if it is equal to zero.

Note

A numeric report field will only be **blank when zero** if the corresponding dictionary field has the attribute **Blank When Zero AND** it is painted using **Z** characters.

REP06 - SLIDE: Report Sorting

REPORTS

Defining a Report

The Report Sorting Screen

1st	customer_name	Ascending
2nd	state_code	Ascending

The report sorting screen allows the developer to define up to eight sort fields for a report.

When the report runs, HP ALLBASE/4GL internal report sorting will be invoked to sort the data by the specified fields.

Sort fields also define when sub-heading and sub-total linegroups will be printed in the report.

Student Notes

Sorted Data

Product Number	Option Number	Cost
XY1000	0001	1500.00
XY1000	0002	1200.00
XY1000	0003	800.00
XY1001	0001	2500.00
XY1001	0002	1800.00
XY1001	0003	1300.00
XY1002	0001	500.00
XY1002	0002	300.00
XY1002	0003	200.00

Report Sorting

After reading the slide, refer the students to the screen image of the **Report Sorting** screen at the end of this module.

Sorting

The eight fields on which the report can be sorted do not have to be index fields; they only have to be valid fields from the record layout.

HP ALLBASE/4GL can sort in ascending or descending order on any one of the fields. If the file is already in order (the report is in the same order as the order of the primary key), the developer can specify that the report does not need to be sorted, speeding up the report considerably.

SQL Sorting

If the report is operating on an HP ALLBASE/SQL database, sorting should be done by an SQL select statement using the **ORDER BY** clause in the **Before Report Function**.

The **Report Sorting** screen should specify that the file is already in order, as no re-sorting is required. It may still be useful to specify some fields on this screen; this will inform HP ALLBASE/4GL which fields were used for sorting, so that control breaks will occur in the correct places in the report.

Control Breaks

A control break occurs whenever the value of a sorting field changes. The level of the control break depends on the level of the sorting field which changed; the control break level in turn determines which totalling line group T1 to T8 is printed.

Sorted Data

Product Number	Option Number	Cost
XY1000	0001	1500.00
XY1000	0002	1200.00
XY1000	0003	800.00
XY1001	0001	2500.00
XY1001	0002	1800.00
XY1001	0003	1300.00
XY1002	0001	500.00
XY1002	0002	300.00
XY1002	0003	200.00

This table has three sort fields; **Product Number** is the first, **Option Number** is the second level and **Cost** is the third level.

A control break occurs each time **Cost** changes value which is also when **Option Number** changes value; these level 2 and level 3 control breaks can be ignored.

The level 1 control break occurs whenever **Product Number** changes; this could be used to produce a sub-total for the cost of all options with that product number.

Newsletter Article

Refer the students to the newsletter article **REPORTS: Reporting Control Breaks in HP ALLBASE/4GL** by Philip Anderson. The article is in **ASO Support Newsletter #11.**

Intentionally blank

REP07 - SLIDE: Report Selection

REPORTS

Defining a Report

The Report Selection Screen

	Field Spec. Name
	zip_code
Selection Criteria: Values FROM	"00000"
TO	V-max_zip_code

The report selection screen allows the developer to define one or more selection criteria, thereby specifying a subset of the data being provided by the database.

Student Notes

Report Selection

After reading the slide, refer the students to the screen image of the **Report Selection** screen at the end of this module.

Selection Criteria

After each record is read from the primary file, **all** of the defined selection criteria are applied to that record.

If the report is not using an HP ALLBASE/SQL database, selection criteria can be specified to determine which records from the primary file will be included in the report.

Selection criteria can be specified in several ways:

Ranges

Many **FROM** and **TO** ranges can be entered to select records. The values in the ranges can be literals, or any HP ALLBASE/4GL data items.

Tables

Instead of using ranges, validation tables can be specified; only those records that match values from the tables will be printed.

Function

An **After-selection** function may be specified that will be executed once after every record is printed. By manipulating communication area fields, the after selection function can specify that a record should not be included in the report, or allows a record that has already been rejected by the other criteria to actually be included.

Efficiencies

HP ALLBASE/4GL's selection mechanism requires every record in the primary file to be read. It is often far more efficient to employ a small amount of procedural logic in the **Before Report Function** to limit the number of records which HP ALLBASE/4GL must process. This will produce significant performance improvement if the report is from a large database and only a small number of records are required.

Refer the students to the article on this topic by Keith Glennan in **ASO Support Newsletter #1**, entitled **Record selection with HP TODAY reports.**

If the report uses an HP ALLBASE/SQL database, use the selection facilities provided by HP ALLBASE/SQL rather than those from HP ALLBASE/4GL. The SQL Logic Block which defines the cursor into the database should include a **WHERE** clause in the **SELECT** statement.

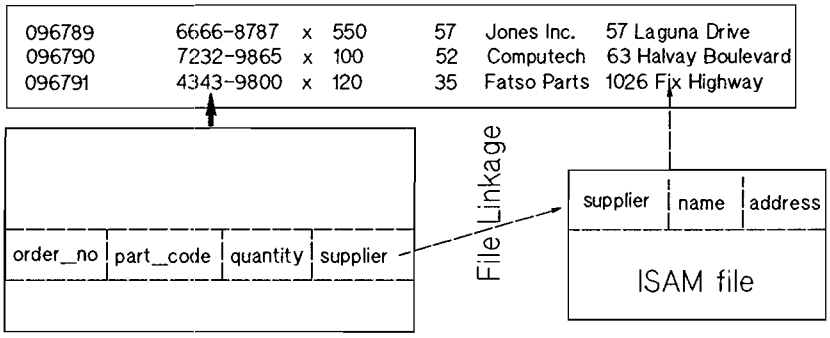
Intentionally blank

REP08 - SLIDE: File Linkages

REPORTS

Defining a Report

– The file linkages screen.



File linkages cause the report processor to automatically extract data from a secondary file based on values in the primary file.

This provides a convenient method of merging data from many files into one report.

Student Notes

File Linkages

ISAM/KSAM and Serial Files, HP TurboIMAGE data sets

To print out information on more than one file, it is necessary to **link** those files, called **link files**, into the report. HP ALLBASE/4GL provides an automatic file linkage facility to do this. The **link file** can only be an ISAM/KSAM file or an HP TurboIMAGE data set.

The **link file** must contain a field that also exists in the primary file. If the value in the primary file field is the same as that in the corresponding field in the link file, the secondary file record will be linked in.

The report can have a special line group printed as a result of the link, or the linked information can be included into existing report lines.

HP ALLBASE/SQL

File linkages should not be used with reports which access multiple tables in an HP ALLBASE/SQL databases. The HP ALLBASE/SQL approach is to use the concept of a **join**, which will combine all the tables into one large **virtual table.** One way to do this is to create a **view** within the HP ALLBASE/SQL database. However, HP ALLBASE/4GL makes the task very simple to implement by means of SQL Select Lists. The select list is used as the primary file for the report.

Newsletter Article

Refer the students to the newsletter article in **ASO Support Newsletter #2**, entitled **PHIL'S TIPS: Reports and Multiple Record Layouts** by Philip Anderson.

REP09 - SLIDE: Totalling Communication Areas

REPORTS

Totalling Communication Areas

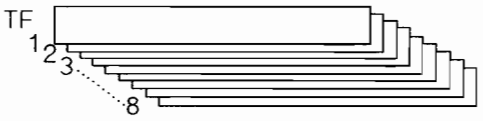
*COUNT(1 - 5)

can be used to count the number of times a linegroup has been printed.

*CROSS(1 - 5)

can be used to total values across the page.

*TOTALS(1 - 16)



The 16 *TOTALS communication areas accumulate totals during report generation.

A set of 16 *TOTALS fields is provided for each sort level defined for the report. The set which your report will reference at any given point in the report is determined by the current sort level.

Student Notes

Totalling Communication Areas

*CROSS

The ***CROSS** communication fields are used to accumulate across-the-page totals. There are 5 fields, ***CROSS(1)** to ***CROSS(5)**, and they can be used on any report line. The ***CROSS** specification must be entered in the report field details section of the report painter.

*COUNT

The ***COUNT** fields are used to count the number of times that a particular line group gets printed. There are 5 *COUNT fields, ***COUNT(1)** to ***COUNT(5)**. The count fields allow the report to simultaneously keep track of the number of times that five separate line groups have been printed.

*TOTALS

The ***TOTALS** communication fields are used to accumulate totals for columns. There are 16 ***TOTALS** fields, ***TOTALS(1)** to ***TOTALS(16)**; a report can accumulate totals for 16 separate columns simultaneously.

In reality, there are more than 16 totalling fields, as separate **running totals** must be kept for each of the 8 sorting levels. This enables the report to output the current state of the accumulation at any control break level, in a **T** type line group at any point in the report printing. The **Reporting control breaks** example illustrates how two levels of sorting work, the further levels just being an extension of the same ideas.

Question

How could a report output the number of records selected from the primary file? Recall that the **D1** line group will be printed once for every selected record in the primary file.

Answer:

It could count all the **D1** lines that are printed using one of the ***COUNT** communication areas.

Questions

Some of the questions asked in this exercise have not been covered in the course. The students should refer to the **Developer Reference Manual**, the **Reports** section to find the answer if they are unsure. This will be good practice at using the HP ALLBASE/4GL manuals.

1. Where is the size of a report defined?

   Answer:

   In the **Report Header** screen.

2. When in a report will the **P1** line group be printed?

   Answer:

   P1 line groups are printed as **top-of-page** headings.

3. When in a report will the **D1** line group be printed?

   Answer:

   D1 line groups are detail lines.

4. What is the **TF** line group used for?

   Answer:

   TF line groups are used for final totals.

5. How would you specify that three lines should be skipped after every column heading line is printed?

   Answer:

   By specifying the number **3** in the **After Print** field for a **C1** type line.

6. What is the communication area field that can be printed to show the current page in a report?

   Answer:

   The ***PAGENO** communication area field.

7. How could you specify that a numeric data field should have surrounding parentheses if its value is negative?

   Answer:

   Surround the field by () when painting it using the **Report Painter.**

8. What three report components should be completed before you attempt to generate the report?

   Answer:

   The **Report Header** screen, the **Report Line Header** screen and the **Report Painter.**

9. A report may be invoked by the **REPORT** logic command, or as an action item on a menu or function key. If I have not yet defined a way of invoking a report in my application, how can I run a quick test?

   Answer:

   You could use **menu bypass** as specify **R-** as the action.

10. Do you think this technique would work for other application components?

    Answer:

    Yes, it works for processes **P-**, functions **F-** and screens **D-**.

Application Development

The students will now develop the reports for the **people tracking** and **resource tracking** (book tracking) sections of their application.

As both sections are based on HP ALLBASE/SQL, use a **Before Report Function** which will contain a call to an SQL Logic Block to declare a cursor for the table. Remember that the report processor automatically supplies a **FILE *NEXT** to retrieve the data.

Use the HP ALLBASE/SQL **SELECT** command to perform selection (if any) and sorting.

Person Tracking

The reports for the **people tracking** module should use the SQL table **person** as the primary file.

You should produce two reports. The first should list the names of all people in the database. The second should list the names sorted by **state** and then by **zip code.**

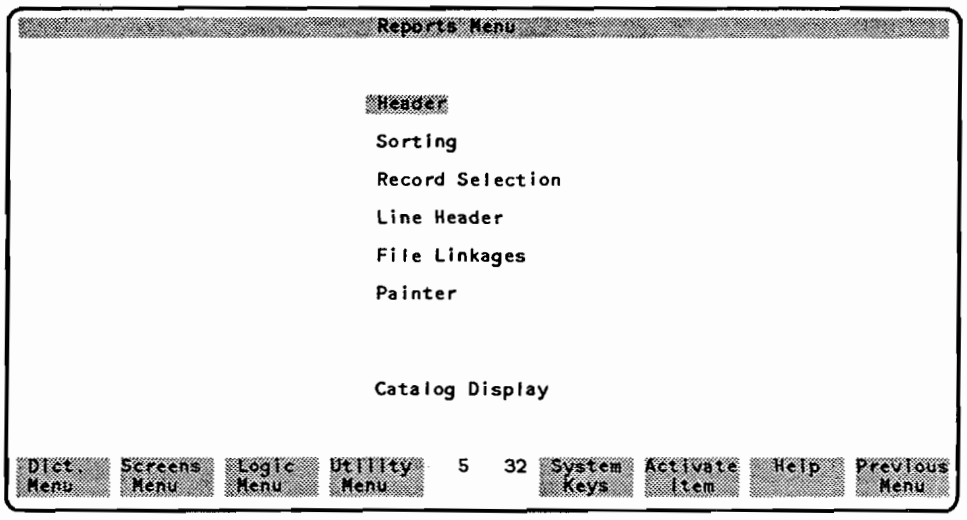
Book Tracking

The reports for the **book tracking** module should use the SQL table **books** as the primary file.

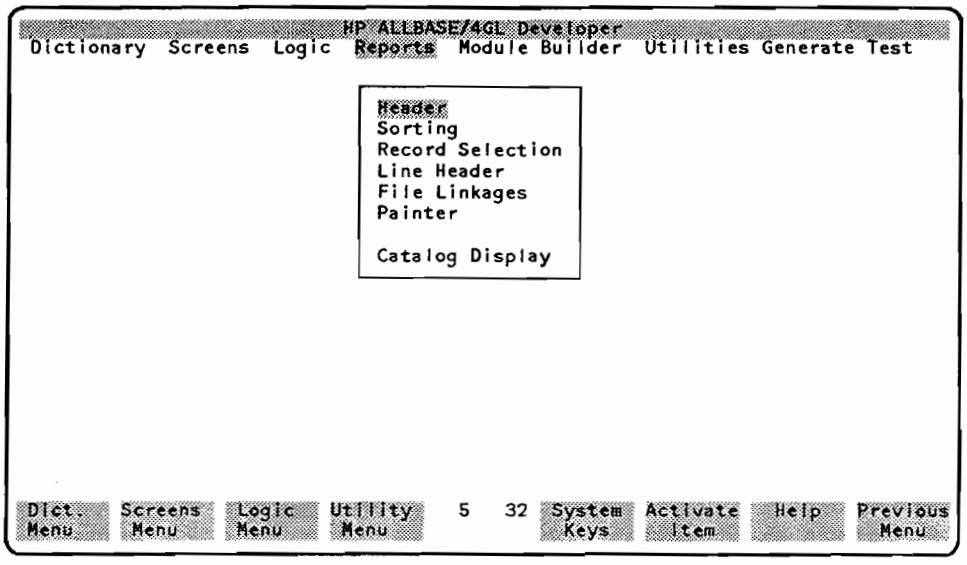
If time permits, you should produce two reports. The first should list the names of all books in the database. The second should list the names sorted by **author** and then by **category**; the report should also indicate the total number of books and the total cost of all stock.

Screen Images

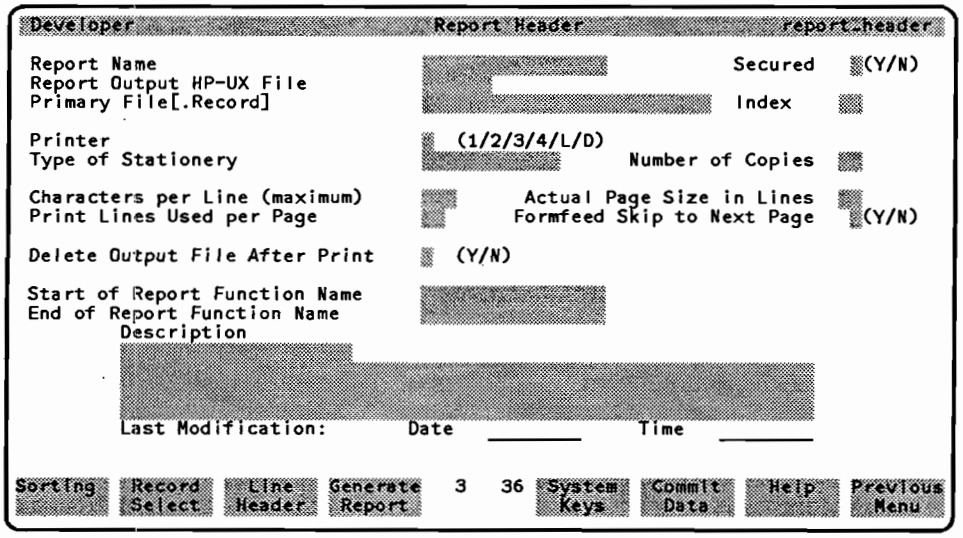
The **Reports Menu screen for HP-UX is shown below:**



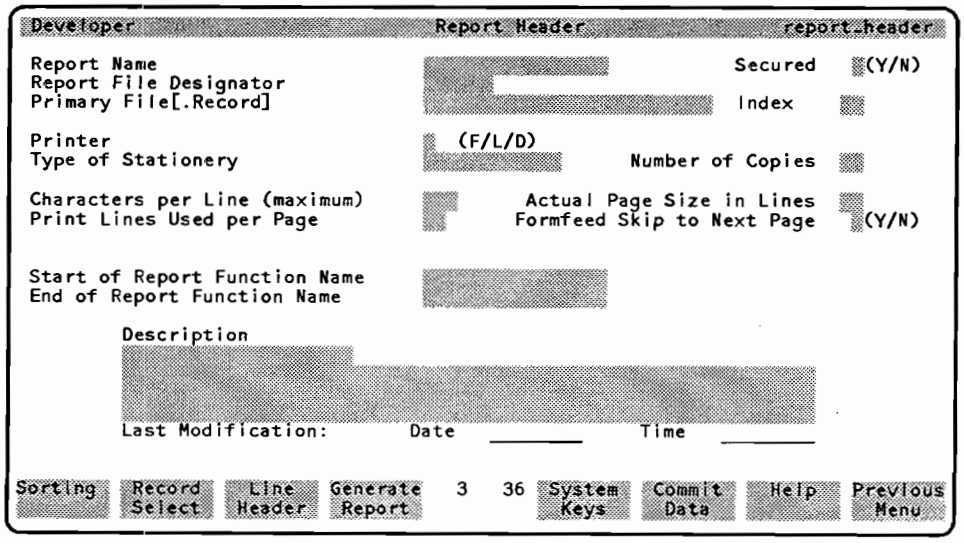
The Reports Menu screen for MPE XL is shown below:



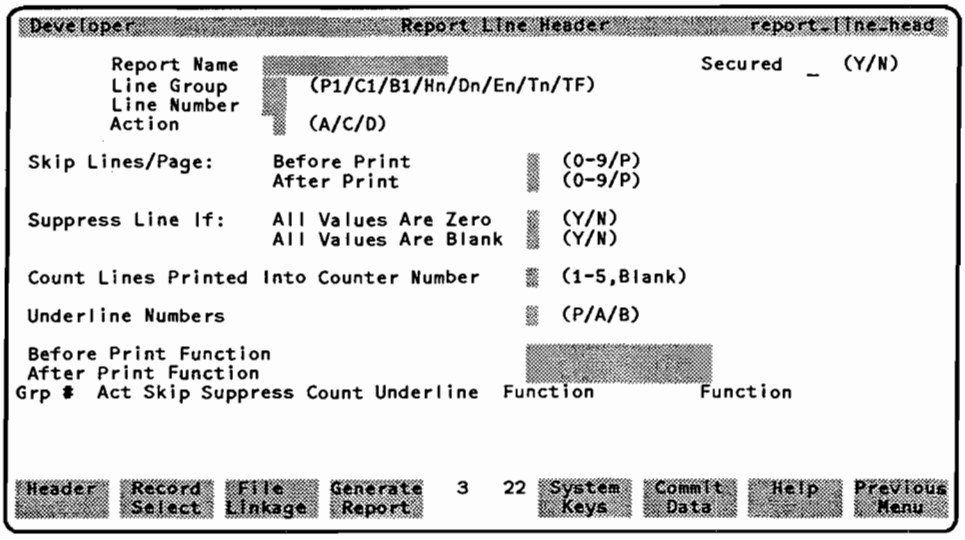
The Report Header screen for HP-UX is shown below:



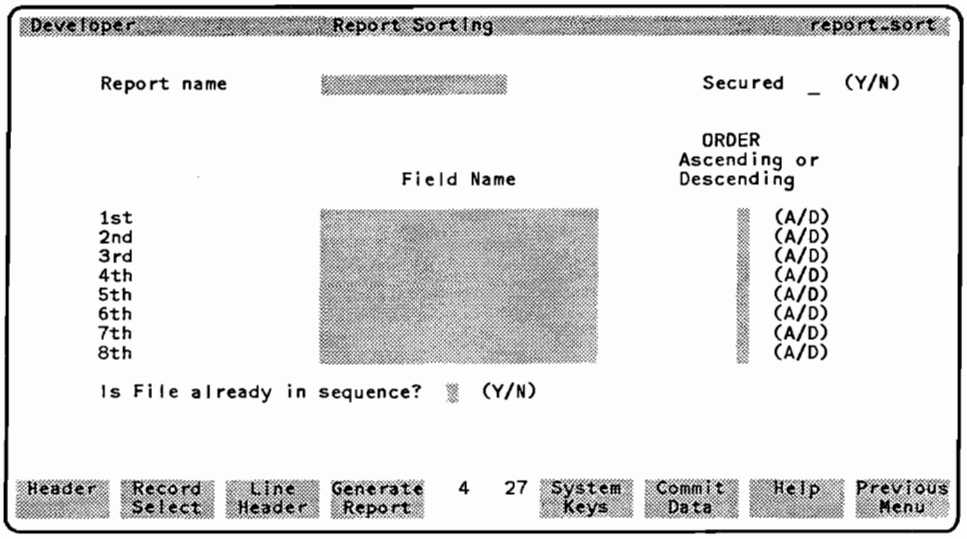
The Report Header screen for MPE XL is shown below:



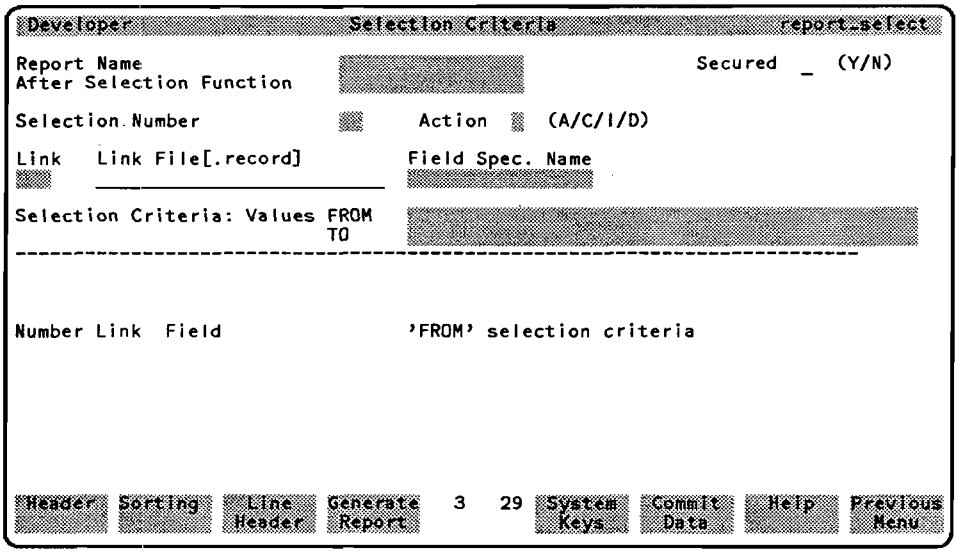
The Report Line Header screen is shown below:



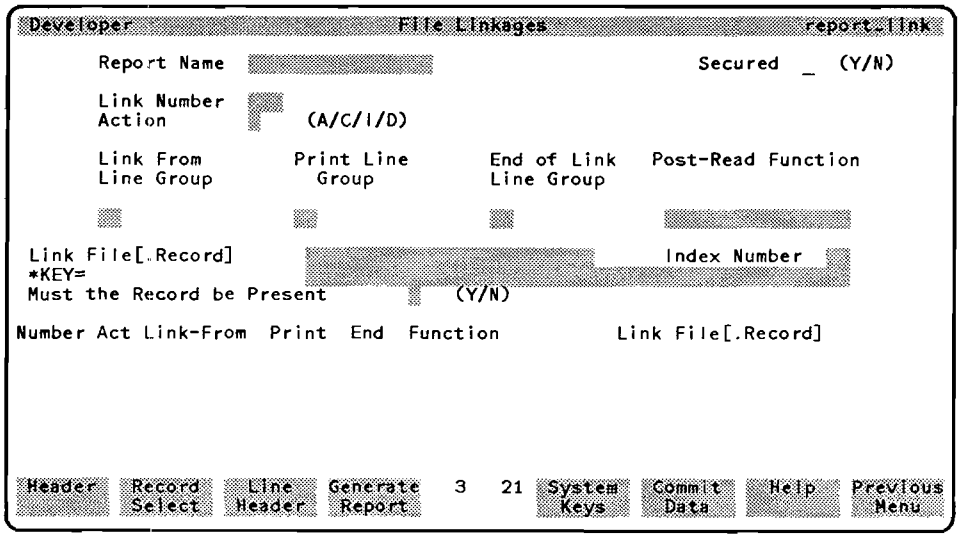
The Report Sorting screen is shown below:



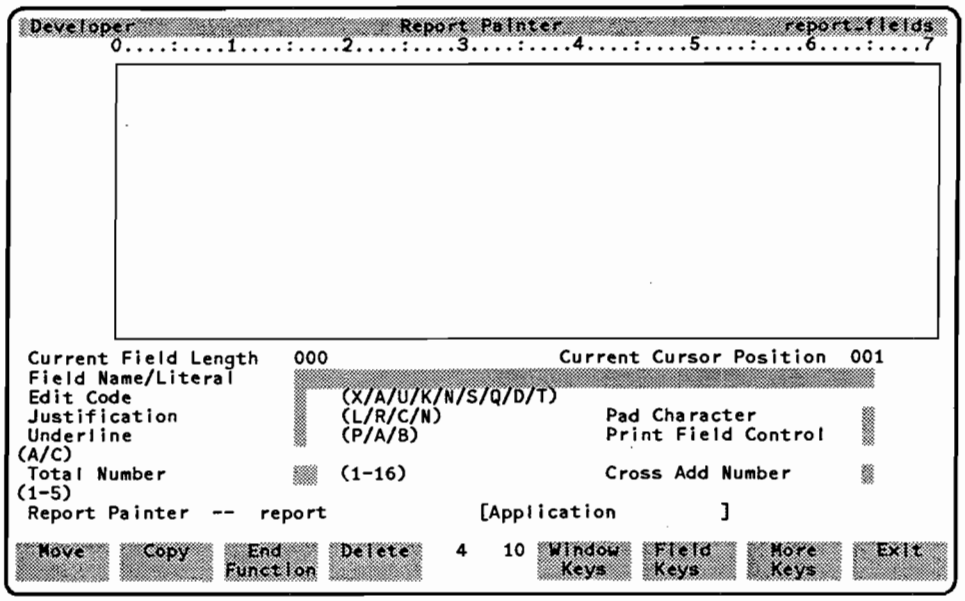
The Report Selection screen is shown below:



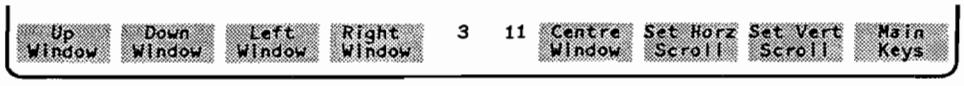
The Report File Linkages screen is shown below:



The Report Painter screen is shown below:



The Window function key set for the Report Painter screen is shown below:



Module: HP ALLBASE/4GL Adminstration

Instructor Notes

ADMIN00 - SLIDE: Module Objectives

HP ALLBASE/4GL ADMINISTRATION

HP ALLBASE/4GL User Definition
Application and Version Administration
System Wide Definitions
System Security

OBJECTIVES:

Upon completion of this module, the student will be able to:

1. Describe the procedure for defining HP ALLBASE/4GL endusers and developers. Describe Training mode.
2. Explain applications and versions, how to define them, and how to load and and unload them.
3. List which aspects of the HP ALLBASE/4GL environment are configurable on a system wide basis, and how this is done.
4. Describe the different security features provided by HP ALLBASE/4GL.

Student Notes

Module Objectives

This module will introduce the students to the **administrator** application.

Student Objectives

At the end of this module, each student will be able to:

- Describe the procedure for defining HP ALLBASE/4GL endusers and developers. Describe Training mode.
- Explain applications and versions, how to define them, and how to load and unload them.
- List which aspects of the HP ALLBASE/4GL environment are configurable on a system-wide basis, and how this is done.
- Describe the different security features provided by HP ALLBASE/4GL.

Resource Allocation

Equipment

The following items are required for this module:

- Instructor notes
- Student notes
- Slides
- A laboratory worksheet
- Access to an HP 3000 Series 900 machine or an HP 9000 Series 800 or HP 9000 Series 300 machine with HP ALLBASE/4GL installed.
- The complete set of HP ALLBASE/4GL manuals

Time

This module should take approximately 2 hours, with 1 hour for theory and 1 hour for the 4 laboratory sessions. Each laboratory session should take approximately 15 minutes.

Pacing

Read through the slides reasonably quickly.

ADMIN01 - SLIDE: Overview

HP ALLBASE/4GL ADMINISTRATION

Overview

The system administrator uses the "administ" application to control the overall operation of the HP ALLBASE/4GL system. This includes:

* definition of system users
* definition of applications and versions
* loading, unloading and deleting applications
* control of system wide parameters
* definition and control of system security

HP ALLBASE/4GL provides all these facilities in one easy to use application.

Australian Software Operation
admin01

HEWLETT PACKARD

Student Notes

Overview

The **administrator application** was designed to manage all the applications in one set of HP ALLBASE/4GL **Special files (S-files)** .

Thus, the HP ALLBASE/4GL **administrator** is able to create and delete applications and user names, unload and load applications, set **system-wide** or **global** parameters, and produce documentation.

System-wide in this case refers to parameters which will affect all of the applications. HP ALLBASE/4GL also permits **local items** which are specific to one application.

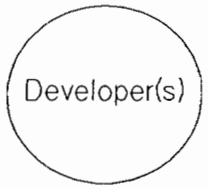
ADMIN02 - SLIDE: Developer and User Definition

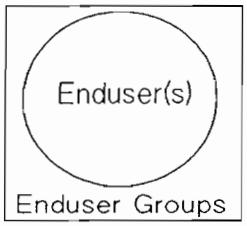
HP ALLBASE/4GL ADMINISTRATION

HP ALLBASE/4GL User Definition

The HP ALLBASE/4GL Administrator is responsible for defining new endusers and developers.

The users who develop new or existing applications. May be many Developers.





The users of developed applications. May be many endusers. Endusers may be grouped.

* Each name must be distinct and used for an enduser or developer.
* Endusers may be placed in groups so that a particular group can be made eligible to use an application.
* Password security may be added for endusers, developers and the administrator.

Australian Software Operation
admin02

HEWLETT PACKARD

Student Notes

Developer and User Definition

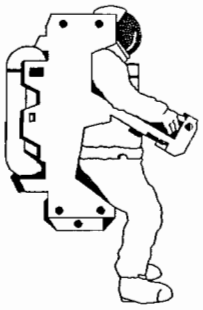
This topic was well covered in the module **Introduction to HP ALLBASE/4GL**. Just read through the slide quickly.

ADMIN03 - SLIDE: User Definition - Training Mode

HP ALLBASE/4GL ADMINISTRATION

HP ALLBASE/4GL User Definition

Training Mode



To help new endusers get started with an application, HP ALLBASE/4GL provides the optional Training Mode.

When in Training Mode, endusers have full access to the application they are running, but any changes they make to data files never get written to disk.

* When working with ISAM/KSAM/serial and HP TurboIMAGE datasets, files insert/update/delete operations are ignored.
* When working with SQL, all 'commit work' SQL transactions are replaced by 'rollback work'.

Australian Software Operation
admin03

HEWLETT PACKARD

Student Notes

User Definition - Training Mode

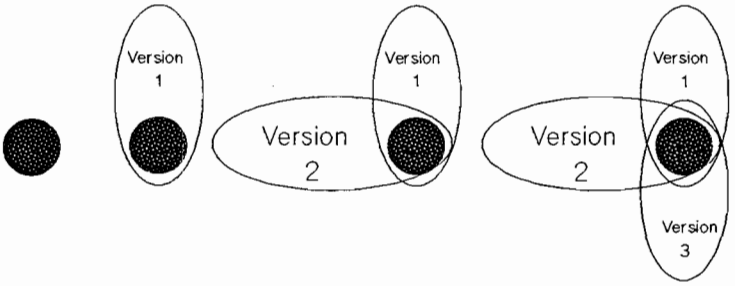
When in **training mode**, a user cannot update information in the database. It will appear that write operations are occurring, but in fact the information will never be written to the database. This is so that new users can operate an application without the possibility of affecting important data.

The HP ALLBASE/4GL administrator can define that a user will operate in **training mode** in the user definition screen in the administrator application. **Training mode** can also be initiated manually by the end-user by using the `Training Mode` key on the **System Keys** function key set.

ADMIN04 - SLIDE: Application and Version Administration I

HP ALLBASE/4GL ADMINISTRATION

Application and Version Administration



* A base application is an application that is complete in itself.
* A version is a set of changes or extensions to a base application.
* Versions are perfect for producing customizations of a standard application for use by a particular customer or enduser.
* A version is not complete within itself.

Student Notes

Application and Version Administration I

Base Applications

A **base application** is complete in its own right; all items it references are contained within it. It has a complete **Application Definition** screen, possibly specifying an HP ALLBASE/SQL database; thus, it has a complete **system master record** contained in file **s03** in the HP ALLBASE/4GL S-files.

Versions

A version must be associated with a base application, and the version will only contain the parts that **differ** from the base application. For all parts of the version that are the same as the base application, the base application definitions will be used. If an item in the base application changes, those changes will also be reflected in the version (assuming that the version does not already have a changed copy of that item).

Versions are not complete in themselves, as they only contain changed items.

Note

The last statement needs qualification. Some HP ALLBASE/4GL items are controlled by a **namelist**; variables are an example.

With such items, developers must be aware that making a change in the version for that item type will prevent future changes in the base for that item type from appearing in the version. For example, making a change to a variable in a version causes HP ALLBASE/4GL to copy the complete set of variables from the base application into the version. Subsequently, changes to any variables in the base will not be reflected in the version as the version now has its own copy.

This is fully documented in the HP ALLBASE/4GL Administrator Reference Manual.

The work-around is to use the utilities **hp4stoa** and **hp4atos** to merge the base and versions into a new base; from that new base, create a new version.

ADMIN05 - SLIDE: Application and Version Administration II

HP ALLBASE/4GL ADMINISTRATION

Application and Version Administration

The Administrator defines new applications in the Application Definition screen. For each application they must define:

* application name
* password for the application endusers (optional)
* security code for application developer (optional)
* list of valid users (may be *ALL - everybody)
* for SQL applications, a database environment must be specified.
* for SQL applications, an effective SQL user or group may be specified.

For a version, similar entries are required; the application on which the version is based on must also be specified.

Australian Software Operation
admin05

HEWLETT PACKARD

Student Notes

Application and Version Administration II

The students should be familiar with the **Application Definition** screen by now. Just read through the slide for revision.

Developer Security Code & Password

Versions can have their own **Development Security Code** and **Password**, separate from the base application.

Valid Users

Versions can have their own list of valid users and groups, separate from the base application.

SQL

Versions are completely dependent on the base application for the name of the HP ALLBASE/SQL database and the owner group. The reason is that in many places when referencing items in a database, it is necessary to mention the **database name** and the **owner group**; allowing a different database and **owner group** would cause great difficulties with internal management of the version.

Refer the students to the **Version Definition** screen as shown in the **Screen Image** page at the end of this module.

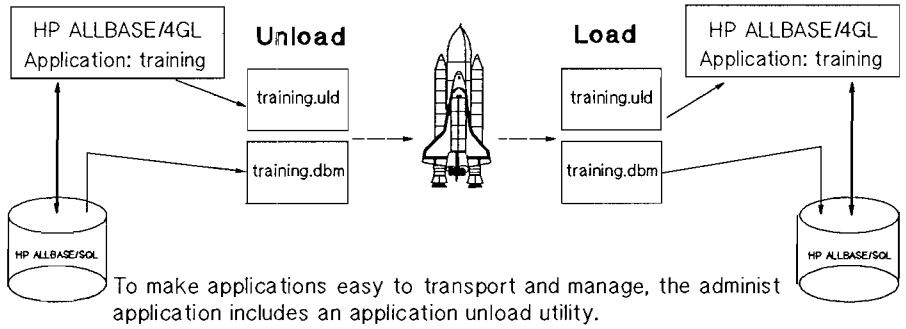
Laboratory Exercise

The students should now do the laboratory exercise entitled **Versions**.

ADMIN06 - SLIDE: Unloading and Loading Applications I

HP ALLBASE/4GL ADMINISTRATION

Application and Version Administration



"HP-UX to MPE XL departing in 15 minutes, gate u.l.d."

To make applications easy to transport and manage, the administ application includes an application unload utility.

The unload utility takes all the components of an application out of the dictionary and stores the complete application in a single flat file. This file is called a 'uld' file, because of its filename extension.

If the application is an SQL application, unload will connect to the database environment and unload all the compiled SQL components into a 'dbm' file.

A load utility is provided for loading the application at the target location.

Student Notes

Unloading and Loading Applications I

Application Unloading

The **Unload Application** screen allows the HP ALLBASE/4GL administrator to unload an application from a developer environment. The runtime environment cannot unload applications.

On HP-UX, the **unload file** is stored in the **HP4SPATH** directory with the same name as the application but with the **.uld** extension. For example, if the application **payments** was unloaded, the **application definition file** would be **payments.uld**

On MPE XL, the **unload file** is stored in the **HP4APPNPATH** group with the same name as the application. In the standard HP ALLBASE/4GL environment, the group is **HP4APPN**. The **application definition file** for the application **payments** would be **PAYMENTS.HP4APPN** in the current account.

Application Loading

The **Load an Application** screen allows the HP ALLBASE/4GL administrator to load an application from an **unload file.** The HP ALLBASE/4GL runtime environment can load applications.

HP ALLBASE/4GL expects to find the file in the **HP4SPATH** directory on HP-UX or the **HP4APPNPATH** group on MPE XL.

The **Load an Application** screen can rename an application during the process of loading; the administrator must specify the old name of the application so that HP ALLBASE/4GL can locate the **unload file** and then specify the new name for the application.

WARNING

Do not try to rename an application by using an operating system utility such as **mv** or **RENAME** as the name of the application is also stored within the file. If HP ALLBASE/4GL finds that the external name and the internal name do not match, an error message will be given and the load will not proceed.

Note that it is possible to determine the internal name by viewing the file with a pager program such as **more** or **PRINT**. Most of the information will be nonsense as the file is binary, but the name of the application should be discernible.

On HP-UX, HP ALLBASE/4GL can also load **unload files** with the extension **.tdy**; these were produced by the older version of HP ALLBASE/4GL called **HP TODAY**. They are identical to **.uld** files; in fact, it is possible to rename a **.tdy** file to give it the extension **.uld** without affecting the loading operation.

Application Transfer

It is possible to unload an HP ALLBASE/4GL application from any system on which HP ALLBASE/4GL is available, and reload it on any other. This means that it is easy to move applications between machines very easily, whether the transfer is between HP-UX machines, MPE XL machines or between HP-UX to MPE-XL. The application can be run with no modification necessary; it is not even necessary to re-generate.

Stand-alone use of HP4LD and HP4ULD

Load and **Unload** can be executed from the Administrator, or they can be run stand-**alone**. There are advantages in using the utilites in this fashion; these include the ability to:

- Unload or load more than one application at a time; multiple applications can be stored in one **.uld** file.
- Circumvent passwords in case somebody has forgotten an application password.
- Unload or load from HP ALLBASE/4GL S-files which have a different version number to the HP ALLBASE/4GL program files.

The Database Module File

A **Database Module** file is created only if the application uses HP ALLBASE/SQL. This file then contains the executable form of all the SQL items that the application needs; thus it contains the generated sections from all **SQL Logic Blocks** and the **SQL Table** definitions.

On HP-UX, the **Database Module file** is stored in the **HP4SPATH** directory with the same name as the application but with the **.dbm** extension. The application **payments** would produce the file **payments.dbm**

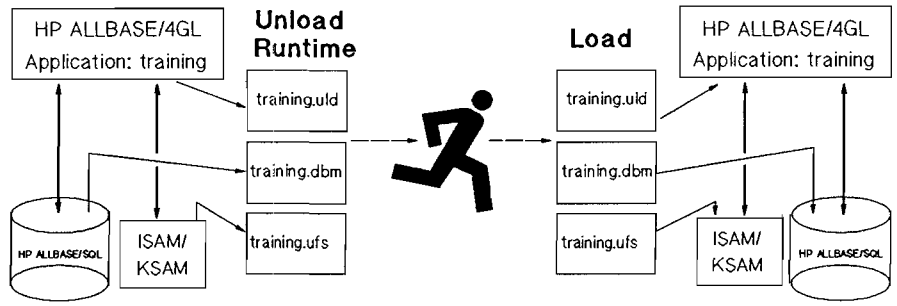
On MPE XL, the **Database Module file** is stored in the **HP4DBMPATH** group with the same name as the application. The group is called **HP4DBM**, so the application **payments** would produce the file **PAYMENTS.HP4DBM** in the current account.

When moving from developer system to developer system this file is not really needed, as all the SQL objects can be re-generated from their source on the destination machine. However when moving an application to a run-time environment, the **Database Module** file is essential as there is no way to re-generate the items.

ADMIN07 - SLIDE: Unloading and Loading Applications II

HP ALLBASE/4GL ADMINISTRATION

Application and Version Administration



"Hey Joe, I've got the latest update of the application"

If the target site is a runtime site, a runtime only application can be unloaded. This form of the application contains no source and is therefore more secure and compact.

When unloading from HP ALLBASE/4GL, an File Structure File can be unloaded. This file contains the definition of all ISAM/KSAM files in the application. On HP-UX, the file has a '.ufs' file extension.

Student Notes

Unloading and Loading Applications II

Run-only Applications

When moving an application, the developer can specify that the source of the application should not be included in the **application definition file**. This is done by setting the **Unload Run-Only Copy** field to **Y**. This option means that the **unload file** will be smaller and provides security in that it is impossible for somebody to modify the application.

This option should be used when the destination system is a **runtime environment**. It can, however, be used to produce a **secure** version of the application in a **developer environment**. Thus, it is possible to **emulate** a **runtime system** on a **developer system**.

Note

No harm is done by loading an **unload file** containing the application source code into a runtime environment. The application will behave the same as in a developer environment.

Automatic Run-time Data File Reformat - The File Structure File

HP ALLBASE/4GL provides the facility to automatically re-format ISAM/KSAM files during the installation process for run-time systems.

When the administrator specifies that a runtime application is to be unloaded, HP ALLBASE/4GL creates a **File Structure** file which contains the structure of all of the ISAM/KSAM files used by the application.

The File Structure File

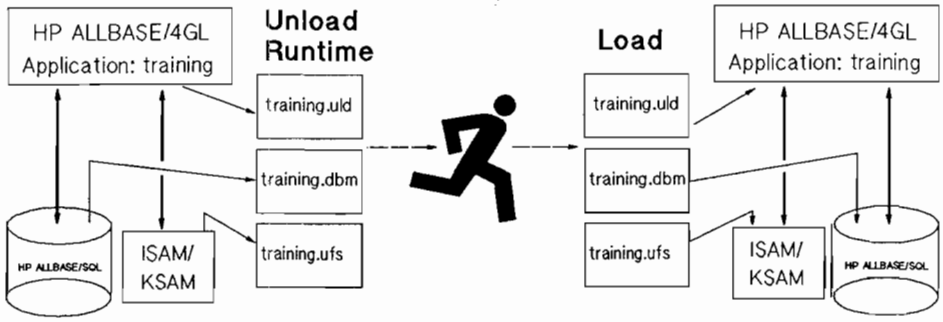
On HP-UX, the **File Structure** file is created in the **HP4SPATH** directory with the same name as the application and the extension **.ufs**. The application **payments** produces the file **payments.ufs**

On MPE XL, the **File Structure** file is created in the **HP4FSPATH** group with the same name as the application. The default name for the group is **HP4FS**, so the **payments** application produces the file **PAYMENTS.HP4FS** in the current account.

ADMIN08 - SLIDE: Unloading and Loading Applications III

HP ALLBASE/4GL ADMINISTRATION

Application and Version Administration



For runtime systems, the load utility allows automatic reformat of the application ISAM/KSAM data files, even though the application has no source.

On HP-UX, the contents of the '.ufs' file is compared with the existing '.str' file for each ISAM file, and reformatting is carried out as needed.

Automatic data file reformat is not provided for SQL tables. The ISQL UNLOAD and RELOAD commands must be used for this function.

Student Notes

Unloading and Loading Applications III

Automatic Run-time Data File Reformat - The Loading Process

For runtime systems at the time of loading an application, if a **File Structure** file is present, the load program will compare the record layouts in the **File Structure** file with the file's current physical structure. If they differ, the load program will reformat the file so that they match the new file definitions.

The **File Structure File** must be copied to the runtime system along with the **Application Definition File**. When HP ALLBASE/4GL loads the application, the loading program **HP4LD** will check for the presence of the **File Structure File** in the **HP4SPATH** directory or **HP4FSPATH** group. If the file exists, **HP4LD** will call the **HP4FUTL** program to compare the record layouts contained in the **File Structure File** for each file against those of the existing files in the runtime system. For the data files whose structures differ, **HP4LD** will automatically reformat the data file.

SQL Reformat

Automatic reformat is not provided for HP ALLBASE/SQL tables, however there are several ways to achieve the same result. All methods require the information to be written out of the table, and then be written into a table with the new format.

The HP ALLBASE/SQL tool **sqlgen**, available on both HP-UX and MPE XL, can be used in this process as it can write the table structure and data out to a flat files. It can also produce **unload** and **load** scripts suitable for processing by **isql**.

After using **sqlgen**, run **isql** to process the **unload** script to extract the data from the table, delete the table, create the new table definition and then process the **load** script to load the data back in.

An alternative method, using the power of HP ALLBASE/4GL, makes it very easy. First create a table with the new format, then write a process to read the data from one table and insert it into the other. Of course, this method cannot be used in a runtime system.

Laboratory Exercise

The students should now do the laboratory exercise entitled **Unloading and Loading**.

ADMIN09 - SLIDE: System Wide Specifications - Definitions

HP ALLBASE/4GL ADMINISTRATION

System Wide Specifications

System Wide Definitions

The System Wide Definitions are split into two functional areas:-

System Wide values

- date format
- decimal radix character
- currency symbol

Environment parameters

- shell program or command interpreter
- printers
- cartridge device
- magnetic tape device

} only on HP-UX

Australian Software Operation
admin09

HEWLETT PACKARD

Student Notes

System Wide Specifications - Definitions

The **System Definition** screen allows the administrator to set a number of parameters for the entire HP ALLBASE/4GL system. The screen displays the format/setting for each parameter.

System Wide Values

Date Format

The date can be in either the European format **DD/MM/YY** or the American format **MM/DD/YY**. The only difference is that the day and month are reversed. The date format only applies to the **external** date format; **internally**, the date is always stored as **YY/MM/DD**. Thus an application that has been developed using American date format can be used in a system with European date format.

The administrator can also specify the **Date Separator** character; by default it is / but it can be changed to anything. A setting of - would set the default date format to be MM-DD-YY.

Decimal Radix Character

The **Decimal Radix Character** is the separator between units and decimal fractions, but it also determines the **thousands separator**. It may only be a comma , or a period . character.

If the **Decimal Radix Character** is a period, the **thousands separator** is a comma and vice versa. The allowable numeric formats are NN,NNN.NN and NN.NNN,NN.

Currency

The administrator can define the currency symbol character; it may be a dollar sign or a pound sign for example.

Environment Parameters

Shell Program

The administrator may specify which **shell program** is to be called when the user presses the `Op. System` function key. By default the shell is **/bin/sh** for HP-UX and **CI.PUB.SYS** for MPE XL.

To deny users access to the operating system, it is possible to name a script here which returns immediately, such as **/bin/true**. Alternatively, it is possible to write a script which simply echoes **You cannot access the operating system** before it exits and returns control to HP ALLBASE/4GL.

Note

On HP-UX, the setting in this field will only take effect if the **SHELL** environment variable is not set. If you wish to truly disable access to the operating system, you must ensure that **SHELL** is unset, possibly in the HP ALLBASE/4GL start-up script.

The following parameters are only used on HP-UX. They are not required on MPE-XL because the operating system supplies this functionality itself.

Printers

The administrator can define four system printers, known in HP ALLBASE/4GL as printer #1, #2, #3, #4. Printer #1 is known as the **system printer. Developer** and **administrator** reports, as well as screen prints, will be sent to the **system printer.**

They can all be the same printer; alternatively, #1 may be a line printer and #2 may be a laser printer, for example.

Cartridge Device

This specifies the cartridge tape device file that will be used for HP ALLBASE/4GL backups; the default is **/dev/rct/rct.** The device file name determines all parameters such as tape density and number of tracks. HP ALLBASE/4GL checks that any entry in this field is a valid device file.

Magnetic Tape Device

This specifies the magnetic tape device file or the flexible disk device file to be used for HP ALLBASE/4GL backups. The default is **/dev/rmt/0m** for magnetic tape devices. For flexible disks, the device file will be **/dev/rfd** or **/dev/rfda**. Again, HP ALLBASE/4GL checks the entry.

Enter **/dev/null** if no tape drive or flexible disk drive is connected.

ADMIN10 - SLIDE: System Wide Specifications - Site Synonyms I

HP ALLBASE/4GL ADMINISTRATION

System Wide Specifications

Site Synonyms

GOTO = ENTER ***NOW = *DATE**

HP ALLBASE/4GL allows the definition of synonyms for the names of all:

* Logic Commands
* Communication Area Fields

Australian Software Operation
admin10

HEWLETT PACKARD

Student Notes

System Wide Specifications - Site Synonyms I

Site Synonyms

Developers may rename HP ALLBASE/4GL logic commands to suit their tastes; for example a site may wish to call the **ENTER** command **GOTO**.

Communication Areas

Communication areas can also be renamed. For example, ***SUITE** could be renamed to ***APPLN**

Note

An application which relies on synonyms for logic commands and communication areas will not be able to run in another system unless the same synonyms are established. This may be workable in a developer environment but in a runtime environment it presents real problems.

ADMIN11 - SLIDE: System Wide Specifications - Site Synonyms II

HP ALLBASE/4GL ADMINISTRATION

System Wide Specifications

System Wide Definitions

HP ALLBASE/4GL allows the administrator to disable particular logic commands for particular sites.

Student Notes

System Wide Specifications - Site Synonyms II

Any HP ALLBASE/4GL command may be disabled if the administrator for a particular site does not think it is appropriate.

For example, the **EXTERNAL** command could be disabled so that programmer cannot access the operating system. Any applications imported from another system which allows the **EXTERNAL** command will not run correctly in the new environment.

ADMIN12 - SLIDE: System Security

HP ALLBASE/4GL ADMINISTRATION

System Security

The HP ALLBASE/4GL administrator controls all aspects of system security. The various levels of security provided are:

- Operating System security
- HP ALLBASE/4GL user access control
- Application access control
- Version access control

HP ALLBASE/SQL and HP TurboIMAGE provide their own security which HP ALLBASE/4GL accommodates.

Australian Software Operation
admin12

HEWLETT PACKARD

Student Notes

System Security

Initial Security

HP ALLBASE/4GL provides a number of types of security checking. Before a person can run an HP ALLBASE/4GL application, there may be three levels of security which must be satisfied. Once in the application, further security checks can be performed.

Operating System Security

The user must first be able to successfully log-on to the system.

WARNING

Never countenance a scheme whereby all HP ALLBASE/4GL users log-on with the same operating system name and password! This would be a most effective way to ruin system security by eliminating the ability to discriminate between users.

The only thing worse than this is allowing access without passwords!

HP ALLBASE/4GL User Access Control

The user must type in the name of a valid user, and optionally type in a password for that user.

Application Access Control

The user's name must be on the **access list** for the requested application; they may optionally be asked to enter a password for the application.

If the user is a developer, the password is actually the **development security code**. Without entering the password, the developer is unable to modify secured items.

Version Access Control

This is the same as application access control, but applies to versions.

Security within the HP ALLBASE/4GL Application

HP ALLBASE/SQL Security

HP ALLBASE/4GL users must comply with the security requirements imposed by HP ALLBASE/SQL.

Having gained entry to an HP ALLBASE/SQL-based application, the user must pass the security checks applied by HP ALLBASE/SQL before the database can be accessed.

Developers must have **CONNECT** and **RESOURCE** authority. End-users must have **CONNECT** and **RUN** authority. The operating system user who creates the database using **isql** automatically gains **Database Administrator or DBA** authority; this user must grant authorities to other users. Normally this task would be assigned to the HP ALLBASE/4GL administrator.

WARNING

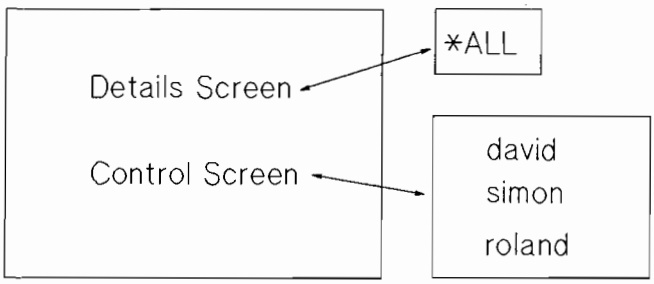
The HP ALLBASE/SQL security scheme is partially based on the operating system login name. If all users have the same operating system login name, then all users effectively have **DBA** authority for the database. Enforce operating system user names and passwords!

Intentionally blank

ADMIN13 - SLIDE: Menu Item Security

HP ALLBASE/4GL ADMINISTRATION

Menu Item Security



Within each application you can secure particular menu items. This means only the authorised user(s) for the menu item can exercise it.

Student Notes

Security within the HP ALLBASE/4GL Application I

Menu Item Security

HP ALLBASE/4GL provides a means of securing parts of an application by allowing the administrator to define an access list for each part of the application's menu tree. This facility is known as **Menu Item Security.**

A list of valid users may be defined for any menu item. Any user, whose name is not on the **access list**, who tries to select the menu item will receive a message indicating that access has been denied.

Note

Menu item security only applies to end-users, not to developers. Developers can always access any menu item.

Also note that end-users can potentially use the menu bypass feature to jump over secured menus. Therefore all sensitive menus in a particular branch should have the same menu item security defined.

ADMIN14 - SLIDE: Development Security Codes

HP ALLBASE/4GL ADMINISTRATION

System Security

Development Security Codes

A development security code can be defined for an application which will prevent unauthorised modification by anybody other than the developers who know the security code.

The following items can be secured:

- field specifications
- record layouts
- validation ranges
- validation tables
- scratch-pad field name declarations
- screens
- reports
- decision tables

Functions and processes are protected by deleting source when unloading.

Australian Software Operation
admin14

HEWLETT PACKARD

Student Notes

Security within the HP ALLBASE/4GL Application II

Development Security Codes

For each application, HP ALLBASE/4GL allows two groups of developers; those who know the application's **development security code** and those who do not. **Development Security Codes** are used when **unsecuring** HP ALLBASE/4GL items which have been secured.

Unsecuring Items

Secured items cannot be modified. All fields will be **display-only** except for the name field. Secured items can only be **unsecured** by a developer who has supplied the application's **development security code** when signing-on.

Securing Items

To secure an item, a developer must place a **Y** in the **Secured** field in the developer screen for that item. Thereafter, only developers who have supplied the **development security code** for the application when logging-on to HP ALLBASE/4GL can change that field to **Y** in order to modify the item.

Unloading Secured Items

The source for secured items will not be unloaded when an application is unloaded. This feature is important for those HP ALLBASE/4GL items which do not generate as the source for these must be included in a runtime version of the application. Unsecured items could be modified by means of **hp4stoa** and **hp4atos**.

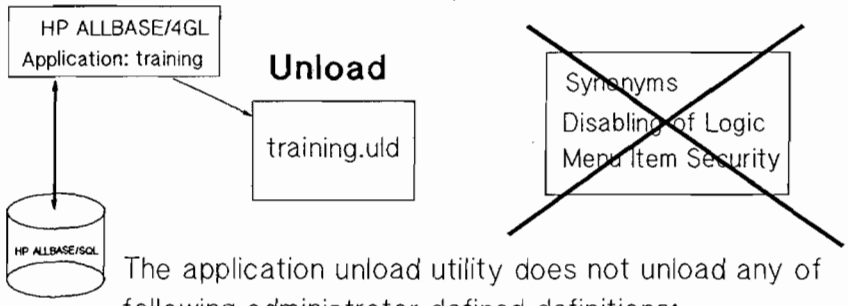
Laboratory Exercise

The students should now do the laboratory exercise entitled **Menu Item Security**

ADMIN15 - SLIDE: Unload - Considerations

HP ALLBASE/4GL ADMINISTRATION

Application and Version Administration



The application unload utility does not unload any of following administrator defined definitions:

* Logic Command Synonyms
* Logic Command Disabling
* Menu Item Security

If needed, these items must be redefined at the target site.

Student Notes

Unload - Considerations

Many of the system wide specifications that are created in the administrator are not taken with an unloaded application because they would override the system definitions at the target site. Administrator definitions are meant to be specific to **one** site. If an application is meant to be transferred, the assumption is that the target system has compatible aims and would already have established the required environment.

If sending an application to a target environment which does not have these definitions, applications with synonyms will not be able to run without modifications. Applications will now be able to use commands previously disallowed and end-users will be able to gain access to all sections of the application.

The items that are not taken are:

- Logic Command Synonyms
- Logic Command Disabling
- Menu Item Security

Note

These definitions can be transferred using the utilities **hp4stoa** and **hp4atos.**

ADMIN16 - SLIDE: Terminal Display Enhancements

HP ALLBASE/4GL ADMINISTRATION

System Wide Specifications

Terminal Display Enhancements

	Brightness	Video	Underline	Blink	Color
Banner	H	I	N	N	C
Non-Active Input Field	H	I	N	N	Y
Active Input Field	F	I	N	N	G

HP ALLBASE/4GL allows the administrator to control the way in which terminals will display screen items:

* full bright normal video
* half bright normal video
* full bright inverse
* half bright inverse
* underlining
* blinking
* color changes

Australian Software Operation
admin16

HEWLETT PACKARD

Student Notes

Terminal Display Enhancements

The **Terminal Display Control** screen allows the administrator to tailor the appearance of HP ALLBASE/4GL screens for all of the applications in the S-files.

By changing certain attributes, HP ALLBASE/4GL applications can take on a very different appearance.

Attributes

The screen allows the administrator to change the appearance of all HP ALLBASE/4GL screen items by specifying the attributes **brightness, video attribute, underlining, blinking** and **color.**

Brightness can be **Full** or **Half.** The **video attribute** can be **Reverse Video** or **Normal Video. Underlining** can be **On** or **Off. Blinking** can be **On** or **Off.**

Colors

The allowable colors are **Red, Green, Blue, Cyan (Light Blue), Magenta, Yellow** and **White.**

Note

When setting color attributes, many HP terminals require **full brightness** before the color can be displayed correctly.

Screen Operation

The administrator's changes are shown immediately on the special fields on the screen; the actual screen continues to behave according to the previous settings until the administrator commits the screen, returns to the previous menu and then re-enters the screen.

Default Settings

By default, non-active input fields are shown in **half-bright reverse video.** The active input field is shown by **full-bright reverse video.**

Display fields are shown in **full-bright normal video.**

Un-selected menu items are shown in **half-bright normal video.** Selected menu items are shown in **full-bright reverse video** and when activated, the menu item changes briefly to **half-bright reverse video.**

The default settings are shown in the HP ALLBASE/4GL Administrator Reference Manual.

Global versus Local Settings

As mentioned above, the settings on this screen apply to all applications in the S-files, so the settings are **global.** It is possible for individual applications to override certain settings by use of the **FIELD** command, providing **local** settings for that application.

Demonstration

Ask the students to gather around a terminal while you demonstrate the behaviour of the screen.

Intentionally blank

ADMIN17 - SLIDE: System Wide Specifications - Master Titles

HP ALLBASE/4GL ADMINISTRATION

System Wide Specifications

Master Titles

DATA DICTIONARY

Titles

Application Titles - Application Wide
Master Titles - System Wide

* Application Titles are literal strings that can be displayed on screens or within reports.
* Master Titles are like Application Titles, but are maintained by the administrator on a system wide basis.
* When a Master Title is referenced in an application, '4GL looks for an Application Title of the same name. If none is found, HP ALLBASE/4GL uses the appropriate Master Title.

HP ALLBASE/4GL allows the definition of Master Titles that can be used by ALL application developers on the system.

Australian Software Operation
admin17

HEWLETT PACKARD

Student Notes

System Wide Specifications - Master Titles

Master Titles are literal strings that are **globally available** for an entire set of S-files. This means that any application in those S-files can use the **master title** without re-defining it in the dictionary for that application.

If an application tries to use an **application title** which does not exist, a **master title** with the same name may be found instead. In this way, there can exist **global** literals which may be over-ridden by applications on an individual basis.

Application Titles are local titles; **Master Titles** are global titles.

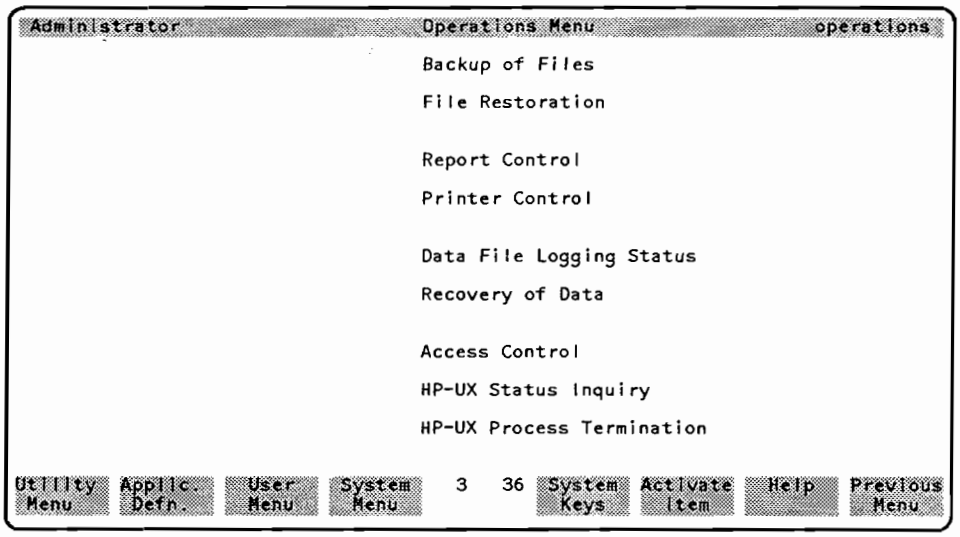
Laboratory Exercise

The is no laboratory exercise for **System Wide Specifications** because of the possibility of interaction between the different students. Normally there is only one HP ALLBASE/4GL administrator!

The students should experiment with the various **System Wide Specifications** in their own time.

ADMIN18: The HP-UX Operator Functions

The **Operations Menu** available on HP-UX:



Student Notes

The HP-UX Operator Functions

On HP-UX, HP ALLBASE/4GL supplies an **Operator** menu, called from the **Main Menu**, which allows the administrator to perform the operations shown on the menu.

Backup of Files

The administrator is able to initiate a **full** or **incremental** backup for the entire filesystem or just for the HP ALLBASE/4GL filesystem.

The administrator has the choice of **cartridges**, **magnetic tapes** or **floppy disks**; HP ALLBASE/4GL calculates the number of volumes required as determined by the media type and density.

File Restoration

This option allows the administrator to restore files from a backup created within HP ALLBASE/4GL.

Report Control

The HP ALLBASE/4GL administrator has full control over HP ALLBASE/4GL reports. The administrator can:

- List reports

  The display shows the report number, the printer name, the report name, the name of the user generating the report, the paper type used for the report, the date and time the report was submitted to the queue, and the number of copies of the report being printed.

  The administrator can scroll forwards and backwards through the list of reports using the `Next Page` and `Previous Page` function keys.

- Cancel reports

  The administrator can cancel any report by specifying its number and details.

- Restart reports

The administrator can restart reports, for example after a paper jam.

Printer Control

The HP ALLBASE/4GL administrator can:

- List the current status of the system printers
- Oversee requests to change paper

HP ALLBASE/4GL displays a paper change request message if a user queues a report which requires different paper from that currently available on the printer.

The administrator can press the `Change Paper` function key after changing the paper on the printer.

- Enable printers

If HP-UX disables a printer, the administrator can enable the disabled printer.

Data File Logging

The HP ALLBASE/4GL administrator can enable **data file logging** for all ISAM data file transactions since the most recent backup.

Recovery of Data

The HP ALLBASE/4GL administrator can initiate **roll-forward recovery** to rebuild application data files that have been lost or corrupted due to system failure or operator error.

Access Control

The administrator can **enable** or **disable** access to HP ALLBASE/4GL. Disabling access prevents new users logging on to the HP ALLBASE/4GL system; current users can continue to work until they log out.

Access is normally disabled before a backup or recovery operation; the administrator can then enable access to users after the operation.

HP-UX Status Inquiry

The administrator can obtain a list of processes running on the system; the list can cover all processes or just those for a particular user.

HP-UX Process Termination

The administrator can terminate HP-UX processes from within HP ALLBASE/4GL.

Laboratory Exercise

The students should now do the laboratory exercise entitled **Operator Functions on HP-UX.**

Versions

Perform the following steps:

1. Create a version of your application; simply append **v1** to the name of your application.

2. Log in to the version; paint the ***VERSION** communication area field onto the main menu, to demonstrate that the version is different to the base application.

3. Make a change to a function key set in the version and demonstrate that the change is not in the base application.

4. Make a change to a different function key set in the base application and demonstrate that it is reflected automatically in the version.

Unloading and Loading

Perform the following steps:

1. Unload your application without the version. Note the numbers reported for each file.

2. Escape to the operating system and examine the unloaded file. If running on HP-UX, look in the **HP4SPATH** directory using ls -l; if running on MPE XL, look in the **HP4APPN** group using **listf name,2**.

3. Return to HP ALLBASE/4GL and load the application; during the process, rename it by appending the number **2** to the original name. Note the numbers reported for each file.

4. Delete the renamed application. Note the numbers reported for each file.

5. Unload your application with the version. Note the numbers reported for each file.

Menu Item Security

Add menu-item security provisions to protect the **resource allocation** section of your application.

Demonstrate that it is effective.

Operator Functions on HP-UX

If running on HP-UX, work with students on another terminal and experiment with the **Operator Functions.**

1. Run a report from one of your applications and immediately use the **Report Control** screen to monitor the status.

2. Investigate the **Printer Control** screen.

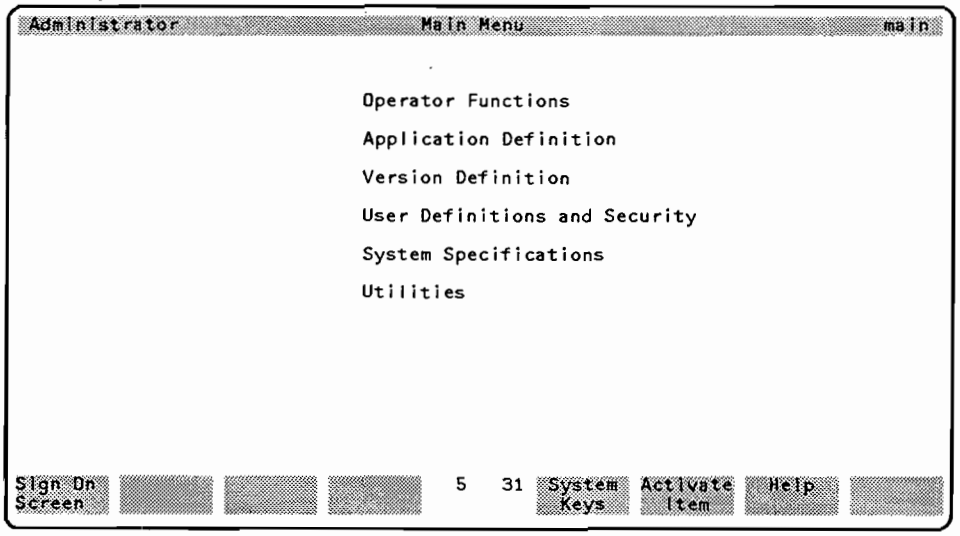
3. Use the **HP-UX Status Inquiry** screen to monitor the processes running on one of your terminals.

4. Use the **HP-UX Process Termination** screen to terminate the HP ALLBASE/4GL session running on the other terminal.

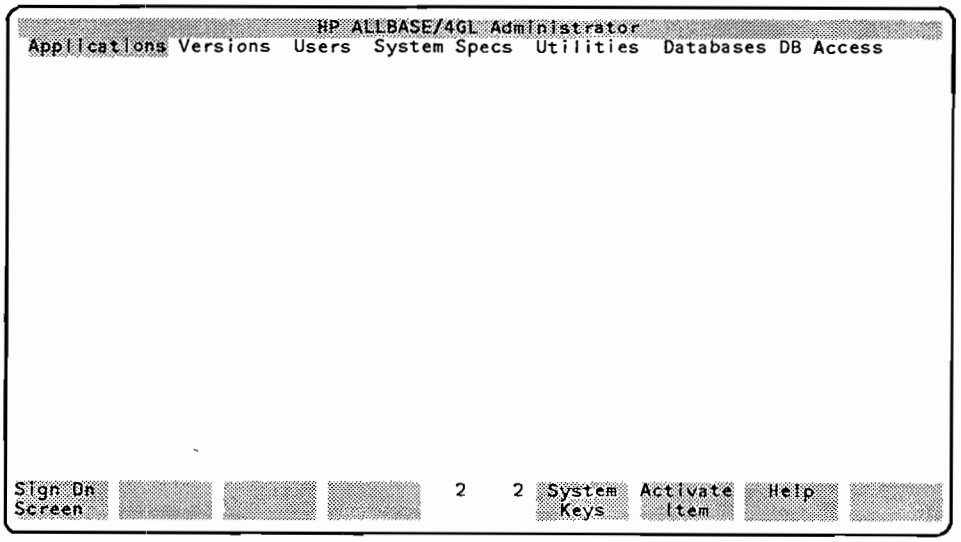
5. Browse through the other screens within the **Operator Functions** section of the administrator.

Screen Images

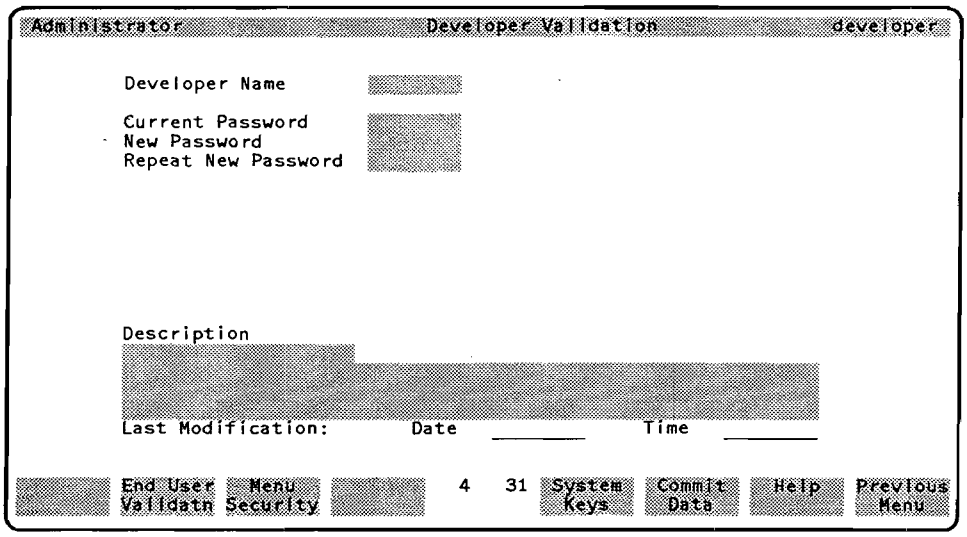
The screen image for the HP-UX version of the **Main Menu** screen:



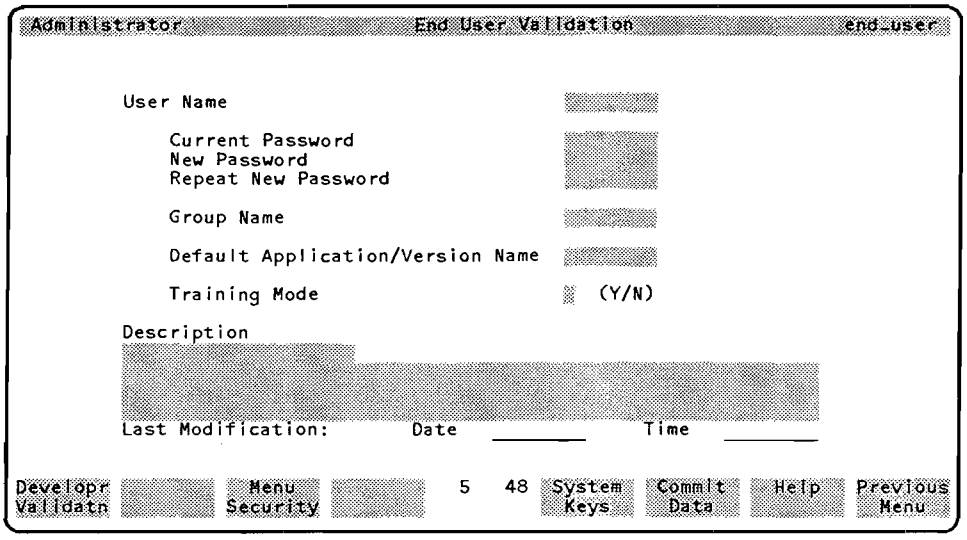
The screen image for the MPE XL version of the **Main Menu** screen:



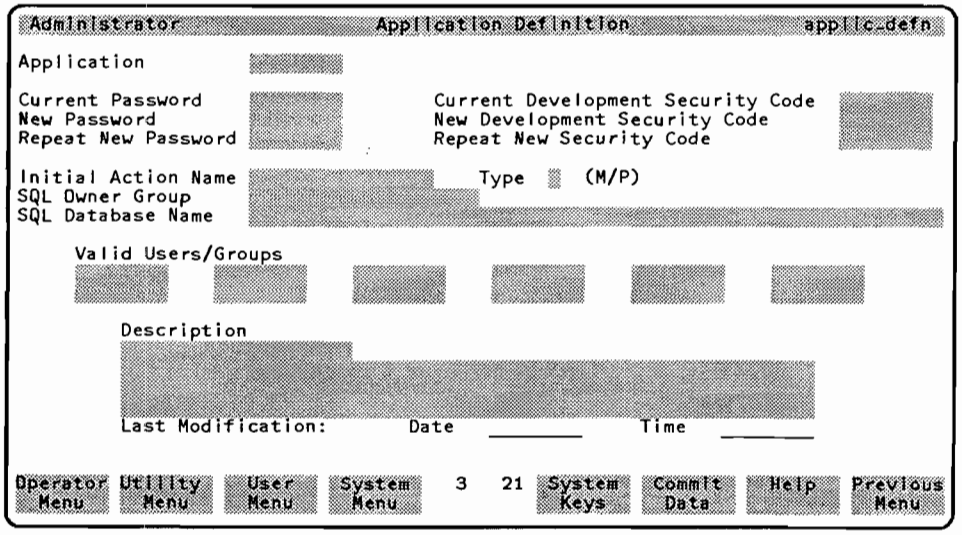
The screen image for the **Developer Validation** screen:



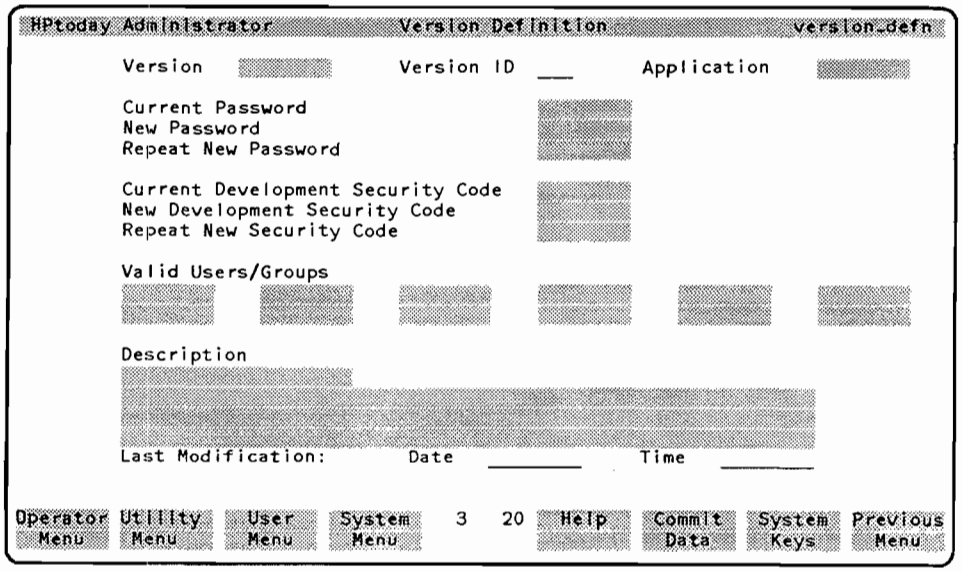
The screen image for the **User Validation** screen:



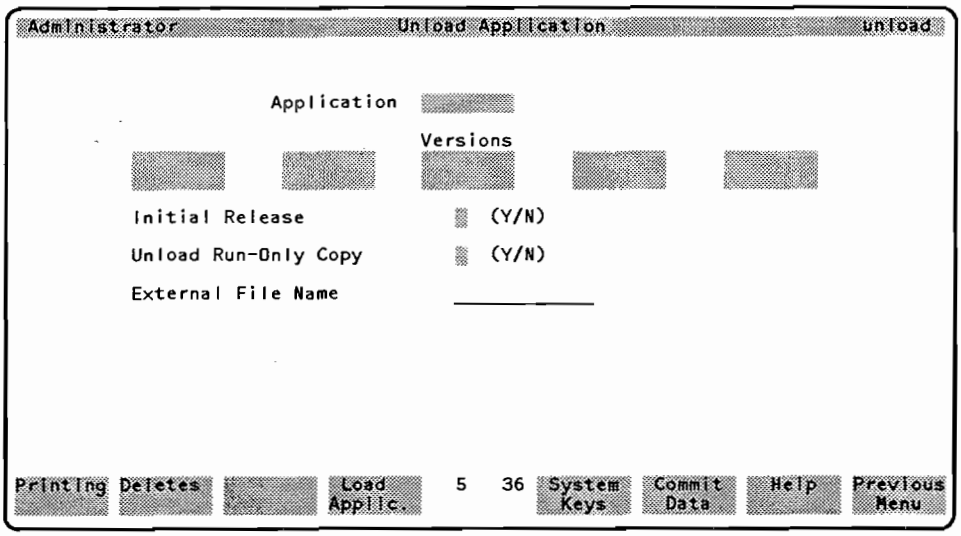
The screen image for the **Application Definition** screen:



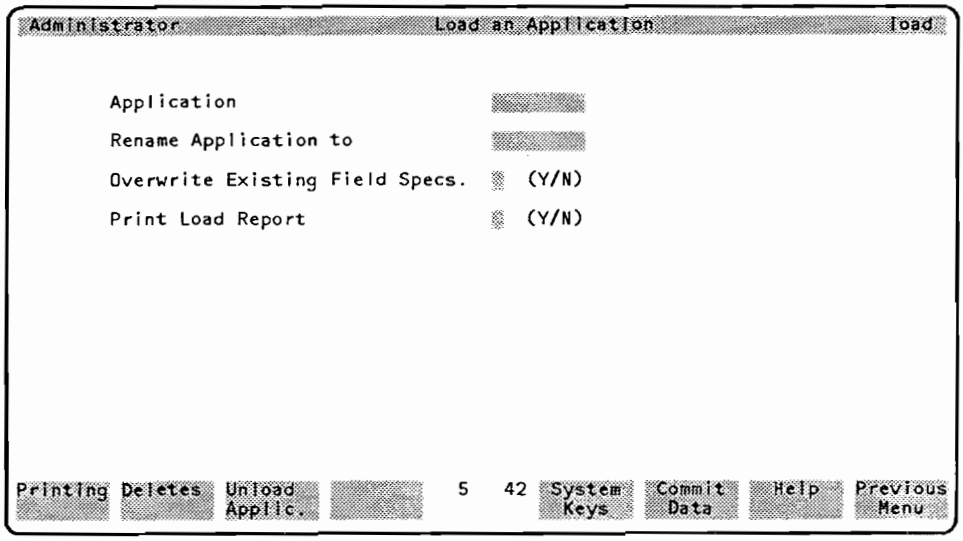
The screen image for the **Version Definition** screen:



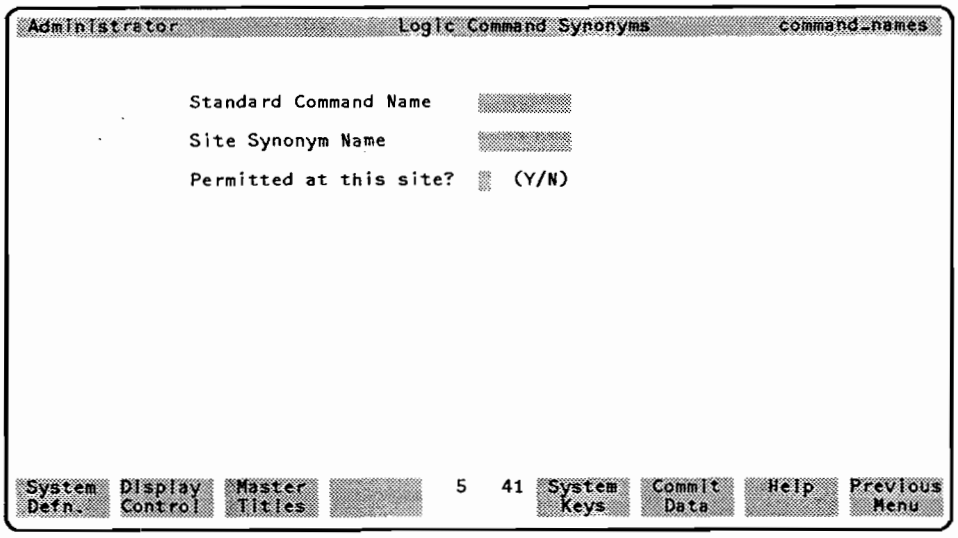
The screen image for the **Application Unload** screen:



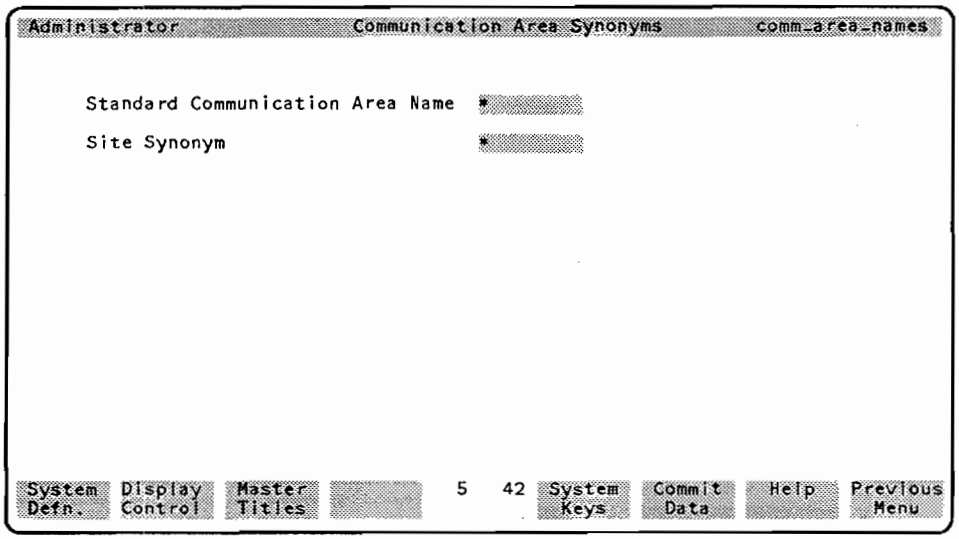
The screen image for the **Application Load** screen:



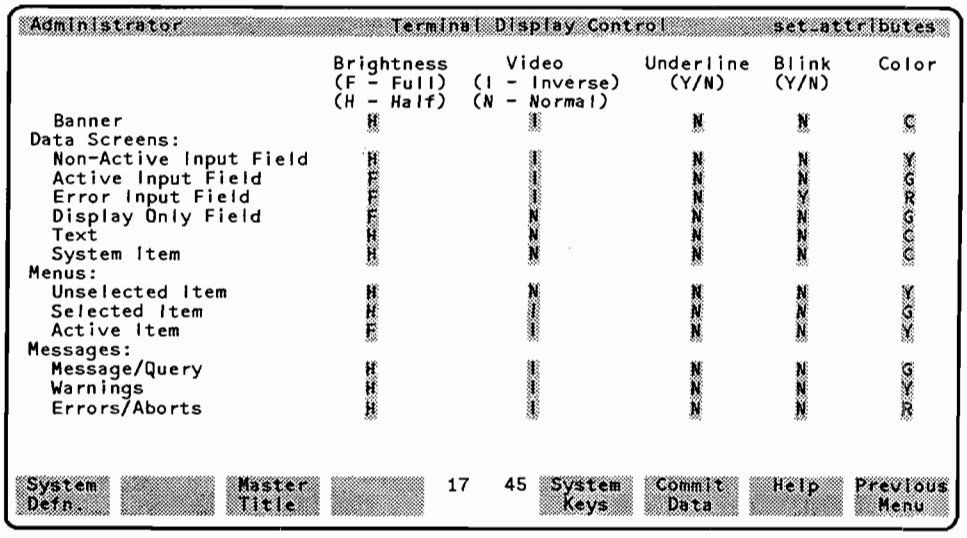
The screen image for the **Logic Commands Synonyms** screen:



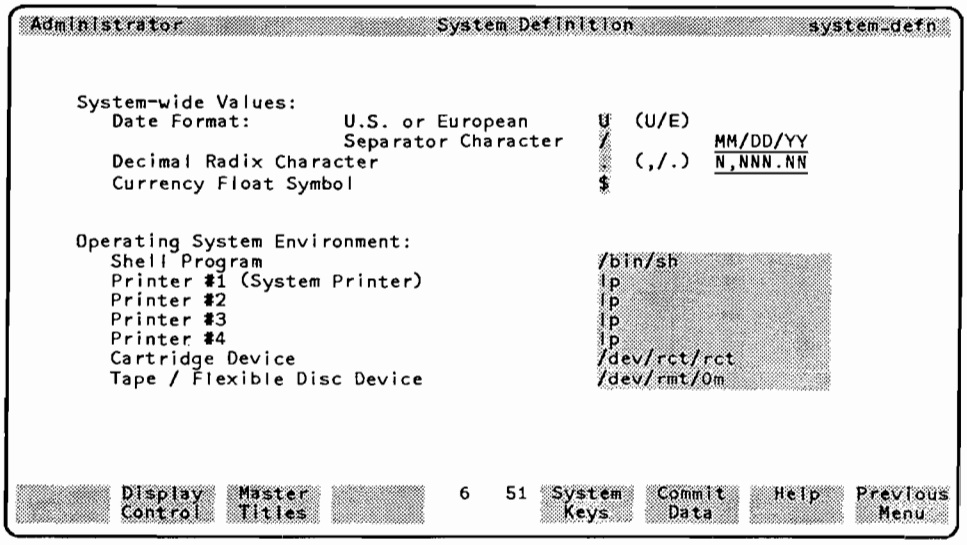
The screen image for the **Communication Area Synonyms** screen:



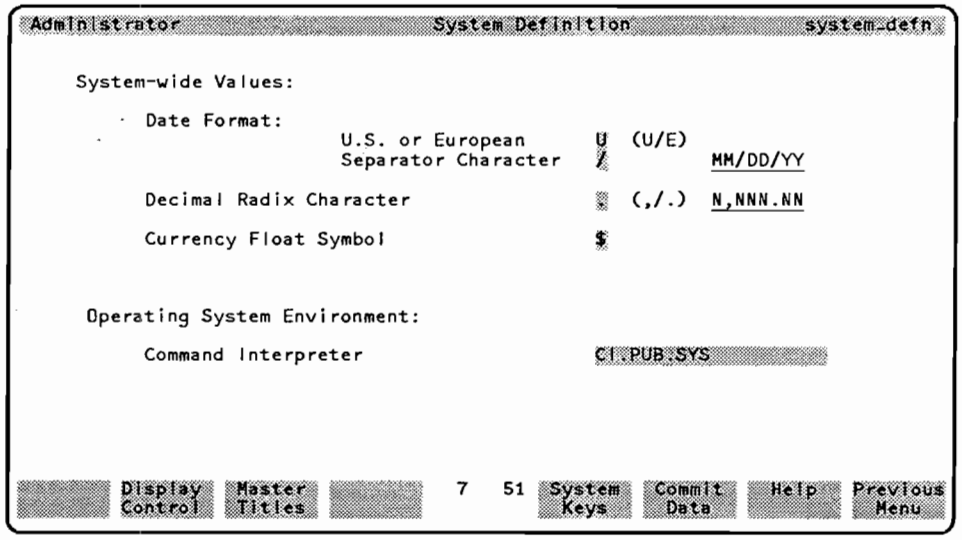
The screen image for the **Terminal Display Control** screen:



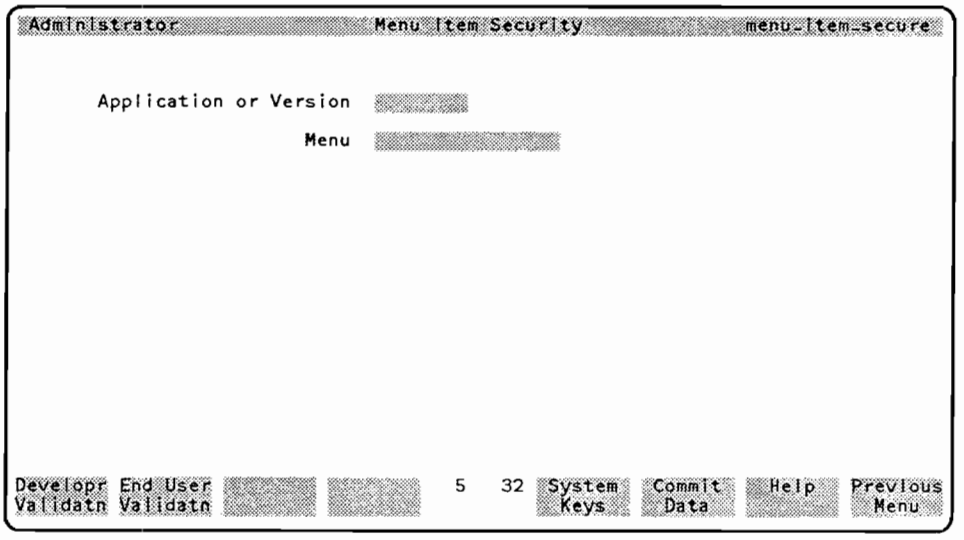
The screen image for the HP-UX version of the **System Definition** screen:



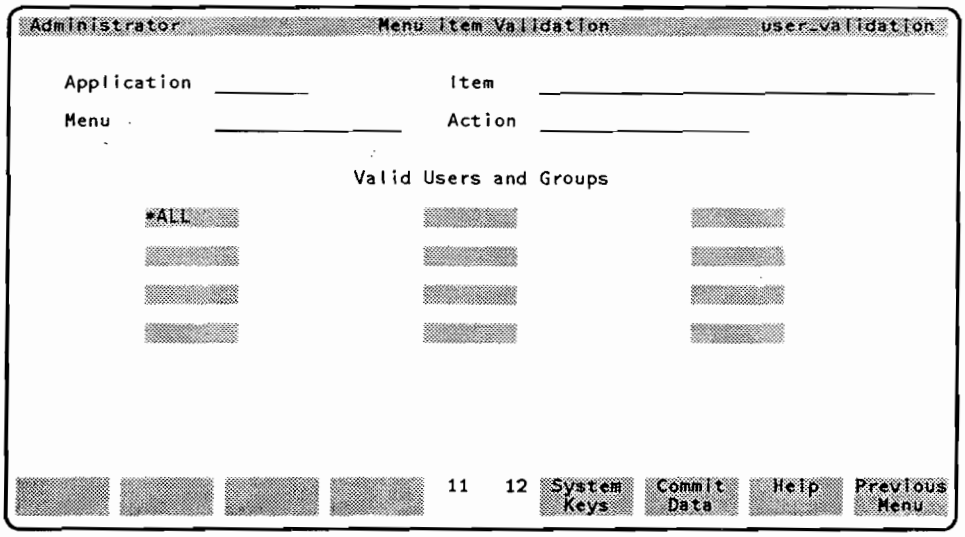
The screen image for the MPE XL version of the **System Definition** screen:



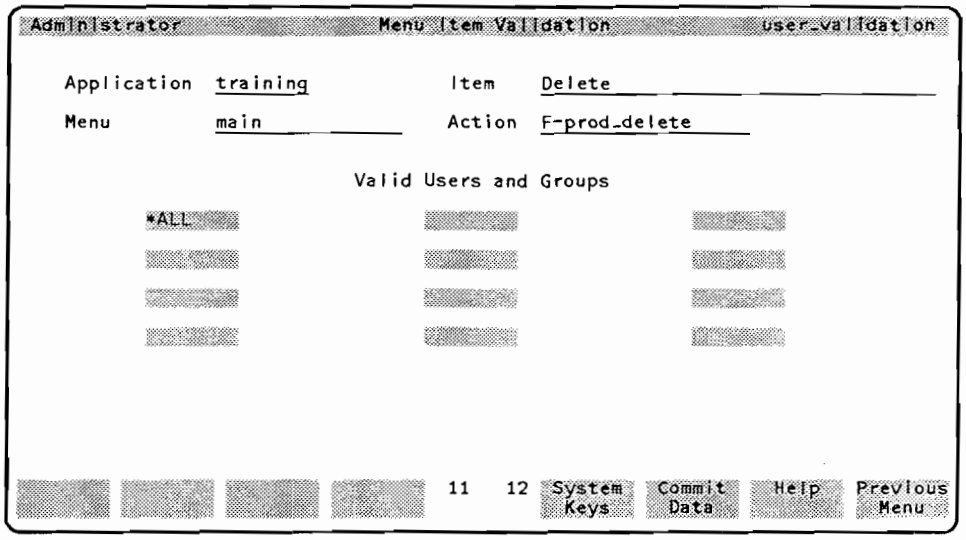
The screen image for the **Menu Item Security** screen:



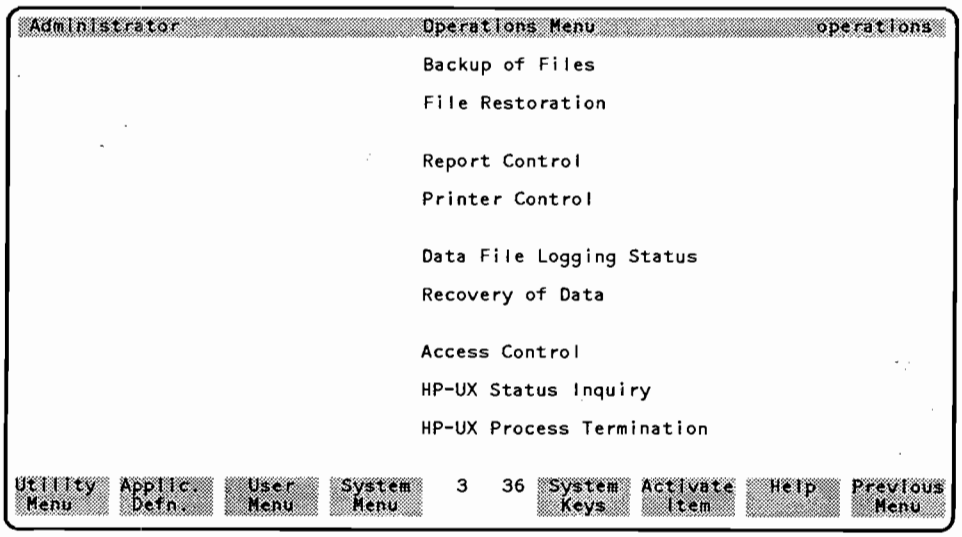
The screen image for the **Menu Item Validation** screen:



The screen image for an example **Menu Item Validation** screen:



The screen image for the **Operations Menu** screen:



Module: Operating System Environment

Instructor Notes

OPSYS00 - SLIDE: Module Objectives

HP ALLBASE/4GL OPERATING ENVIRONMENT

HP ALLBASE/4GL Files

Environment Variables

HP ALLBASE/4GL File System

Startup Scripts

OBJECTIVES:

Upon completion of this module, the student will be able to:

1. Describe the files used by HP ALLBASE/4GL and their purpose.
2. Describe the HP ALLBASE/4GL environment variables and their purpose.
3. Describe the layout of HP ALLBASE/4GL files and how this relates to the environment variables.
4. Understand the purpose of the HP ALLBASE/4GL startup script, how it is implemented and how it can be used to customize the HP ALLBASE/4GL environment.

Student Notes

Module Objectives

This module will introduce the students to the HP ALLBASE/4GL Operating System Environment under both HP-UX and MPE XL.

Student Objectives

At the end of this module, each student will be able to do the following:

- Describe the files used by HP ALLBASE/4GL and their purpose.
- Describe the HP ALLBASE/4GL environment variables and their purpose.
- Describe the layout of HP ALLBASE/4GL files and how this relates to the environment variables.
- Understand the purpose of the HP ALLBASE/4GL startup script, how it is implemented and how it can be modified to customise the HP ALLBASE/4GL environment.

Resource Allocation

Equipment

The following items are required for this module:

- Instructor notes.
- Student notes.
- Slides.
- A laboratory worksheet.
- Access to an HP 3000 Series 900 machine or an HP 9000 Series 800 or HP 9000 Series 300 machine with HP ALLBASE/4GL installed.
- The complete set of HP ALLBASE/4GL manuals.

Time

This module should take approximately 1.5 hours, with 1 hour for theory and 30 minutes for the 2 laboratory sessions. Each laboratory session should take approximately 15 minutes.

Pacing

Read through the slides reasonably quickly.

OPSYS01 - SLIDE: HP ALLBASE/4GL Files

HP ALLBASE/4GL OPERATING ENVIRONMENT
HP ALLBASE/4GL Files

HP ALLBASE/4GL uses several different types of files. They are:

* HP ALLBASE/4GL S-files.
* HP ALLBASE/4GL program files.
* Application data files.

Student Notes

HP ALLBASE/4GL Files

HP ALLBASE/4GL can use four types of files:

S-Files

The **S-Files** or **Special Files** are used to store user's applications; the HP ALLBASE/4GL **administrator** and **developer** applications are also stored in the **S-files**.

The **S-files** are similar to ISAM/KSAM files.

Program Files

The **binaries** and **scripts/command files** that are used to develop and run HP ALLBASE/4GL applications.

Application Data Files

The data files that user's applications create and modify. They may be serial or ISAM/KSAM files, HP ALLBASE/SQL tables or HP TurboIMAGE data sets.

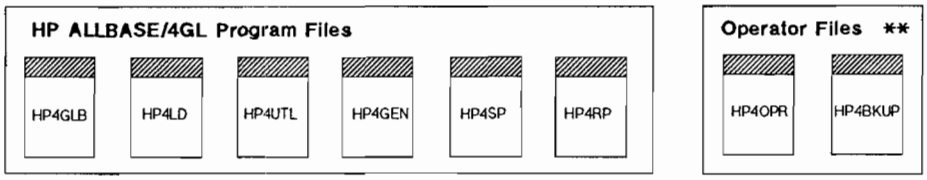
Auxiliary Files

Programs written in other languages can be called from HP ALLBASE/4GL applications. They might be written in a 3GL language such as **C**, **Pascal** or **Fortran** etc., or perhaps HP-UX shell scripts or MPE XL command files.

OPSYS02 - SLIDE: HP ALLBASE/4GL Files - Program Files

HP ALLBASE/4GL OPERATING ENVIRONMENT

HP ALLBASE/4GL Files
Program Files



The program files are the executable binary program files that run HP ALLBASE/4GL. Only one set of these is required on each system.

On an MPE XL system, HP ALLBASE/4GL is stored in PUB.SYS.

All files start with the prefix "hp4" or "HP4".

On HP-UX systems, the program files are usually kept in a separate "bin" directory for HP ALLBASE/4GL.

On HP-UX systems, a number of programs are included to run the "administ" Operator Facility. This is not required on MPE XL.

Student Notes

Refer to the page entitled **HP ALLBASE/4GL Program Files: Summary** at the end of this module.

HP ALLBASE/4GL Files - Program Files

HP ALLBASE/4GL is actually a large suite of programs. The source code is mainly written in the **C** programming language, with some of the HP ALLBASE/SQL interface code written in **Pascal**. There are over 200,000 lines of code.

The main binary HP4GLB

The main HP ALLBASE/4GL developer binary is called **hp4glb**; the runtime binary is called **hp4glbr**. All other programs start with the **hp4** prefix and runtime versions have the letter **r** appended. Some binaries are not used in the runtime environment.

External Binaries

While running, **hp4glb** will call in a number of other external binaries. These include: **hp4gen**, the **generates** program; **hp4sp**, the **screen painter** program; **hp4rp**, the **report painter** program. These are some of the programs called by the HP ALLBASE/4GL developer.

Other files are used during program execution, such as **hp4sort** for report sorting. Still others are called from the HP ALLBASE/4GL administrator, such as **hp4uld** and **hp4ld** for application unloading and loading respectively.

Generates

You may have noticed that the first time the developer attempts to generate an HP ALLBASE/4GL item, a **Please wait** message appears while **hp4gen** loads. For subsequent operations, generation begins immediately; once loaded, **hp4gen** remains resident along with **hp4glb**.

Shell Scripts & Command Files

On HP-UX, there are a number of **shell scripts** which are called from the **administrator** application to perform some operator functions, such as **printer control** and **report control**. Furthermore, there is a special script called **hp4_lp** which is used whenever an HP ALLBASE/4GL report is sent to the printer.

None of those scripts are used with the MPE XL version of HP ALLBASE/4GL.

The startup script HP4GL

However, on both HP-UX and MPE XL, a startup script/command file called **hp4gl** is supplied; the purpose of this is to establish the correct environment before calling the main HP ALLBASE/4GL binary.

Note

The HP ALLBASE/4GL system administrator may wish to modify **hp4gl** to change the environment or structure. Please keep the original! If any problems are detected while running HP ALLBASE/4GL with a different environment, please test the problem with the standard **hp4gl** startup script.

The 'bin' directory & the PUB.SYS group

On HP-UX, the HP ALLBASE/4GL program files are stored in the **bin** directory underneath the **HP4GL** directory where HP ALLBASE/4GL was installed.

On MPE XL, the HP ALLBASE/4GL program files are stored in the **PUB.SYS** group. Developer and runtime binaries can co-exist as the runtime binaries have the letter **R** appended.

Note

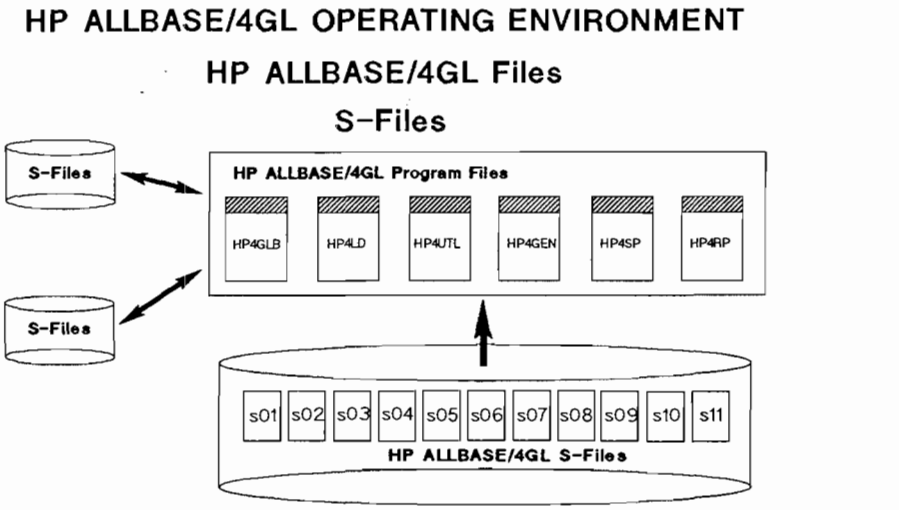
For the **Evaluation Pack** version, a separate account called **HP4DEMO** is created and the binaries are stored in the **BIN** group.

Binaries and S-Files

Only one set of HP ALLBASE/4GL binaries is necessary for any one machine. However, many sets of S-files can share the same set of HP ALLBASE/4GL program files.

For example, it would be possible to have two sets of S-files, one for development and the other for testing, both using the same set of HP ALLBASE/4GL program files. Of course, a separate version of the **hp4gl** startup script would be required to point to the new set of S-files. More on how to do that later.

OPSYS03 - SLIDE: HP ALLBASE/4GL Files - S-Files



The HP ALLBASE/4GL S-files are a set of 11 ISAM/KSAM-like data files which together form a data base of all application objects (source and HP ALLBASE/4GL generated form), and system definition items.

A system may have more than one set of S-files, serviced by one common set of program files.

Australian Software Operation
opsys03

HEWLETT PACKARD

Student Notes

Each S-file contains objects of similar purpose and/or size. These are broken up as follows:

S01	Terminal Attributes	S07	Field Specifications
S02	Users and Groups	S08	System Objects, Namelists
S03	Applications, Versions	S09	Storage Type Objects
S04	Object Descriptions	S10	Executable Object Source
S05	Messages, Titles	S11	Generated Executable Object
S06	Help Screens		

Each set of S-files is independent of any other set on the system. Each can support different system-wide configurations.

HP ALLBASE/4GL Files - S-Files

Physical Structure

Physically, the S-files are 11 ISAM/KSAM data files. As you know, ISAM/KSAM files consist of a data file and an index file.

On HP-UX, the files with **.i** extensions are the index files, the **.d** files contain the data; for example, the two **s04** files are **s04.i** and **s04.d**.

On MPE XL, the index files have the letter **I** appended to the name and the data files have the letter **D** appended to the name; for example, the two **S04** files are **S04I** and **S04D**.

S01	Terminal Attributes	S07	Field Specifications
S02	Users and Groups	S08	System Objects, Namelists
S03	Applications, Versions	S09	Storage Type Objects
S04	Object Descriptions	S10	Executable Object Source
S05	Messages, Titles	S11	Generated Executable Object
S06	Help Screens		

Each set of S-files is independent of any other set on the system. Each can support different system-wide configurations.

Different HP ALLBASE/4GL items are stored in each file, for example, the screen source may be stored in **S10**, and the generated form of the screen in **S11**.

Question

Why are there 12 different types of files?

Answer:

Each file has its own record layout (structure and length), appropriate for the type of item it is to store.

WARNING

The 22 files which go to make up a set of S-files must be seen as one unit. An individual HP ALLBASE/4GL object may be split across more than one file, so it is vital for the integrity of the application that both files reflect the same instantaneous snap-shot of the object.

Never contemplate restoring a subset of the S-files from backup; it is imperative to get the entire set.

The only exception to this is with the **S11** file. It may be possible to restore just this file as it only contains **generated forms** of HP ALLBASE/4GL objects. A consistent set of **generated records** could be re-created by performing a **Generate ALL** for all applications within the S-files.

System Master Record

The **S03** file contains the **System Master Record** for each application; this is created by committing the **Application Definition** screen. It is included in an **Application Definition File** when the **Initial Release** field is set to **Y** on the **Unload** screen.

Contents

The **developer** and **administrator** application are in every set of S-files. Also some example applications are included in every set of S-files. The applications are: **example**, a small telephone accounting program; **tutorial**, which explains some HP ALLBASE/4GL concepts; and **bike**, a small HP ALLBASE/SQL-based application. These applications require the developer to create some data files before they will work.

There is one **adminis**t application per set of S-files; any system configurations that are defined in the **adminis**t application will apply to all user applications in that set of S-files.

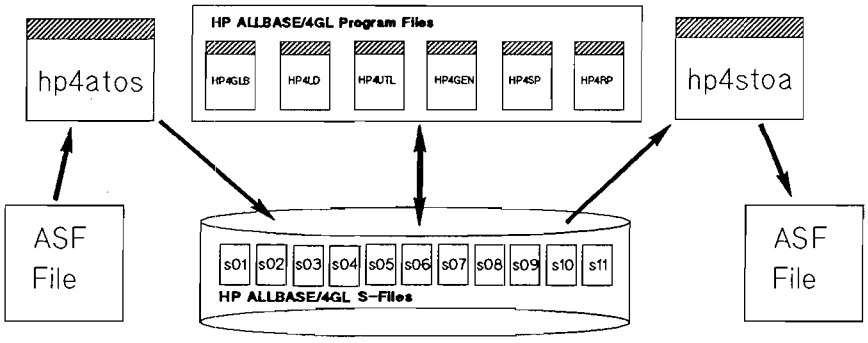
Intentionally blank

OPSYS04 - SLIDE: HP ALLBASE/4GL Files - ASF Files

HP ALLBASE/4GL OPERATING ENVIRONMENT

HP ALLBASE/4GL Files

S-Files



The S-file system is a proprietary data file implementation similiar to ISAM or KSAM files. However, all objects can be read or written using the hp4stoa (S TO Ascii) and hp4atos (Ascii TO S) tools.

Student Notes

On HP-UX, unload **debtors** to **debtors.asf**

On MPE XL, create a group called **ASF**. Then unload **debtors** to **DEBTORS.ASF**

HP ALLBASE/4GL Files - ASF Files

ASF Files are **ASCII** files containing an entire HP ALLBASE/4GL application or certain objects from the application.

The term **ASF** means **Ascii Special File** or **Ascii S-File**, implying that it is an **ASCII** version of the HP ALLBASE/4GL S-files.

Creation of ASF files

An **ASF** file is produced by the utility **hp4stoa**; the complimentary operation of writing the contents of an **ASF** file back to the S-files is performed by the utility **hp4atos**.

Uses of ASF files

ASF files can be used for transporting entire HP ALLBASE/4GL applications. Another use is for inserting new or changed objects into an existing application; this operation cannot be done using **hp4uld** and **hp4ld**, as they can only work with an entire HP ALLBASE/4GL application.

However, perhaps the greatest advantage of **ASF** files is that they allow the developer to use a standard text editor, such as **vi** or **emacs** on HP-UX and **HPEDIT** on MPE XL, to make changes to the application. This could range from small changes to just one object, such as a process, to global changes throughout the application.

For example, the task of localising (or translating into a foreign language) the message and screen text in an application could be done quite quickly using the **global search and replace** facilities of an external editor.

The number of uses for **ASF** files is enormous!

Location and Naming of ASF files

On HP-UX:

By convention, **ASF** files are given the **.asf** extension. So, to unload the application **debtors** (or part thereof), the **ASF** file would be **debtors.asf**.

Many users choose to create **ASF** files in the **HP4SPATH** directory, along with the **.uld** files. For convenience, all **ASF** files could be kept in a new directory called **./HP4GL/asf**.

On MPE XL:

It is best to create a new group called **ASF** within the account containing the HP ALLBASE/4GL S-files. Using this scheme, the **ASF** file for **debtors** could be referenced as **debtors.asf**, maintaining consistency with HP-UX.

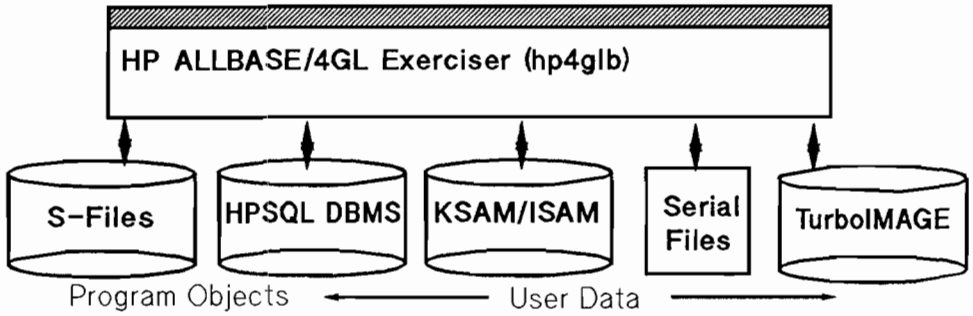
Intentionally blank

OPSYS05 - SLIDE: HP ALLBASE/4GL Files - Application Data Files

HP ALLBASE/4GL OPERATING ENVIRONMENT

HP ALLBASE/4GL Files

APPLICATION DATA FILES.



The HP ALLBASE/4GL data files contain the data used by the enduser's application.

HP ALLBASE/4GL provides facilities to access different data management systems.

Student Notes

HP ALLBASE/4GL Files - Application Data Files

HP ALLBASE/4GL can access a number of different sorts of data files/data bases. All HP ALLBASE/4GL systems can use **Serial files** (variable length or fixed length) and **HP ALLBASE/SQL** tables.

On HP-UX, an application can also use **ISAM** data files.

On MPE XL, an application can also use **KSAM** data files and **HP TurboIMAGE** data sets.

Directories & Groups

ISAM/KSAM and Serial Files

The user's ISAM/KSAM and serial file data is normally kept in the **data** directory on HP-UX and the **HP4DATA** group on MPE XL; the environment variable **HP4DATAPATH** determines where HP ALLBASE/4GL will look for these files.

Note

The environment variables used by HP ALLBASE/4GL will be explained more fully in later slides.

HP ALLBASE/SQL Databases

On HP-UX, HP ALLBASE/SQL databases should probably be kept in a directory called **sql**, although they may also be kept in the **data** directory. On MPE XL, they are kept in a group called **HP4SQL**. The environment variable **HP4DATAPATH** determines where HP ALLBASE/4GL looks for HP ALLBASE/SQL databases.

HP TurboIMAGE Databases

On MPE XL, HP TurboIMAGE databases are kept in the **HP4TI** group; the environment variable **HP4TIPATH** determines where HP ALLBASE/4GL looks for HP TurboIMAGE databases.

Conventions

These directory and group names are merely conventions; administrators and developers should feel free to change the names if so required.

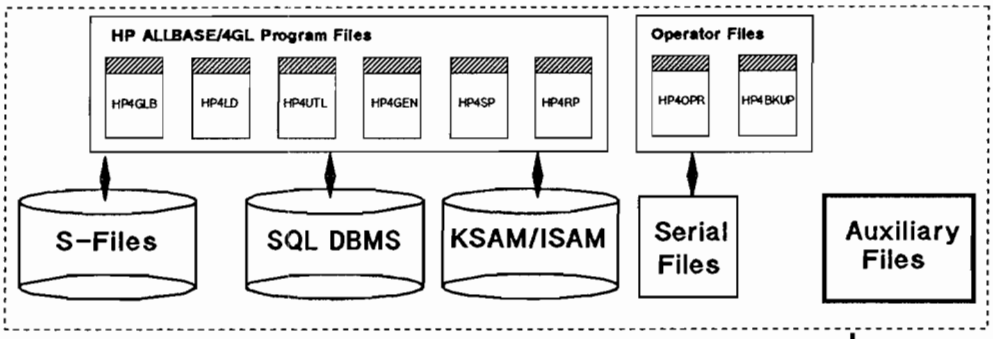
Intentionally blank

OPSYS06 - SLIDE: HP ALLBASE/4GL Files - Auxiliary Files

HP ALLBASE/4GL OPERATING ENVIRONMENT

HP ALLBASE/4GL Files

Auxiliary Files



Auxiliary files are files that are used in conjunction with HP ALLBASE/4GL applications. They are not part of, nor are they necessarily directly accessed by the HP ALLBASE/4GL system.

Auxiliary Files are an historical convention. **

Australian Software Operation
opsys06

HEWLETT PACKARD

Student Notes

HP ALLBASE/4GL Files - Auxiliary Files

Auxiliary files are executable files or data files that are going to be called from user's HP ALLBASE/4GL applications.

On HP-UX, HP ALLBASE/4GL uses the **HP4APATH** environment variable to point to the **aux** directory. HP ALLBASE/4GL will include the **aux** directory in any HP ALLBASE/4GL-operator initiated backup. Hence, **auxiliary files** must be included in the **aux** directory if they are to be backed up.

There is no corresponding environment variable on MPE XL as HP ALLBASE/4GL does not support operator backup.

On HP-UX, for an application to find auxiliary programs, the **aux** directory must be included in the user's **PATH** environment variable. If an application uses external auxiliary files, the HP ALLBASE/4GL start-up script should contain the line:

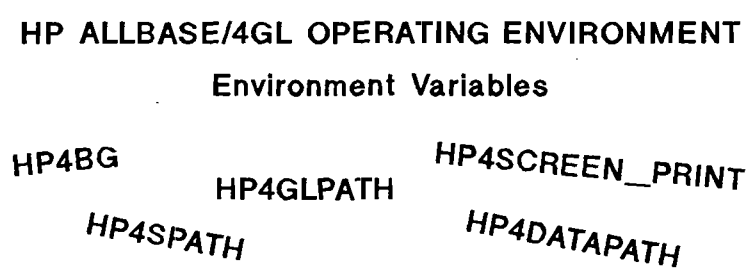
```
PATH=$PATH:$HP4APATH
```

to ensure that the user's path includes the **aux** directory.

On MPE XL, **auxiliary files** can be stored in the logon group (usually **PUB**), however this will quickly clutter the group with files. The administrator could very easily create a group called **HP4AUX** in which to store auxiliary files. In this case, the user's **HPPATH** environment variable must be modified to include the new group so that external programs will be located when called from an HP ALLBASE/4GL application.

The use of **aux** or **HP4AUX** is purely a convention.

OPSYS07 - SLIDE: Environment Variables



HP ALLBASE/4GL uses environment variables to define important information used by the HP ALLBASE/4GL program.

On HP-UX the HP ALLBASE/4GL environment variables are implemented using HP-UX user environment variables.

On MPE XL the HP ALLBASE/4GL environment variables are implemented using XL (command interpreter) variables.

Student Notes

HP-UX - Bourne Shell and Korn Shell

```
VAR=value; export VAR

For example:  HP4SPATH=test_sfiles; export HP4SPATH
```

C-shell

```
setenv VAR=value

For example:  setenv HP4GLPATH=/tmp/HP4GL
```

On MPE XL

```
setvar VAR "value"

For example:  setvar HP4SPATH "hp4s"
```

Environment Variables

Why Use Them?

In attempting to make the HP ALLBASE/4GL environment as flexible as possible, the HP ALLBASE/4GL binaries have been designed to look for **environment variables** (sometimes called **system variables** on MPE XL). The other method, used by many other programs, of having hard-coded paths or group names is very inflexible. Some would even say **brain-damaged!**

Able to be Modified

The values of the environment variables are established in the **hp4gl** start-up script/command file. As this is an **ASCII** file, users can modify it to change the settings of the environment variables to suit their individual needs.

HP-UX

On HP-UX, HP ALLBASE/4GL environment variables are implemented using HP-UX user environment variables. They are not established by the system. HP-UX allows the user to create environment variables with any name, even **foo!**

Environment variables are set differently according to the user's shell, as follows:

Bourne Shell and Korn Shell

With either **sh**, the **Bourne Shell**, or **ksh**, the **Korn Shell**, the syntax is:

```
VAR=value; export VAR
```

where **VAR** is the name of the environment variable.

For example, to set **HP4SPATH** to **test_sfiles**, the syntax is:

```
HP4SPATH=test_sfiles; export HP4SPATH
```

C-shell

With **csh**, the **C Shell**, the syntax is:

```
setenv VAR=value
```

For example, to set **HP4GLPATH** to **/tmp/HP4GL**, the syntax is:

```
setenv HP4GLPATH=/tmp/HP4GL
```

Displaying Environment Variables

To display the value of all of the environment variables, the user can type **env**. To just display the value of one environment variable, the **echo** command can be used.

For example, to display the contents of the environment variable **HP4PPATH**, the syntax is:

```
echo $HP4PPATH
```

Changing the Environment

On HP-UX, a change to an environment variable only affects the current environment or a **child environment**. It cannot affect the **parent environment.**

It is possible to suspend HP ALLBASE/4GL temporarily and start up a **sub-shell** by pressing `System Keys` `More Keys` `Op. System`. Because of the operation of HP-UX as described above, a change to an environment variable in the **sub-shell** will not affect the original HP ALLBASE/4GL environment; the change to the environment will be lost once the user ends the **sub-shell** and returns to HP ALLBASE/4GL.

Any environment variable's required by HP ALLBASE/4GL on HP-UX must be set before HP ALLBASE/4GL is run, preferably in the **hp4gl** shell script.

On MPE XL

On MPE XL, environment variables are sometimes called **system variables** or **command interpreter variables**. Those set by the user are not actually set by the system, so we should refer to them as environment variables.

Using the command interpreter **CI.PUB.SYS**, variables are set using the **setvar** command, as follows:

```
setvar VAR "value"
```

For example, to set **HP4SPATH** to **hp4s**, the syntax is:

```
setvar HP4SPATH "hp4s"
```

Displaying Environment Variables

To display the value of all of the environment variables, the user can type **showvar**. To just display the value of one environment variable, the **showvar** command is qualified by the name of the environment variable.

For example, to display the contents of the environment variable **HP4SPATH**, the syntax is:

```
showvar HP4SPATH
```

Changing the Environment

MPE XL environment variables are **global**, unlike those on HP-UX. Thus, it is possible for an change in a **sub-shell** to affect the **parent shell**.

This means that it is possible change the overall HP ALLBASE/4GL environment by a shell escape or by calling an external program. This can cause problems although there are some advantages. Later on we will see that HP ALLBASE/4GL uses file equations to specify report destinations. It is possible for an external program called by an HP ALLBASE/4GL application to change the setting of the file equation and thus re-direct the report according to the end-user's requirements.

OPSYS08 - SLIDE: Environment Variables on HP-UX I

HP ALLBASE/4GL OPERATING ENVIRONMENT

Environment Variables - HP-UX

The following environment variables are specific to, and used by HP ALLBASE/4GL on HP-UX:

HP4GLPATH - identifies the location of the HP ALLBASE/4GL directory.

HP4SPATH - identifies the directory containing the HP ALLBASE/4GL S-files.

HP4PPATH - identifies the directory containing the HP ALLBASE/4GL program files.

HP4DATAPATH - identifies the directory containing the HP ALLBASE/4GL data files.

HP4APATH - identifies the directory containing the HP ALLBASE/4GL auxiliary files.

HP4SQLPATH - identifies the HP-UX directory containing the DBEcon file used by applications that access HP ALLBASE/SQL.

HP4SCREEN_PRINT - the screen print control variable.

HP4LOCAL_PRINT - the local printer control variable.

HP4INV_PRT_CHAR - inverse space printing character.

HP4BG_PRI - controls priority of background processes.

HP4ISAM_BUFS - number of ISAM index node buffers.

Australian Software Operation
opsys08

HEWLETT PACKARD

Student Notes

Environment Variables on HP-UX I

When HP ALLBASE/4GL is installed, the installation script prompts the user for the name of a directory under which HP ALLBASE/4GL should be located. The installation script creates a directory called **HP4GL** in that directory and then installs HP ALLBASE/4GL underneath **HP4GL**.

For example, if the user specifies the directory **/developer**, HP ALLBASE/4GL will be installed under the directory **/developer/HP4GL**. The environment variable **HP4GLPATH** will be set in the **hp4gl** start-up script to reflect this.

All other HP ALLBASE/4GL environment variables are based upon **HP4GLPATH**. For example, **HP4SPATH** is defined as:

```
HP4SPATH=$HP4GLPATH/s
```

Remember that it is possible to change the location of the S-files, program files, data files and auxiliary files as long as the environment variables are set correctly to reflect the new locations.

Most of the environment variables listed on the slide have been covered previously and should be self-explanatory. **HP4SCREEN_PRINT** and **HP4LOCAL_PRINT** will be covered in more detail in later slides.

HP4INV_PRT_CHAR

HP4INV_PRT_CHAR determines the character which HP ALLBASE/4GL uses when representing a reverse-video space character in a screen print. It is normally the **DEL** character, which prints a **checkerboard** pattern on some printers.

HP4BG_PRI

HP4BG_PRI determines the priority at which background processes, initiated from HP ALLBASE/4GL, will run on the system. HP ALLBASE/4GL passes the current value of **HP4BG_PRI** through the HP-UX **nice** system call.

The format is **HP4BG_PRI=nn** where **nn** is a number from 0 to 19. Larger values of **nn** indicate a **lower priority** for the background process. A value of 0 indicates the same priority as the foreground process. If **HP4BG_PRI** is not set, HP ALLBASE/4GL assumes a priority of 11.

HP4ISAM_BUFS

The value of **HP4ISAM_BUFS** indicates the number of index node buffers used by the HP ALLBASE/4GL **ISAM** routines. The default value is 16, giving good performance in almost all applications.

If an application has many buffers open at the same time, increasing **HP4ISAM_BUFS** to 30 or 40 may increase performance.

Intentionally blank

OPSYS09 - SLIDE: Environment Variables on HP-UX II

HP ALLBASE/4GL OPERATING ENVIRONMENT

Environment Variables - HP-UX

HP ALLBASE/4GL also uses the value of some standard HP-UX variables:

PATH - locating external programs.
TERM - the terminal type.
TERMDATA - the directory containg the terminal database.
LANG - the user's language.
HPSQLPROClocation - location of the HP SQL process.

The following environment variables are used by HP ALLBASE/4GL on HP-UX:

HP4CART_TAPE - Cartridge device defined in "administ" system definition screen.
HP4MAG_TAPE - Mag Tape device defined in "administ" system definition screen.
HP4SYS_PRINT - '4GL system printer, defined in the "administ" system definition screen.
HP4TERM_ATTRS - Terminal attributes as defined in the terminal database entry for the user's terminal.
NOSKIP - turn off screen skip facility.

Australian Software Operation
opsys09

HEWLETT PACKARD

Student Notes

Environment Variables on HP-UX II

The LANG Environment Variable

By default, **LANG** is set to **american**; this allows 8-bit characters and supports the American collating sequence.

HP ALLBASE/4GL also supports 16-bit characters such as Chinese and Japanese; of course, **LANG** must be set to the appropriate value in each case.

Creating HP ALLBASE/SQL Databases

Unfortunately, HP ALLBASE/SQL does not use **LANG**; rather, the language of the database is specified in the **schema** file at the time of creation of the database. If no language is specified, the default is **n-computer.**

In this case, whenever HP ALLBASE/4GL connects to the database, the user will receive the message:

```
User language and Data language differ
```

The warning message only indicates that the **native-computer** collating sequence will be used rather than the American sequence. No corruption of the database will occur.

There are two ways to overcome this issue. The first is to modify the **hp4gl** script to change the default language to **n-computer.** This will mean that 8-bit characters can no longer be used. The second method is preferable; specify the correct language in the database schema before the database is created.

The schema file shown in the **HP ALLBASE/SQL Interface** module shows how to set the language of the database.

The NOSKIP Environment Variable

On HP-UX, HP ALLBASE/4GL provides the feature known as **screen skip**. This means that intermediate screens will not be displayed when the user enters a rapid sequence of characters to move from one screen to another.

This feature can be disabled by setting **NOSKIP** to any value, even blank, as follows:

```
NOSKIP=;export NOSKIP
```

Screen skip must be disabled if the user wishes to capture keystrokes, using the HP-UX utility **tee**, while running HP ALLBASE/4GL. If interested in further information, refer to the article by Simon Hiscox in ASO Support Newsletter #15 entitled **Automation of HP ALLBASE/4GL applications.**

The TERM Environment Variable

HP ALLBASE/4GL supports all HP terminals plus a number of non-HP terminals. The value of **TERM** must be set according to the terminal type.

For those terminals which don't support line drawing characters, HP ALLBASE/4GL uses standard **ASCII** characters.

For those terminals which don't support function keys labels, HP ALLBASE/4GL will display the function key labels over the bottom 2 lines of the screen, usually reserved for HP ALLBASE/4GL messages. In this case, the user can toggle the display between the function key labels and the message.

The generic setting for HP terminals is **hp**; a more intelligent setting which supports line drawing is **hp2392** and one which supports color and line drawing is **hp2397**.

Intentionally blank

OPSYS10 - SLIDE: Environment Variables on MPE XL I

HP ALLBASE/4GL OPERATING ENVIRONMENT

Environment Variables - MPE XL

The following XL variables are specific to HP ALLBASE/4GL:

HP4SPATH - identifies the group and account containing the S-files.
HP4SQLPATH - identifies the group containing the DBEcon file used by applications that access HP ALLBASE/SQL.
HP4DATAPATH - identifies the group and account where the KSAM and serial data files reside.
HP4APPNPATH - group and account containing unloaded application files.
HP4DBMPATH - group and account containing data base module files.
HP4FSPATH - group and account containing KSAM file structure files.
HP4SCREEN_PRINT - The screen print control variable.
HP4INV_PRT_CHAR - Inverse video print character.
HP4BG - Job logon command string for BACKGRND processes.
HP4DBMSIZE - File limit for data base module files.
HP4TERM - The terminal type.

Student Notes

Environment Variables on MPE XL I

Most of the environment variables are identical in purpose to those for HP-UX. However, there is no **HP4GLPATH** variable as HP ALLBASE/4GL is always installed in the same place.

HP4APPNPATH & HP4DBMPATH

Some new variables have been added however to cope with limitations of the MPE XL file system. On HP-UX, it is possible to include an extension (a . and a number of characters) at the end of a filename; this allows files with similar names to be stored in the same directory, distinguished by their extensions.

As MPE XL does not support this, separate groups must be used for files of the same name. Whereas on HP-UX, the unloaded application file (**.uld**) and the database module file (**.dbm**) are both stored in the S-file directory, on MPE XL those files are now stored in the **HP4APPN** and **HP4DBM** groups respectively.

The environment variables **HP4APPNPATH** and **HP4DBMPATH** point to those groups.

The HP4TERM Environment Variable

On MPE XL, HP ALLBASE/4GL only supports terminals compatible with the **HPTERM0** standard. This is not a problem as no other terminals are supported for MPE XL.

A typical setting would be **hp2392**, allowing line drawing characters. Setting it to hp causes HP ALLBASE/4GL to replace line drawing characters with standard **ASCII** characters.

The HP4BG Environment Variable

The **HP4BG** environment variable is also not found on HP-UX. On MPE XL, background processes are implemented as stream jobs and, as such, must supply a valid MPE XL logon command string; this is done using **HP4BG**.

A sample setting would be:

```
setvar HP4BG "hello james,manager/password.hp4gl/password"
```

OPSYS11 - SLIDE: Environment Variables on MPE XL II

HP ALLBASE/4GL OPERATING ENVIRONMENT

Environment Variables - MPE XL

The following XL variables are used by HP ALLBASE/4GL:

HPPATH - must be set to include PUB.SYS

The following Job Control Words are used by HP ALLBASE/4GL:

NLUSERLANG - selects appropriate message catalog.
NLDATALANG - selects collating sequence.

The following File Designators are used by HP ALLBASE/4GL:

Name	Default	Description
HP4KYIN	$STDIN	file to read keystrokes from.
HP4SCOUT	$STDLIST	file to write screen output to.
HP4KYOUT	none	keystroke log file.
HP4RFLOG	DEV=DISC;TEMP	Log file for KSAM reformat.
HP4GNERR	DEV=DISC;TEMP	Log file for 'generates'.
HP4TRACE	DEV=DISC;TEMP	Trace mode log file.
HP4REP	DEV=LP	HP4GL 'System Printer'.

Australian Software Operation
opsys11

HEWLETT PACKARD

Student Notes

Environment Variables on MPE XL II

Most of the variables shown on the slides should be self-explanatory.

HPPATH

MPE XL uses the **HPPATH** system variable to locate files; HP ALLBASE/4GL also uses it to locate external programs.

The group **PUB.SYS** must be included so that **HP4GLB** can find external binaries such as **HP4SP** and **HP4RP**. Without this, HP ALLBASE/4GL must be run with **PUB.SYS** as the current group.

If an end-user's application is to locate external programs in a group other than the current group, the name of that group must also be included in **HPPATH**. This could be done within the **HP4GL** command file.

Formal File Designators

HP ALLBASE/4GL also uses a number of **formal file designators** to allow re-direction of a number of input/output streams.

HPKYOUT

By setting **HPKYOUT**, it is possible to capture keystrokes into a log file while HP ALLBASE/4GL is being run. The **keystroke file** can later be used to supply input to HP ALLBASE/4GL so that the application can be run automatically.

HP4KYIN

By setting **HP4KYIN**, HP ALLBASE/4GL will read keystrokes from the file specified in the file equation rather than from the keyboard, enabling the application to be run automatically.

HP4SCOUT

By setting **HP4SCOUT**, screen output from HP ALLBASE/4GL will be captured to the file specified by the file equation. The contents could later be displayed using **FCOPY** or **PRINT** or some other utility.

For further information on this, refer to the article by Simon Hiscox entitled **Automating HP ALLBASE/4GL applications** in ASO Support Newsletter #15. The article describes two command files which implement capture of keystrokes and automatic playback.

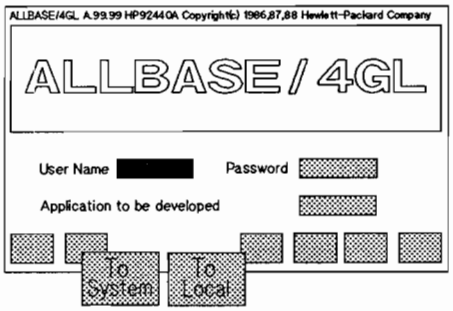
Intentionally blank

OPSYS12 - SLIDE: Environment Variables - HP4SCREEN_PRINT

HP ALLBASE/4GL OPERATING ENVIRONMENT

Environment Variables

HP4SCREEN_PRINT



control-P

HP ALLBASE/4GL screen image printing enables the user to print an image of the screen at any time, by using function keys:

* 'To Local' function key in the 'Screen Printing' set.
* 'To System' function key in the 'Screen Printing' set.
* Pressing control-P (+ CR on MPE XL).

Student Notes

Environment Variables - HP4SCREEN_PRINT

HP ALLBASE/4GL allows the user to print a screen image of the screen at any time.

There are two ways to do this, namely via the `Screen Printing` function key or by pressing **control-P**.

On MPE XL, it is necessary to press **control-P** followed by (Return).

The `Screen Printing` function key gives the user the choice of sending the report to the system printer or to a local printer, a printer connected directly to the user's terminal.

OPSYS13 - SLIDE: Environment Variables - HP4SCREEN_PRINT

HP ALLBASE/4GL OPERATING ENVIRONMENT

Environment Variables

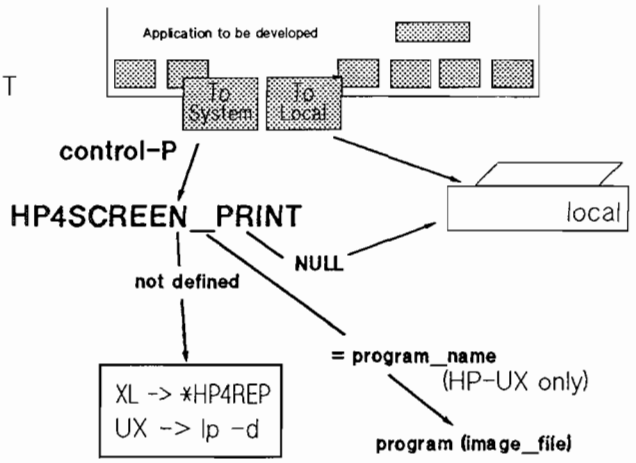
HP4SCREEN_PRINT variable

When the user presses the To System' or 'control-P' key, the HP4SCREEN_PRINT variable is used to determine the target printer.

If the variable is undefined, the screen image is sent to the system default printer.

If the variable is defined, and set to the NULL string, the target is a local printer.

If the variable is defined, with a non-NULL value, the string is the name of a program to call for printing ** - only on HP-UX: XL users; use FILE equations.



Student Notes

Environment Variables - HP4SCREEN_PRINT

The environment variable **HP4SCREEN_PRINT** is used to direct the output from a screen print operation.

Three conditions are placed upon **HP4SCREEN_PRINT**. If **HP4SCREEN** is:

1. **undefined**, the screen image is sent to the system printer.

2. **defined but NULL**, the screen image is sent to the local printer.

3. **defined and not NULL**, the string defines the name of a program to be called for printing. This is only available on HP-UX.

   This method is handy if you want to save the current image of the screen to a file. As the file is formatted for a line printer, you will need to edit it before you can use it for other purposes. The program specified by **HP4SCREEN_PRINT** could be a shell script which modifies the output before printing.

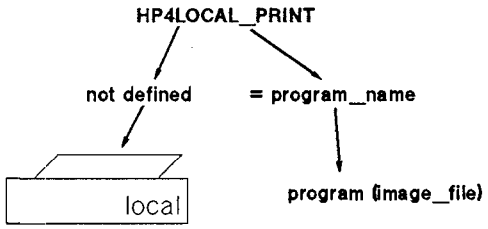
   To do this under MPE XL, a file equation should be used.

OPSYS14 - SLIDE: Environment Variables - HP4LOCAL_PRINT

HP ALLBASE/4GL OPERATING ENVIRONMENT

Environment Variables

HP4LOCAL_PRINT variable (HP-UX only)



A print job can be sent to the user's local printer from a number of different sources:

* Screen Printing.
* Report Printing.

By using the HP4LOCAL_PRINT environment variables, processing can be redirected to a chosen program - for special treatment.

Australian Software Operation
opsys14

HEWLETT PACKARD

Student Notes

Environment Variables - HP4LOCAL_PRINT

The environment variable **HP4LOCAL_PRINT** is used to direct screen images and reports to a local printer or to bypass the standard HP ALLBASE/4GL spooling system. It is only available on HP-UX.

Two conditions are placed upon **HP4LOCAL_PRINT**. If **HP4LOCAL_PRINT** is:

1. **undefined**, the screen image or report is sent to the local printer. The environment variable **HP4SCREEN_PRINT** must be set to **NULL**; it is is not set, any **HP4LOCAL_PRINT** setting is ignored.

2. **defined and not NULL**, the string defines the name of a program to be called for printing.

   The program specified by **HP4LOCAL_PRINT** could be a shell script which processes the output before printing.

If a non-**HPTERM0** terminal is used, **HP4LOCAL_PRINT** must be set if screen images or reports are to be sent to a local printer.

OPSYS15 - SLIDE: HP ALLBASE/4GL File System - HP-UX

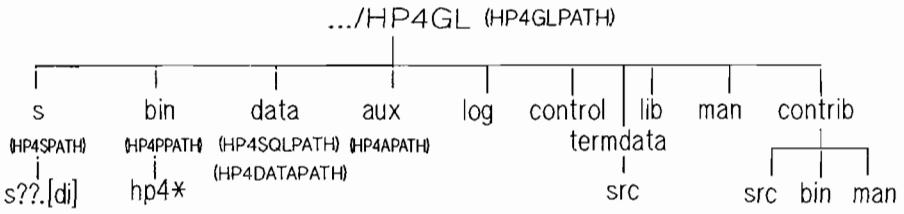
HP ALLBASE/4GL OPERATING ENVIRONMENT

HP ALLBASE/4GL File System

HP-UX

The HP ALLBASE/4GL installation procedure creates the following HP-UX directory structure.

The environment variables must be set to match the directories as shown:



Message Catalogs	Terminal Database	LP Spooler interface
/usr/lib/nls/*/HP4GL.???	/usr/lib/termdata/*/*	/usr/spool/lp/model/hp4_lp

Student Notes

The HP ALLBASE/4GL File System - HP-UX

The directories **s**, **bin**, **data** and **aux** have already been explained.

contrib

The **contrib** directory contains some unsupported utilities that a developer may wish to try. The contents of this directory must be manually extracted from the HP ALLBASE/4GL product tape by using the **tar** command.

The programs in the **contrib** directory are summarized below:

i_mgr

This utility displays the contents of an ISAM data file and lists the file index details.

bcheck

This utility can rebuild the index for an ISAM data file or check the integrity of an ISAM file index.

i_pack

This utility compacts an ISAM data file to recover disk space occupied by logically deleted records.

hp4xed

This utility allows the use of a standard HP-UX text editor to edit HP ALLBASE/4GL logic blocks. The program **hp4xed** runs **vi** using a temporary file which contains the logic blocks. Use **vi** to alter the logic blocks as required.

hp4remk

This utility rebuilds the indexes for the HP ALLBASE/4GL system files (S-files). Only one S-file can be rebuilt at a time.

hp4reod

This utility compacts HP ALLBASE/4GL S-files to recover space occupied by logically deleted records, and also rebuilds the S-file indexes.

WARNING

hp4remk should **always** be run **before hp4reod.** Both programs take as input, or as a command line argument, the number of the S-file to be rebuilt or hp4reoded. If **hp4reod** fails for any reason, which is of course unlikely, **hp4remk** must be run on that S-file again **before** attempting **another hp4reod.**

Make a backup copy of the S-files before running **hp4remk** and **hp4reod.**

---

hp4grep

This utility, like the HP-UX **grep** command, searches for a given pattern through HP ALLBASE/4GL logic blocks.

hp4ch_keys

HP ALLBASE/4GL B.01 changed the location of some function keys from the positions for HP TODAY A.02; `System Keys` has moved from (F7) to (F5), while `Help` has moved from (F5) to (F7). The utility **hp4ch_keys** will swap the function keys for a user's application, even in an HP ALLBASE/4GL runtime environment.

Running the Contributed Programs

These utilities must be run directly from HP-UX. Either the **HP4GL/contrib** directory must be in the user's HP-UX **PATH** or the utilities moved to a directory which is in their search path.

Contributed Programs Manual Pages

The **HP4GL/contrib/man** directory contains source files for HP-UX manual pages for some of these utilities. Please read the manual page carefully before attempting to use any of these utilities.

HP ALLBASE/4GL Message Files

The HP ALLBASE/4GL message catalog files are implemented as standard HP ALLBASE/4GL ISAM files called **HP4GL.dat, HP4GL.idx** and **HP4GL.str**. The record structure is as follows:

Field Description	Field Length (bytes)
message number	16
severity name	05
severity number	01
response length	02
default response	16
help name	16
text for line1	64
text for line2	64
class	01
operating system	02
Total:	187

Location

The message files are stored under the **/usr/lib/nls** directory. The actual directory is determined by the setting of the **LANG** environment variable. For example, if **LANG** is **spanish**, HP ALLBASE/4GL will look for the message catalogs in **/usr/lib/nls/spanish**.

This system allows localised message catalogs for different languages to be present on the one system.

Localisation

Most of the HP ALLBASE/4GL system messages are stored in the message catalogs, making them relatively easy to localise. The **ASO Support Group has an HP ALLBASE/4GL application** which can be used to translate the text of the messages; contact them for a copy.

Intentionally blank

OPSYS16 - SLIDE: HP ALLBASE/4GL File System - HP-UX

HP ALLBASE/4GL OPERATING ENVIRONMENT

HP ALLBASE/4GL File System

(HP-UX only)

$HP4GLPATH/control:

HP ALLBASE/4GL operator backup volume catalog
HP ALLBASE/4GL lp spooler system control files
ISAM data file logging system files

$HP4GLPATH/log:

ISAM data file logging files

$HP4GLPATH/lib:

ISAM libisam.a library file
install directory with many useful files

This directory must be a subdirectory of the HP4GL directory and its name cannot be changed.

Student Notes

The HP ALLBASE/4GL File System - HP-UX

control

Under the **control** directory are directories relating to **operator backups**, the **spooler control system** and the **data file logging system** files. If this directory is not present below the HP4GLPATH directory, HP ALLBASE/4GL will not be able to run.

log

The **log** directory contains files that are associated with the built-in **ISAM data-logging** which can be activated by the HP ALLBASE/4GL administrator.

lib

This contains programs that are run during the installation of the HP ALLBASE/4GL system, and may be re-run when the developer needs to move things around. For example, if a new printer is installed on the system after HP ALLBASE/4GL has been installed, the **pr_install** script should be re-run so that HP ALLBASE/4GL recognises the new printer.

It also contains the file **libisam.a** which can be used by 3GL programmers to access ISAM data files produced by HP ALLBASE/4GL.

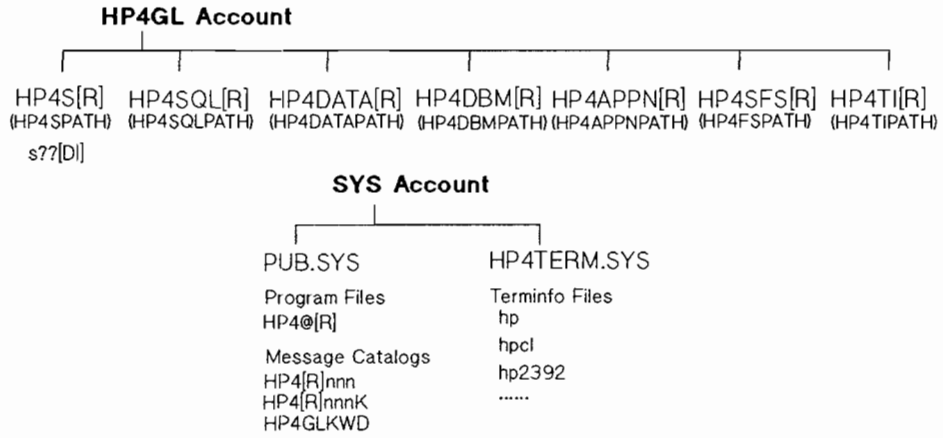
OPSYS17 - SLIDE: HP ALLBASE/4GL File System - MPE XL

HP ALLBASE/4GL OPERATING ENVIRONMENT

HP ALLBASE/4GL File System

MPE XL

The ALLBASE/4GL installation procedure creates the following account and group structure. The environment variables must be set to match accordingly.



Australian Software Operation
opsys17

HEWLETT PACKARD

Student Notes

The HP ALLBASE/4GL File System - MPE XL

The HP ALLBASE/4GL message catalog files are implemented as standard KSAM files. The record structure is identical to that for HP-UX. In fact, the files can be transported from HP-UX to MPE XL without difficulty; the process of converting an ISAM file to a KSAM file, or vice versa, is involved but not difficult.

The files are located in **PUB.SYS** and are called **HP4nnn** and **HP4nnnK**, where **nnn** represents the three digit number for the language. The language is determined by the setting of **NLUSERLANG**; **000** is for **NATIVE-3000**, **001** is for **AMERICAN**, etc. Refer to the **NLS Manual** for further details.

Thus, the message catalogs used by HP ALLBASE/4GL if the language was **AMERICAN** would be **HP4001.PUB.SYS** and **HP40001K.PUB.SYS**.

As for HP-UX, the **ASO Support Group** has an HP ALLBASE/4GL application which can be used to assist in the task of localising the message catalog.

OPSYS18 - SLIDE: Terminal Support

HP ALLBASE/4GL OPERATING ENVIRONMENT

HP ALLBASE/4GL File System

termdata directory / HPTERM group

The 'termdata' directory on HP-UX and the 'HP4TERM' group on MPE XL contain the terminal data base files. These files describe the capabilities and implementations of the different terminals used with HP ALLBASE/4GL.

HP ALLBASE/4GL on HP-UX will work with most intelligent terminals. The MPE XL version requires TERM0 compatibility (2622 and up).

HP ALLBASE/4GL on all operating systems provides full support for HP terminal highlighting, color, line drawing, and other HP features.

Termdata entry source and the hptic and hpuntic programs are provided for creating termdata entries for unsupported terminals. **

Student Notes

Terminal Support

On HP-UX, the TERM Environment Variable

On HP-UX, HP ALLBASE/4GL supports all HP terminals plus a number of non-HP terminals. The value of **TERM** must be set according to the terminal type, otherwise **hp2622** is assumed.

For those terminals which don't support function keys labels, HP ALLBASE/4GL will display the function key labels over the bottom 2 lines of the screen, usually reserved for HP ALLBASE/4GL messages. In this case, the user can toggle the display between the function key labels and the message.

On MPE XL, the HP4TERM Environment Variable

On MPE XL, HP ALLBASE/4GL only supports terminals compatible with the **HPTERM0** standard. This is not a problem as no other terminals are supported for MPE XL. The **HP4TERM** variable must be set to the appropriate terminal type, otherwise **hp2622** is assumed.

The generic setting for HP terminals is **hp**; a more intelligent setting which supports line drawing is **hp2392** and one which supports color and line drawing is **hp2397.**

For those terminals which don't support line drawing characters, HP ALLBASE/4GL uses standard **ASCII** characters.

hptic, the Terminfo Compiler

On HP-UX, HP ALLBASE/4GL provides the source files for the **terminal definition files** and provides the utilities **hptic** and **hpuntic**, compatible with the standard HP-UX **tic**(1) and **untic**(1) commands. However, **hptic** and **hpuntic** support some extra fields in the terminal description to allow for color, function keys, labels, etc..

The user can modify the source files to change the behaviour of a certain terminal with HP ALLBASE/4GL, or to produce a terminal file for a currently unsupported terminal. The utility **hptic** will compile the source file into a runtime file; the utility **hpuntic** can produce a source file from a runtime file.

The source files are stored under the **HP4GL/termdata/src** directory; **hptic** creates files under the directories **1**, **2**, **3**, **4**, **7**, **9**, **f**, **h**, **v** and **w**. Why so many directories? The purpose is to allow **TERM** to be set in a number of different ways.

For example, the **HP 150** can be found under **1** as **150** and under **hp** as **hp150**. Similarly, the **HP 2392** can be found under **2** as **2392** and under **hp** as **hp2392**. **7** is for the **700-Series** terminals such as the **700-41**; **v** is for the **DEC** terminals such as the **vt220**; **w** is for the **Wyse** terminals such as the **wy99**.

Refer to the manual pages supplied for **hptic** and **hpuntic** as well as **Appendix D of the Developer Reference Manual.**

Intentionally blank

OPSYS19 - SLIDE: The HP ALLBASE/4GL Startup Scripts

HP ALLBASE/4GL OPERATING ENVIRONMENT

HP ALLBASE/4GL Startup Scripts

```
#!/bin/sh
# HP ALLBASE/4GL Startup Script.
HP4GLPATH=/extra/HP4GL
HP4SPATH=$HP4GLPATH/s
HP4PPATH=$HP4GLPATH/bin
HP4DATAPATH=$HOME/data
HP4SQLPATH=$HOME/DB
TERM=2397-t
export HP4GLPATH HP4SPATH HP4PATH
export HP4DATAPATH HP4SQLPATH TERM
exec $HP4PPATH/hp4glb
```

```
comment HP ALLBASE/4GL Startup Script
setvar HP4SPATH "s.hp4gl"
setvar HP4DATAPATH "ksam.lizard"
setvar HP4SQLPATH "db.lizard"
setvar HP4TERM "hp150-l"
setvar HP4SCREEN_PRINT ""
NLUSERLANG=001
NLDATALANG=001
run hp4glb.pub.sys
```

To setup the environment for HP ALLBASE/4GL, we usually use a startup script. Useful default/example scripts are provided with the system:

hp4gl.pub.sys	- MPE/XL startup script.
.../HP4GL/hp4gl	- HP-UX startup script.
.../HP4GL/wmhp4gl	- HP-UX startup script for HPwindows.
.../HP4GL/x11hp4gl	- HP-UX startup script for X-windows.

Student Notes

The HP ALLBASE/4GL Startup Scripts

The **hp4gl** script on HP-UX and the **HP4GL** command file on MPE XL provide the user with a standard environment for running HP ALLBASE/4GL. They can be easily modified to suit the user's needs.

Command Line Parameters

Both the HP-UX and MPE XL startup scripts will accept command line parameters, allowing them to be **wrapped up** in other scripts or command files.

For example, consider the situation where a developer wishes to allow end-users of an application called **demo** to run the application with just one command; the command is to be **demo** and the end-user name is also **demo.** Programmers often must make things as easy as possible for end-users!

On HP-UX, create a script called **demo**, as shown below:

```
#
#               Name:           demo
#               Author:         Fred
#               Date:           6/2/1990

hp4gl -u demo -a demo
```

On MPE XL, create a command file called **demo**, as shown below:

```
comment
comment         Name:           demo
comment         Author:         Fred
comment         Date:           6/2/1990
comment

hp4gl "-u demo -a demo"
```

Notice the liberal use of comments; go and do likewise!

Further Command Line Parameters on MPE XL

The MPE XL version also allows the end-user to override the default settings for any of the environment variables. For example, to run HP ALLBASE/4GL using an alternate set of S-files in the group **NEWHP4S**, the syntax is:

```
hp4gl hp4spath="newhp4s"
```

This cannot be done with the HP-UX version but only a small amount of shell programming would be required to supply that feature.

The HP-UX Start-up Script

The HP-UX version of **hp4gl** uses the HP-UX command **exec** to start the HP ALLBASE/4GL binaries, as shown below:

```
#
if [ -f $HP4PPATH/hp4glb ]
then
 exec $HP4PPATH/hp4glb $*

elif [ -f $HP4PPATH/hp4glbr ]
then
      exec $HP4PPATH/hp4glbr $*

else
      echo Cannot find an ALLBASE/4GL binary. 1>&2
fi
```

The purpose of **exec** is to cause the new process to overlay the old process. The **hp4gl** script will be assigned a **process id**, say **17193**. When the **hp4glb** binary is initiated, it is forced **on top of** the original **hp4gl**, thus it will run with **pid=17193**, saving a process on the system.

Environment Variables

Note

Instructor:

If on HP-UX, copy the S-files in **HP4GL/s** to a new directory called **HP4GL/s_new.** On MPE XL, copy the **S-files** to a new group called **HP4SNEW**; use the **HP4SCOPY** utility.

If on HP-UX, copy the HP ALLBASE/4GL message catalog files from the directory **/usr/lib/nls/american** to **/usr/lib/nls/spanish.**

For the following situations, write down the environment variable(s) that you would set to achieve the desired result.

Questions

1. Your system administrator has moved your HP ALLBASE/SQL database directory from **/extra/tempDB** to **/mnt/DATABASE.** Which variable must be set so that the new database can be found?

   Answer: **HP4SQLPATH**

2. You have a **vt220** terminal with an attached printer. You wish to print screen images on a laserjet on your desk. The laserjet is configured into the host operating system. Which variable or variables need to be set?

   Answer:

   HP4SCREEN_PRINT must be set to **NULL** to force the screen images to go to the local printer; **HP4LOCAL_PRINT** must be set to **NULL** to allow local printing with a **non-HPTERM0** terminal.

3. Now you want to print reports that are targeted for the **L** printer on that same laserjet. Which variable must be set?

Answer: **HP4LOCAL_PRINT**

4. On MPE-XL, you wish to run **BACKGRND** processes from your application. Which environment variable must be set first?

   Answer: **HP4BG**

5. On HP-UX, you wish to lower the priority given to **BACKGRND** processes by the operating system. How can you do this?

   Answer: **HP4BG_PRI**

6. I have just taken delivery of a new HP 2397 color terminal, but I see no color when I run HP ALLBASE/4GL. Which environment variable must be reset on HP-UX and MPE XL?

   Answer: **TERM** on HP-UX, **HP4TERM** on MPE XL.

Startup Scripts

The standard HP ALLBASE/4GL startup scripts are satisfactory for most users. However, it is a fairly simple task to customise or create your own startup script.

Using a text browser (**print** on MPE XL or **more** on HP-UX should be fine) or an editor, take a look at the startup script you have been using for the class so far. Your instructor will tell you where to find the script. Usually on HP-UX it will be **/usr/local/bin/hp4gl**; on MPE XL it will be **hp4gl.pub.sys** or perhaps **hp4gl.pub.hp4demo**.

You will notice that the script basically sets a number of environment variables before it starts up the **hp4glb** main binary.

1. Which variable indicates to HP ALLBASE/4GL where to look for the S-files?

   Answer: **HP4SPATH**

2. Which variable indicates the location of the HP ALLBASE/4GL program files?

   Answer: **HP4GLPATH**

3. HP-UX users only: Why does the script use the **exec** command to start the **hp4glb** program? What does this save?

   Answer:

   The **exec** command causes the **hp4glb** binary to overlay the shell used to run the **hp4gl** script. This reduces the number of processes involved in running HP ALLBASE/4GL.

4. Which variables do you think must be set as an absolute minimum before HP ALLBASE/4GL will operate?

   Answer:

   The minimum requirement is **HP4GLPATH** and **HP4SPATH**, although it is not entirely satisfactory. If the current directory contains the HP ALLBASE/4GL binaries, everything will work correctly; if not, you must specify where to find the main **hp4glb** binary but the screen painter, generates, report painter, unload and load will not work.

5. Copy the standard script, giving it your name, and modify the copy so that it points to a different set of S-files in the directory **s_new** or the group **HP4SNEW**. Try running it.

6. If running on HP-UX, modify your new script so that it sets the **LANG** environment variable on HP-UX to **spanish**. Check that HP ALLBASE/4GL runs correctly. Inspect the contents of the directory **/usr/lib/nls/spanish** using **ls -l**.

7. Try creating your own startup script that sets this minimal set of variables, before starting the **hp4glb** binary? (Only try this if you have time.)

   Does it work? Of course it works!

   Do all your applications work? Yes!

Screen Printing

In this module we learned about the HP ALLBASE/4GL screen printing facility. This can be useful for getting a hard copy of one of your application screens - or for getting a hard copy of a screen that displays an error (ask your customers to FAX you screen prints!).

One common reason for **dumping** the screen is for documenting an application. You can print out the screen, then paste it into a reference manual page. Another way to get a screen image into a document is to take a fully **electronic** approach.

We can use the **HP4SCREEN_PRINT** variable to control what happens when we press **control-P** or `To System` in HP ALLBASE/4GL. In this lab we will set **HP4SCREEN_PRINT** so that we dump screen images into a file called **screen_print**.

If you are in HP ALLBASE/4GL, leave and return to the command line. Now we set the **HP4SCREEN_PRINT** variable so that image is dumped into a file; use the following command:

HP-UX sh, ksh:

```
HP4SCREEN_PRINT="cat > screen_print"; export HP4SCREEN_PRINT
```

HP-UX csh:

```
setenv HP4SCREEN_PRINT="cat > screen_print"
```

MPE/XL:

```
FILE HP4REP=SCRNIMG;DEV=DISC
```

Now start up HP ALLBASE/4GL, and use the **control-P** function to dump the Sign-On screen. Exit the Sign-On screen, and examine the screen print file (screen_print on HP-UX, SRNIMG on MPE XL).

The HP ALLBASE/4GL Program Files: Summary

Program filenames all start with **hp4**, and on developer systems have no more than 7 letters in the name.

The program files can be classified as shown below. The **S** against some programs indicate that they can be used stand alone.

General Running

S	hp4glb	Main binary
	hp4gen	Generates
	hp4sort	Sort program
	hp4sp	Screen painter
	hp4rp	Report painter
	hp4drpt	Developer reports
	hp4arpt	Administrator reports

Utilities

	hp4futl	Data file utilities
S	hp4uld	Unload application(s)
S	hp4ld	Application load
S	hp4stoa	S-files to ASF
S	hp4atos	ASF to S-file
S	hptic	HP terminfo compiler
S	hpuntic	HP terminfo un-compiler

Maintenance

S	hp4remk	Remake S-files (pack)
S	hp4reod	Reorder S-files (index)
	hp4chkap	Check application for corruptions
	hp4fixap	Fix corruptions in application
S	upgrade	Upgrade to this revision

HP-UX Operator Facility

hp4bkup	Operator backup
hp4opr	Operator functions
hp4oprk	Operator kill process
hp4oprs	Operator process status
hp4pctl	Printer control
hp4penbl	Printer enable
hp4pstat	Printer status
hp4rec	Recovery from backup
hp4rprst	Report restart
hp4rptx	Report cancel
hp4rpupd	Report update

Runtime system program files have the same name with an **r** appended. This allows MPE/XL runtime and developer program files to co-exist in the PUB.SYS group.

Module: The Outside World

Instructor Notes

OUT00 - SLIDE: Module Objectives

THE OUTSIDE WORLD

EXTERNAL Command
Passing Parameters
Returning Parameters
Utilities

OBJECTIVES:

Upon completion of this module, the student will be able to:

1. Explain the purpose and usage of the EXTERNAL command.
2. Describe how to pass parameters between HP ALLBASE/4GL and an external program.
3. Use the EXTERNAL command to communicate with an external program.
4. List and describe some of the utilities provided with the HP ALLBASE/4GL system.
5. Use some of the external HP ALLBASE/4GL utilities.

Student Notes

Module Objective

This module will introduce the students to some of the HP ALLBASE/4GL external utilities and will demonstrate communication with external programs using the EXTERNAL command.

Student Objectives

At the end of this module, each student will be able to:

- Explain the purpose and usage of the EXTERNAL command.
- Describe how to pass parameters between HP ALLBASE/4GL and an external program.
- Use the EXTERNAL command to communicate with an external program.
- List and describe the external utilities provided with the HP ALLBASE/4GL system.
- Use some of the external HP ALLBASE/4GL utilities.

Resource Allocation

Equipment

The following items are required for this module:

- Instructor notes.
- Student notes.
- Slides.
- A laboratory worksheet.
- Access to an HP 3000 Series 900 machine or an HP 9000 Series 800 or HP 9000 Series 300 machine with HP ALLBASE/4GL installed.
- The complete set of HP ALLBASE/4GL manuals.

Time

This module should take approximately 3 hours, with 2 hours for theory and 1 hour for the 2 laboratory sessions. Each laboratory session should take approximately 30 minutes.

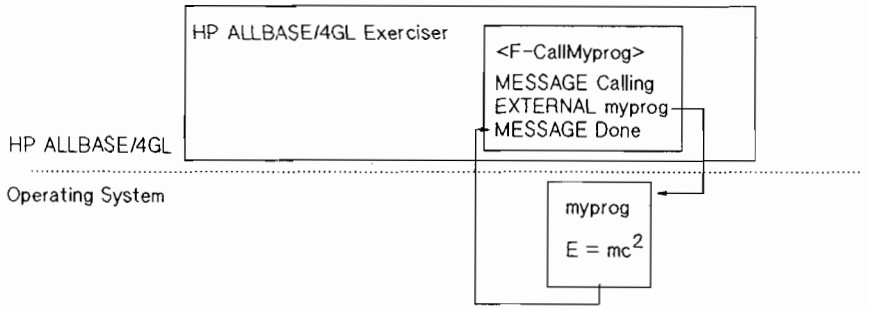
Pacing

Read through the slides reasonably quickly.

OUT01 - SLIDE: The EXTERNAL Command

EXTERNAL COMMAND

Description



The EXTERNAL command executes an external program which is written in a language other than HP ALLBASE/4GL.

Information can be passed to the external program as arguments, and can be returned into HP ALLBASE/4GL data objects.

Student Notes

The EXTERNAL Command

HP ALLBASE/4GL is an excellent programming system for certain types of applications, namely **information management** or **on-line transaction processing**. It is not well suited to developing applications to read scientific instruments in real-time, plot graphical data or play computer games; it was simply not designed for those tasks.

To cover its deficiencies, HP ALLBASE/4GL must have a mechanism whereby it can communicate with the **outside world**, where it can **talk** to programs which were designed to do those **other things**. This ability is provided with the **EXTERNAL** logic command.

Another Language

An HP ALLBASE/4GL application can execute an external program written in another language. It simply passes control to the operating system, which then executes the program name given to it as an argument. The external program may be written in **C**, **Pascal**, **COBOL**, **FORTRAN** or in the **command interpreter shell**, **/bin/sh** on HP-UX or **CI.PUB.SYS** on MPE XL. In fact, the program can be written in any language executable by the operating system.

How Are Externals Found?

To find the external program, HP ALLBASE/4GL looks in the user's **search path**. On HP-UX, all the listed directories in the user's **PATH** environment variable will be searched. On MPE-XL, the **HPPATH** variable will be used for the same purpose.

The 'aux' directory

On HP-UX, it is recommended that external programs reside in the **aux** directory, although there is nothing hard-coded in HP ALLBASE/4GL to make it look in the **aux** directory. The developer must include the **aux** directory into the **PATH** environment variable to ensure that HP ALLBASE/4GL will look in that directory.

The 'HP4AUX' group?

On MPE-XL, it is recommended that external programs reside in a group, created by the user, within the same account as the other files used by HP ALLBASE/4GL. A good suggestion would be a group called **HP4AUX**, clearly indicating that the programs belong to HP ALLBASE/4GL. The external program could also reside in **PUB.SYS**, where it will certainly be found.

Good housekeeping

Why is it recommended that external programs reside in the **aux** directory or the **HP4AUX** group? The answer is: **For simplicity, convenience and good house-keeping!** Tracking down problems in an end-user's environment can be night-marish if the tracker, probably you, is forced to look throughout the file-system for that elusive external binary! Also, there is less likelihood that an over-zealous system administrator will remove those **unrelated binaries** cluttering his already overfull **/usr/bin** or **PUB.SYS**. Transportation of the entire system is much easier also if it is known that all external programs reside in just one location.

There may be cases where the external binary simply must reside somewhere else. On HP-UX, create a **link** or **symbolic link** from that binary into the **aux** directory.

Another Language? Calling another HP ALLBASE/4GL application.

It is also possible to run another HP ALLBASE/4GL application from within an HP ALLBASE/4GL application. Is this running another language or the same language? Discuss this with the class. More on this later.

Why would you want to do that? Calling another HP ALLBASE/4GL application opens up the possibility of breaking up a large project into a number of smaller applications, each self-contained. This technique provides a simple way to overcome some of the lower HP ALLBASE/4GL limits, such as 255 variables

Another reason is **integration**. Just as an HP ALLBASE/4GL application could be used to **front-end** a number of programs written in other languages, tying them into one menu structure, so a number of HP ALLBASE/4GL applications could be tied into one menu structure, providing a simple way for an end-user to access a suite of programs.

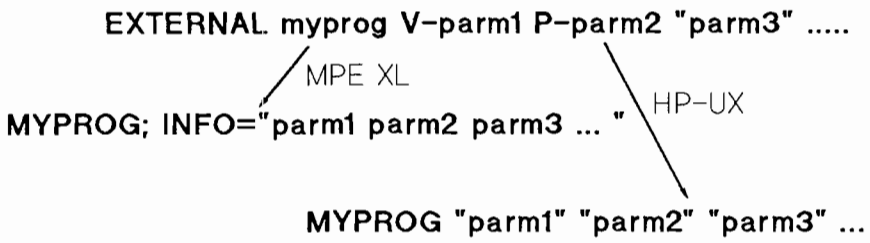
Integration of the programs is nearly seemless. When using logon-bypass, the only clues that another copy of HP ALLBASE/4GL is running are given by the **Please wait** message, the **copyright** message and the **End of program** message.

Intentionally blank

OUT02 - SLIDE: The EXTERNAL Command - Passing Parameters

EXTERNAL COMMAND

Passing Parameters



Parameters can be passed to the external program by specifying their source in the EXTERNAL command.

* The external program should be written to expect the exact number of parameters specified.
* For MPE XL the total length of the INFO string cannot exceed 256 characters.

Student Notes

The EXTERNAL Command - Passing Parameters

An external program is called from an HP ALLBASE/4GL application by using the **EXTERNAL** logic command. The first argument to the command, (ignoring the ***REFRESH** option), must be the name of the external program. This can be specified by a literal or by some HP ALLBASE/4GL item, such as a **variable.**

Note

If the name of the external program contains a dot or other special characters, a literal cannot be used.

The **EXTERNAL** command then accepts any number of arguments; on HP-UX, the only limit is on the length of the command line within the HP ALLBASE/4GL logic block. Even then, there are ways to allow more arguments, using the **LINK** command. On MPE XL, the command line is limited to 256 characters.

The arguments can also be literals or HP ALLBASE/4GL items such as **variables, scratch pads, communication fields,** etc.

Parameter Format

The format of the arguments to the **EXTERNAL** command must agree with the parameter format expected by the external program. Conversely, the external program must expect parameters in the same format as provided by the HP ALLBASE/4GL application. Which program takes precedence, the external program or the HP ALLBASE/4GL application, is a matter to be decided by the developer; this is especially true if the developer also wrote the external program. Often the HP ALLBASE/4GL application should defer to the requirements of the external program as it is easier to change the HP ALLBASE/4GL application, certainly if the external program is written in a compiled language.

If the external program was not written by the developer, it may be impossible to change the expected parameter format, so the HP ALLBASE/4GL application must fit in with the external program.

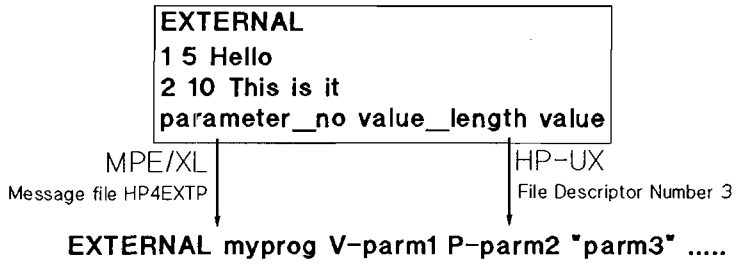
Note

This slide may give the impression that a program cannot be used unless it was specifically written to work with HP ALLBASE/4GL; this is not so.

OUT03 - SLIDE: The EXTERNAL Command - Returning Parameters I

EXTERNAL COMMAND

Returning Parameters



Parameters can be returned to HP ALLBASE/4GL by writing the defined parameter packet to the return path set up for your OS.

* For MPE XL, the return packet must be written to a message file "HP4EXTP".
* For HP-UX, the return packet must be written to file descriptor number 3.

Student Notes

The EXTERNAL Command - Returning Parameters I

As well as passing information to an external program, an HP ALLBASE/4GL application can receive information from the external program.

Output Format

HP ALLBASE/4GL expects the return information to be in a special format, as follows:

- The word **EXTERNAL** must be on the first line.
- On each subsequent line, there must be a number specifying which parameter follows, a space, the length of the data that will be returned for that parameter, another space, and finally the actual data.

Note

The parameter number is taken from the position **after** the program name.

The external program can send information back to any **writeable** HP ALLBASE/4GL item included in the argument list for the **EXTERNAL** command. Obviously, information cannot be returned to constants, literals or read-only communication area fields in HP ALLBASE/4GL.

Furthermore, the external program can write to the HP ALLBASE/4GL items in any order, by specifying the **parameter number**.

This format does not rely upon **delimiters** such as " or '; these characters can be included in the information returned to HP ALLBASE/4GL, as can **spaces**. Also, leading and trailing spaces are preserved in the returned information.

In general, the format is:

```
EXTERNAL
parameter_no value_length value
```

The Example Given

In the example on the slide, information is being passed back into parameters number 1 and 2. There is no need for the parameters to be in any particular order; the example could equally well have had parameter 5 then parameter 2.

For **parameter 1**, the **length** is **5** as the string is **Hello.** For **parameter 2**, the **length** is **10** as the string is **This is it.**

Return Path on HP-UX:

For HP-UX, the external program must write to **file descriptor 3.**

This is a very simple task for a shell script or **C-program**; it is not so easy for a **Pascal** program or using another language.

Shell Script

The following code fragment illustrates how to return information from an HP-UX shell script.

```
echo "EXTERNAL" > &3
echo 1 5 Hello > &3
echo 2 10 This is it > &3
```

The **echo** command normally sends the information to the screen but **&3** re-directs it to **file-descriptor 3.**

C-Program

The following code fragment illustrates how to return information from an HP-UX C-program.

```
fptr = fdopen(3, "w");
fprintf(fptr, "EXTERNAL");
fprintf(fptr, "1 %d %s", strlen(str1), str1);
fprintf(fptr, "2 %d %s", strlen(str2), str2);
```

where **str1** and **str2** are variables of type **char**, containing **Hello** and **This is it** respectively.

fptr is a **file pointer**; it has opened **file descriptor 3** with **write access.**

Return Path on MPE XL:

For MPE XL, the external program must write to the file **HP4EXTP**; this must be a variable length record binary message file.

This is not straight-forward for any programming language on MPE XL.

C-Program

The following code fragment illustrates how to return information from an MPE XL C-program.

```
sprintf(fname, "&HP4EXTP&");
HPFOPEN(8, &filenum, &status, 2, fname, 3, &domain, 11, &access);
sprintf(fname, "EXTERNAL!");
FWRITE((short)filenum, (char ◊)fname, (short)-9, CONTROLCODE);
FWRITE((short)filenum, (char ◊)str1, (short)-strlen(str1), CONTROLCODE);
FWRITE((short)filenum, (char ◊)str2, (short)-strlen(str2), CONTROLCODE);
```

fname is a character array of length 128. **filenum** and **status** are integers. **domain** is an integer with value **2**, **access** is an integer with value **1**.

str1 and **str2** are character arrays containing **Hello** and **This is it** respectively.

Some error checking code has been eliminated for simplicity.

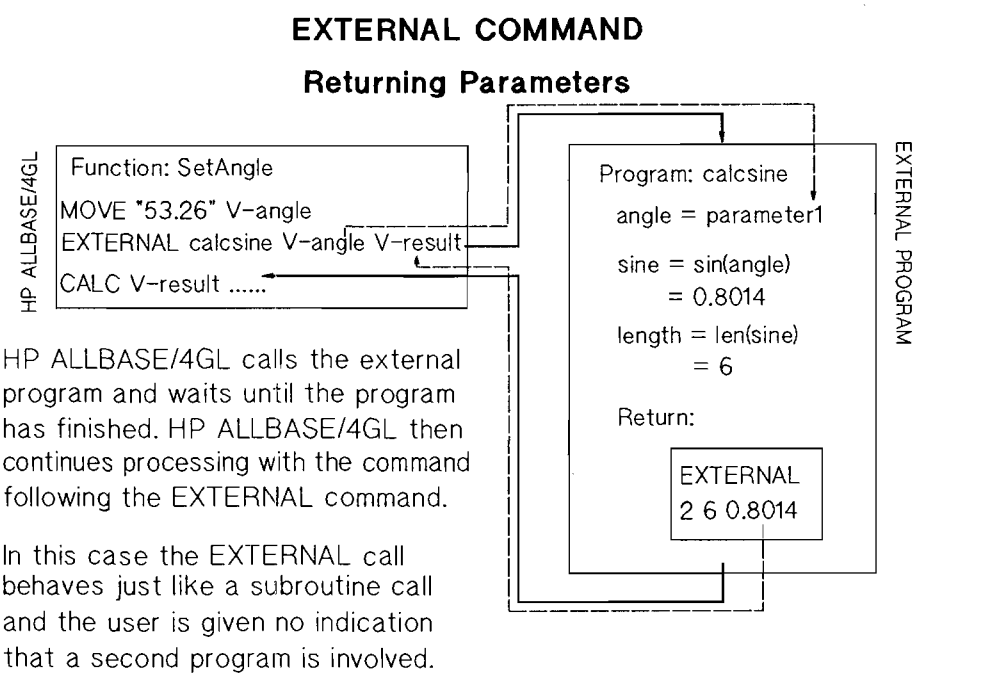
References

Refer the students to the page in this module entitled **Sample Code Fragments** for the examples shown above. The students should also read the Developer Reference Manual entries for the **EXTERNAL** command very thoroughly.

Further information can be obtained from some **ASO Support Newsletter** articles. In particular, Simon Hiscox wrote an article describing how to return parameters from a **Pascal** program running on an HP 9000 Series 300 computer under HP-UX; it is in Newsletter #11. An article, in Newsletter #3, by David Williams also discusses parameter passing with external programs.

Intentionally blank

OUT04 - SLIDE: The EXTERNAL Command - Returning Parameters II



HP ALLBASE/4GL calls the external program and waits until the program has finished. HP ALLBASE/4GL then continues processing with the command following the EXTERNAL command.

In this case the EXTERNAL call behaves just like a subroutine call and the user is given no indication that a second program is involved.

Student Notes

The EXTERNAL Command - Returning Parameters II

The Example

The example on the slide shows how information can be passed to and from an HP ALLBASE/4GL function; this example is HP-UX based.

One parameter, **V-angle**, is being used to pass data **TO** the external program and the other parameter, **V-result**, is being used to read data **FROM** the external program.

As can be seen, the output from the external program is:

```
EXTERNAL
2 6 0.8014
```

This means that the data **0.8614** will be returned into the **V-result** variable.

The HP ALLBASE/4GL function will suspend operation while the external call is being made, and continue when the external program is complete.

Actually, the HP ALLBASE/4GL function will only continue when **file descriptor 3** is closed.

OUT05 - SLIDE: The EXTERNAL Command - the *REFRESH option I

EXTERNAL COMMAND

***REFRESH Option**

Some programs called by the EXTERNAL command use or affect the current display. The EXTERNAL *REFRESH option should be used where external programs fall into this class.

When the EXTERNAL *REFRESH option is used:

1. The terminal interface is returned to the way it was before entering HP ALLBASE/4GL.
2. A message is printed to indicate the external call.
3. The external program is invoked.
4. Once the program is complete, the terminal interface is returned to HP ALLBASE/4GL mode, and the current HP ALLBASE/4GL screen is repainted.

Parameters can be passed and returned without *REFRESH.

Student Notes

The EXTERNAL Command - the *REFRESH Option I

Terminal Modes

During normal operation, HP ALLBASE/4GL sets the terminal into **raw mode**; normally it is in **formatted mode.**

Many external programs require **formatted mode**, so HP ALLBASE/4GL provides the ***REFRESH** option to reset the terminal back to **formatted mode** while the external program is running. The ***REFRESH** option also ensures that the terminal is set back to **raw mode** when the external program terminates and HP ALLBASE/4GL resumes.

Parameters can still be passed and returned in the normal fashion, despite the presence of the ***REFRESH** option.

The Please Wait Message

When calling an external program with the ***REFRESH** option, HP ALLBASE/4GL issues the message **Please wait** until the external program takes control.

ASIDE: Termination of HP ALLBASE/4GL

HP ALLBASE/4GL also resets the terminal to **formatted mode** when it exits in the normal fashion. If an HP ALLBASE/4GL process is terminated abnormally, the screen will not be reset correctly and the terminal will appear to be **locked.**

Correct Termination of HP ALLBASE/4GL on HP-UX

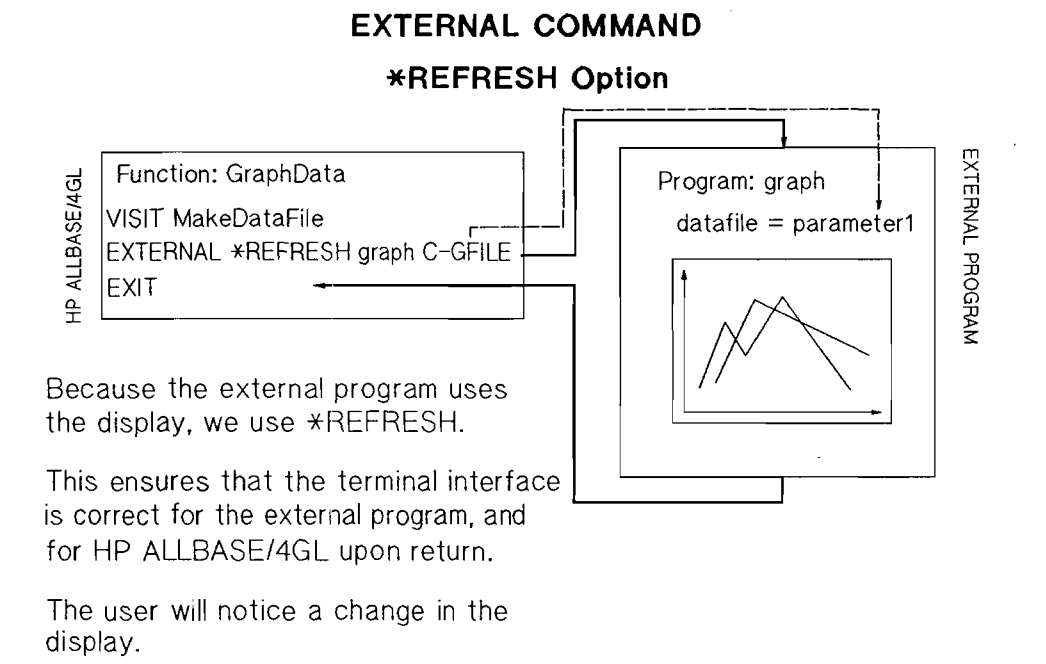
If it is ever necessary to terminate an HP ALLBASE/4GL session on HP-UX, **signal 15** should be sent; only rarely should it be necessary to use **signal 9.**

By sending **signal 15**, HP ALLBASE/4GL is given time to release all database locks, close all files, kill off any child processes and reset the terminal. Sending **signal 9** causes HP ALLBASE/4GL to die immediately; it could leave the database locked and the terminal will certainly require resetting.

To **kill** the HP ALLBASE/4GL session, first find and note the **process id** or **pid** for the **hp4glb** process. Then note the **pid** for each of the child processes belonging to the parent **hp4glb**. Then type:

```
kill -15 <pid> <pid> <pid> ...
```

OUT06 - SLIDE: The EXTERNAL Command - the *REFRESH option II



Australian Software Operation
out06

HEWLETT PACKARD

Student Notes

The EXTERNAL Command - the *REFRESH Option II

The example on the slide shows HP ALLBASE/4GL calling an external graphing program, which takes over the screen and displays a graph. In this case, HP ALLBASE/4GL is being run on an HP graphics terminal such as an **hp2627**.

Before the graphics program can display the bar charts, the screen must be restored to the standard operating mode. When the graphics program has finished, the screen must be restored to the state that it was in before the graphics program was called.

To do this, the external program has been called using the ***REFRESH** option to the **EXTERNAL** command. The ***REFRESH** option ensures that the screen state is saved before the external command is executed, and that it is restored later.

As mentioned before, the terminal is in a **raw** mode when HP ALLBASE/4GL is executing, but is reset to **formatted** mode for the duration of the external program's execution. HP ALLBASE/4GL also resets the terminal to **formatted** mode when it exits in the normal fashion. This is why the terminal setup may be strange if, for example, someone terminates an HP ALLBASE/4GL process.

In the example, the information to be displayed is not passed to the external program in the calling parameters. Instead, the name of a serial file is passed, and the information to be graphed is contained in that serial file.

OUT07 - SLIDE: The EXTERNAL Command - Background Processes

EXTERNAL COMMAND

Backgroung Processes

(HP-UX Only)

```
#!/bin/sh
# Start background process.
bgprocess 3>&- &
```

```
/*
 * Start background process.
 */
main()
{
    close(3);
    fork();
}
```

If your external program initiates a background process, you must close file descriptor 3 before initiating the background process.

If file descriptor 3 is left open, control will not return to HP ALLBASE/4GL until the background process terminates.

Student Notes

The EXTERNAL Command - Background Processes

Be very careful if an external program initiated by HP ALLBASE/4GL starts a background process. **file descriptor 3** must be closed before the background process is started; otherwise, control will not return to HP ALLBASE/4GL until the background process terminates.

The slide contains code fragments for **shell script** and for **C-code**, illustrating how **file descriptor 3** can be closed.

Shell Script

Line 1 of the shell script ensures that the script is being run by **/bin/sh**, not /bin/ksh or **/bin/csh**.

Line 3 starts the background process called **bgprocess**, putting it in the background.

The statement **bgprocess &** would be sufficient to run the process in the background; the extra construct

```
3>&-
```

forces **file descriptor 3** to be closed before the process is started.

C-Code

The statement **close(3)** closes **file descriptor 3**. The **fork();** statement initiates the new process. Thus, **file descriptor 3** is closed before the background process is started.

Laboratory Exercise

The students should now do the first laboratory exercise, entitled **The EXTERNAL Command**.

OUT08 - SLIDE: External Utilities

THE OUTSIDE WORLD

External Utilities

* hp4xed
* hp4stoa
* hp4atos
* hp4xref
* hp4remk
* hp4reod
* hp4ch_keys

Student Notes

External Utilities

A number of **external utilities** are supplied with the HP ALLBASE/4GL system. Some are fully supported, while others are made available on the understanding that they are not supported. Some are available on both HP-UX and MPE XL, others are just available on HP-UX.

The utilities to be investigated in this training course are listed below. Apart from their association with HP ALLBASE/4GL, they all have one thing in common. They all work with HP ALLBASE/4GL S-files or with the **ASCII** output of HP ALLBASE/4GL S-files.

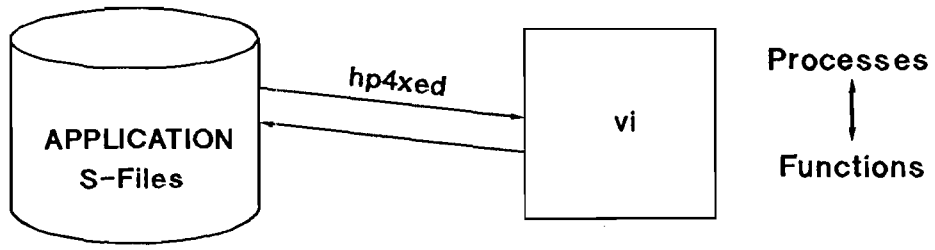
- hp4xed
- hp4stoa
- hp4atos
- hp4xref
- hp4remk
- hp4reod
- hp4ch_keys

OUT09 - SLIDE: External Utilities - hp4xed

THE OUTSIDE WORLD

External Utilities

hp4xed (HP-UX only)



Student Notes

A simple **hp4xed** file could contain the following logic blocks:

```
P-data_entry
01 MODE *WRITE people
02 SCREEN people
03 FILE *WRITE people
04 ENTER 2

F-keyed_read
01 FILE *READ people *KEY=* ; EXIT
02 SHOW *REFRESH
03 EXIT
```

External Utilities - hx4xed

Only available on HP-UX, the utility **hp4xed** allows the developer to extract HP ALLBASE/4GL processes and functions into an **ASCII** file, modify the logic block(s), and then write the changes back into the HP ALLBASE/4GL application.

The name of the utility derives from its facility as an **eXternal EDitor** for HP ALLBASE/4GL logic blocks.

Environment

To operate, **hp4xed** requires the environment variables **HP4GLPATH** pointing to an **HP4GL** directory and **HP4SPATH** to be pointing to a set of HP ALLBASE/4GL S-files. For this reason, it is ideally suited to be run from a **shell escape** from HP ALLBASE/4GL; the parent HP ALLBASE/4GL session will ensure that the correct environment is established.

Jsage

hp4xed can be run just by typing its name. Alternatively, the name of an application can be supplied on the command line.

If no command line argument is given, **hp4xed** prompts the user for the name of an application. It then prompts for the type of logic block (process, function or both). Finally, it prompts for **patterns** by which HP ALLBASE/4GL logic blocks can be identified; it continues to prompt for patterns until the user presses [Return] to finish. Those familiar with HP-UX **pattern matching** will appreciate the power of this feature.

Basic Operation

Having finished the preliminaries, **hp4xed** will unload the logic blocks into a temporary file and automatically invoke the HP-UX editor **vi**. The user is free to make changes, although the essential structure of the HP ALLBASE/4GL logic blocks must be preserved. When the user exits **vi**, **hp4xed** performs some elementary (not HP ALLBASE/4GL syntax) checks on the file. It warns the user if new logic blocks were created or if some were deleted, and prompts to see if the changes should be written back into the S-files.

File Structure

Each logic block is included in the file in essentially the same format the user sees when using the **L**, or **long listing** action within the HP ALLBASE/4GL internal logic editor. In that case, the user sees an unformatted screen containing the text of the logic block.

The only difference is that each logic block contains a one-line header containing the name of the item in the form **P-process_name** or **F-function_name**.

For example, a simple file could contain the following logic blocks:

```
P-data_entry
01 MODE *WRITE people
02 SCREEN people
03 FILE *WRITE people
04 ENTER 2

F-keyed_read
01 FILE *READ people *KEY=* ; EXIT
02 SHOW *REFRESH
03 EXIT
```

Advantages

The advantages of using an editor of the power of **vi** should be readily apparent.

The developer can use the **global-search-and-replace** facility, pattern matching, **yank-and-put**, plus the general editing capabilities of most external editors.

What is not so obvious is the advantage which every experienced HP ALLBASE/4GL developer knows. HP ALLBASE/4GL provides no facility for transferring a process to a function, or vice versa; **hp4xed** makes that possible.

To do so, the developer simply changes the letter **P** to the letter **F**, or vice versa.

Location

hp4xed is stored in the **HP4GL/contrib/bin**; recall that the **contrib** utilities must be manually loaded from the product tape using the HP-UX **tar** utility. It is an unsupported utility.

To load the **contrib** utilities, type the following when logged in as **super-user**:

```
cd $HP4GLPATH
cd ..
tar -xvf /dev/rct/rct HP4GL/contrib/bin
```

The current directory must be one above the **HP4GL** directory. A **README** file is provided in the **./HP4GL/contrib** directory; this provides full instructions.

A manual page is supplied in the **HP4GL/contrib/man** directory.

OUT10 - SLIDE: External Utilities - hp4stoa

THE OUTSIDE WORLD

External Utilities – hp4stoa

The **HP4STOA** utility is used to *unload* an HP ALLBASE/4GL application to an ASCII file.

HP-UX: hp4stoa -a transfer -u developr > transfer.asf

MPE XL: hp4stoa "-a transfer -u developr > transfer.asf"

Student Notes

Refer to the page entitled **Sample Program Fragments** for a sample of the output of **hp4stoa**.

External Utilities - hp4stoa

hp4stoa is similar to **hp4xed** in that it allows HP ALLBASE/4GL items to be extracted from the S-files and stored in an **ASCII** file. **hp4stoa** surpasses **hp4xed** however, in that it allows all HP ALLBASE/4GL items to be stored in the flat file, not just processes and functions.

The name derives from its ability to transfer an application from **S-files TO Ascii.**

hp4stoa is available on HP-UX and MPE XL, where it is known as **HP4STOA**. Both versions operate identically. References to **hp4stoa** are equally applicable to **HP4STOA**.

Environment

To operate, **hp4stoa** requires the environment variable **HP4SPATH** to be correctly pointing to a set of HP ALLBASE/4GL S-files. For this reason, it is ideally suited to be run from a **shell escape** from HP ALLBASE/4GL; the parent HP ALLBASE/4GL session will ensure that the correct environment is established.

Usage

The syntax is identical to that for **logon-bypass**, except that the user can supply further options.

On HP-UX, type:

```
hp4stoa -a application:password -u user:password [option ...]
```

On MPE XL, type:

```
hp4stoa "-a application:password -u user:password [option ...]"
```

The options are very powerful, allowing the developer to use **pattern matching** to select or exclude item types and to select or exclude items of a particular type.

Before using **hp4stoa**, the MPE XL student should refer to the HP-UX manual pages supplied at the end of this module. The HP-UX version of HP ALLBASE/4GL provides the manual page in the **HP4GL/man** directory.

hp4stoa normally directs output to the screen; the output can also be directed to a file using the > symbol.

On HP-UX, type:

```
hp4stoa -a application:password -u user:password [option ...] > asf_file
```

On MPE XL, type:

```
hp4stoa "-a application:password -u user:password [option ...] > asf_file"
```

File Structure

hp4stoa file format is more complicated than that for **hp4xed**. Each file contains a header detailing version numbers and the date and time. Each HP ALLBASE/4GL item in the file is written in a specific format, with a single-line header and curly braces delimiting the beginning and end of the item. Each line is entirely enclosed in single quotes with a semi-colon at the very end. The overall layout is very similar to **C-code** structure.

The developer is free to change the contents of the file using an external editor such as **vi** on HP-UX or **HPEDIT** on MPE XL; however, the very strict **hp4stoa** structure cannot be altered.

For example, a simple file could contain the following items:

```
n-computer
config        test
(
        stoa_ver        'B.01.00';
        sfile_ver       3.6;
        unload_date     '90/02/11';
        unload_time     '10:52:55';
        system          'HP-UX';
        os_release      '6.5';
        machine         '9000/370';
)
```

```
field_spec action
(
        sdesc   'action';
        ldesc   'AUTHOR: developr';
        date    '90/01/12';
        time    '11:39:42';

        secured N;
        length  1;
        repeated            1;
        min_entry           1;
        edit_code           U;
        justification       N;
        pad_char            ' ';
        range   '';
        table   '';
        help    '';
)

process call_pascal
(
        sdesc   'call_pascal';
        ldesc   'AUTHOR: simon'
                'Call the external pascal program úpcase';
        date    '89/10/16';
        time    '20:45:36';

        logic_details
        (
                1 'SCREEN call_pascal';
                2 'EXTERNAL upcase *S01  V-return_value';
                3 'SHOW *REFRESH';
                4 'MESSAGE pause';
        )
)
```

Advantages

As with **hp4xed**, the advantages of using an editor of the power of **vi** or **HPEDIT** are easily apparent.

The developer can use the **global-search-and-replace** facility, pattern matching, **yank-and-put**, plus the general editing capabilities of most external editors.

hp4stoa by itself is limited, however in conjunction with its companion utility **hp4atos**, the power, and applicability, of **hp4stoa** surpasses **hp4xed** because it is possible to deal with the entire application.

It opens up the possibility of global editing of an entire application, as would be needed when localising the application, a task which requires translation of all of the text into another language.

It also provides an alternative method of transferring applications between S-files. Applications can be merged together; an HP ALLBASE/4GL **version** can even be merged back into the base application.

The possibilities are enormous.

Error Checking

The **-n** option to **hp4stoa** prevents any output; this is not entirely useless however. Whenever **hp4stoa** is run, it actually checks each record in the S-files, thus providing a means of checking an application for inconsistencies in the S-files.

The S-files consist of 12 ISAM/KSAM files; an application relies on **all** of them being intact and up-to-date. S-file inconsistencies may occur from time to time, mainly through operating system errors or through incorrect restoration from a backup. For example, copying an **s04** file from another set of S-files would certainly create havoc.

Fortunately, **hp4stoa** can also help when inconsistencies or errors have been identified. Many inconsistencies or errors can be fixed by simply accessing HP ALLBASE/4GL and re-committing the screen for the offending item. Other problems require the developer to delete the item and re-enter the details.

Using the **-c** option, **hp4stoa** can be used to unload as much as possible of the inconsistent or incomplete item before the deletion, allowing a repaired copy of the item to be loaded back in to the application by **hp4atos**.

If a customer is having problems with inconsistencies or errors in their application, contact the **ASO Support Group** for assistance.

Location

On HP-UX, **hp4stoa** is stored in the **HP4GL/bin** directory along with the main HP ALLBASE/4GL program files.

On MPE XL, **hp4stoa** is stored in the **PUB.SYS** group along with the main HP ALLBASE/4GL program files. The **Evaluation Pack** version of HP ALLBASE/4GL will store **hp4stoa** in the **BIN.HP4DEMO** group along with the main HP ALLBASE/4GL program files.

OUT11 - SLIDE: External Utilities - hp4atos

THE OUTSIDE WORLD

External Utilities – hp4atos

The **HP4ATOS** utility is used to *upload* an HP ALLBASE/4GL application from an ASCII file.

HP-UX: hp4atos -a transfer -u developr < transfer.asf
MPE XL: hp4atos "-a transfer -u developr < transfer.asf"

Student Notes

External Utilities - hp4atos

hp4atos is the companion utility to **hp4stoa** as it provides the ability to write information from an **ASCII** file back into the HP ALLBASE/4GL S-files. The **ASCII** file must be in a special format, identical to that produced by **hp4stoa.**

The name derives from its ability to transfer an application from **Ascii TO S-files.**

hp4atos is available on HP-UX and MPE XL, where it is known as **HP4ATOS.** Both versions operate identically. References to **hp4atos** are equally applicable to **HP4ATOS.**

Environment

To operate, **hp4atos** requires the environment variable **HP4SPATH** to be correctly pointing to a set of HP ALLBASE/4GL S-files. For this reason, it is ideally suited to be run from a **shell escape** from HP ALLBASE/4GL; the parent HP ALLBASE/4GL session will ensure that the correct environment is established.

Usage

The syntax is identical to that for **logon-bypass** except that the user can supply further options.

On HP-UX, type:

```
hp4atos -a application:password -u user:password [option ...]
```

On MPE XL, type:

```
hp4atos "-a application:password -u user:password [option ...]"
```

The options are very powerful, allowing the developer to use **pattern matching** to select or exclude item types and to select or exclude items of a particular type.

Before using **hp4atos**, the MPE XL student should refer to the HP-UX manual pages supplied at the end of this module. The HP-UX version of HP ALLBASE/4GL provides the manual page in the **man** directory immediately below **HP4GL** directory.

Although **hp4atos** normally takes input from the keyboard, input can be taken from a file using **re-direction** symbol <.

On HP-UX, type:

```
hp4atos -a application:password -u user:password [option ...] < asf_file
```

On MPE XL, type:

```
hp4atos "-a application:password -u user:password [option ...] < asf_file"
```

File Structure

As mentioned above, **hp4atos** can only read a file in the format produced by **hp4stoa**. **hp4atos** checks very carefully for errors in the format and the slightest problem will cause **hp4atos** to terminate processing.

It cannot afford to write the slightest bit of garbage back into the HP ALLBASE/4GL S-files.

Advantages

hp4atos by itself is limited, however in conjunction with its companion utility **hp4stoa**, the power is considerable.

Error Checking

The **-n** option to **hp4atos** prevents any attempt to write to the S-files; as with **hp4stoa**, this is not entirely useless however. Whenever **hp4atos** is run, it actually checks each line in the ASF file. Thus the **-n** option provides a quick means of checking an application stored in an ASF file.

A replacement for hp4xed

It is very simple, on both HP-UX and MPE XL, to **wrap-up hp4stoa** and **hp4atos** in a script or command file to automate the process of reading out, editing and writing back an HP ALLBASE/4GL application.

This produces a super **hp4xed**, available on both HP-UX and MPE XL, which can deal with an entire application in **ASCII** format.

OUT12 - SLIDE: External Utilities - hp4xref

THE OUTSIDE WORLD

External Utilities

hp4xref

Australian Software Operation
out12

HEWLETT PACKARD

Student Notes

External Utilities - hp4xref

Regardless of the programming language or environment, programmers or developers regularly wish to know about the structure of their application. Such things as which items call other items, which **calls** are to **non-existent** items and which items of their creation are **unreferenced**. A compiler will rudely announce **undeclared** or non-existent items but a **cross-referencer** is required to automatically find unreferenced items.

HP ALLBASE/4GL comes with a **cross-reference** utility called **hp4xref**.

The name derivation is obvious.

hp4xref is available on HP-UX and MPE XL, where it is known as **HP4XREF**. Both versions operate identically. References to **hp4xref** are equally applicable to **HP4XREF**. On HP-UX, is an unsupported binary on HP-UX and is supplied with full source code.

Environment

No special environment is required as **hp4xref** does not read from or write to S-files.

Usage

hp4xref normally takes input from the keyboard; input can also be taken from a file using **input re-direction**.

hp4xref expects input in **hp4stoa** format, as it is intended to read an **ASF** file.

Output is normally directed to the screen but can be captured to a file using **output re-direction**.

On HP-UX, type:

```
hp4xref [options] [< asf_file]  [> xref_file]
```

On MPE XL, type:

```
hp4xref "[options] [< asf_file] [> xref_file]"
```

File Structure

As mentioned above, **hp4xref** expects a file in the format produced by **hp4stoa**. Output is **ASCII** free format.

For example:

```
HP ALLBASE/4GL Procedure call tree for 'demo'
               scr:main -->
                   sof:main -->
                pro:load_application
                   pro:members_local
                   pro:purch_ord_local
                   pro:-training_local
                   pro:example
                   pro:mckinney
```

Advantages

hp4xref can inform the HP ALLBASE/4GL developer about the structure of an application by producing a **procedure call tree.**

It can also list only those items which are **undefined but referenced**, or only those items which are **defined but unreferenced.**

By default, **hp4xref** lists objects, and where they are referenced. The second form, produced by the **-i** option and known as **inverted mode**, lists objects and what they reference.

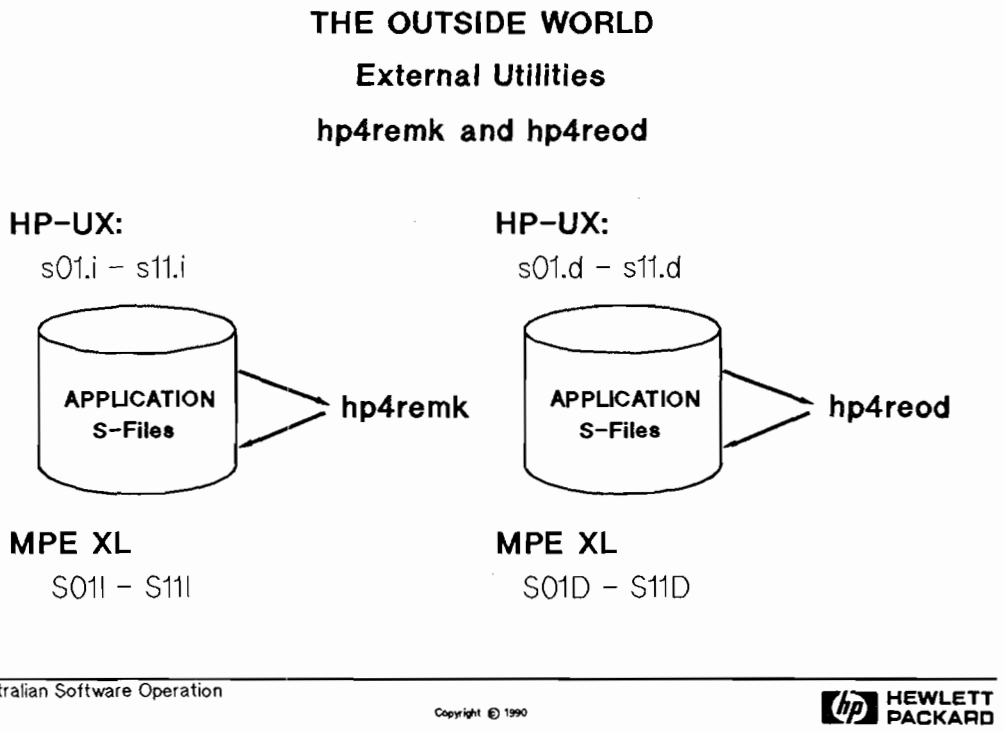
hp4xref can also selectively report on a class of items.

Location

On HP-UX, **hp4xref** is stored in the **HP4GL/contrib/bin** directory; along with the other unsupported binaries, it must be manually loaded from the product tape using the HP-UX utility **tar**. A manual page is stored in the **HP4GL/contrib/man** directory.

On MPE XL, **HP4XREF** is stored in the **PUB.SYS** group along with the main HP ALLBASE/4GL program files. The **Evaluation Pack** version of HP ALLBASE/4GL will store **HP4XREF** in the **BIN.HP4DEMO** group along with the main HP ALLBASE/4GL program files.

OUT13 - SLIDE: External Utilities - hp4remk & hp4reod



Student Notes

Usage on HP-UX

```
hp4remk [nn[-nn]]
hp4reod [nn[-nn]]
```

Usage on MPE XL

```
HP4REMK [nn[-nn]]
HP4REOD [nn[-nn]]
```

External Utilities - hp4remk & hp4reod

The utilities **hp4remk** and **hp4reod** are a **matched pair** and should not be seen separately.

Remake or **hp4remk** is used to rebuild the index files; it reorganises the index file by balancing the index tree, reclaiming any dead space, and checking for invalid indexes.

Reorder or **hp4reod** reoders the data portion of an S-file. It reorganises the data file by recreating it in index order and purging any logically deleted records.

These utilities are to S-files what **bcheck** is to ISAM files and what **ksamutil** is to KSAM files.

hp4remk and **hp4reod** are available on both HP-UX and MPE XL; on MPE XL they are known as **HP4REMK** and **HP4REOD**.

Again, the name derivations should be obvious; the strange spelling is due to the constraint of 8-character filenames on MPE XL.

Environment

hp4remk and **hp4reod** both require that the **HP4SPATH** environment variable be set to point to a set of HP ALLBASE/4GL S-files. On HP-UX, the environment variable **HP4GLPATH** must also be set.

Unlike some of the previous utilities which require an HP ALLBASE/4GL environment, they should **never** be run from a **shell escape**.

Precautions

Only the system administrator or the HP ALLBASE/4GL administrator should use these utilities; they should ensure that no HP ALLBASE/4GL users are running applications while **hp4remk** and **hp4reod** are being used.

Before use, a backup copy of the S-files should be made.

WARNING

hp4reod must be run after **hp4remk**; failure to do so may corrupt the S-files.

Usage on HP-UX

hp4remk and **hp4reod** will accept a command line argument to indicate the file or files to be processed. If no argument is given, the programs will prompt the user for the number of the file.

The syntax is:

```
hp4remk [nn[-nn]]
hp4reod [nn[-nn]]
```

For example, to **remake** and **reoder** file 5, type:

```
hp4remk 5
hp4reod 5
```

To **remake** and **reoder** files 1 to 11, type:

```
hp4remk 1-11
hp4reod 1-11
```

Usage on MPE XL

On MPE XL, **hp4remk** and **hp4reod** accept a command line argument to indicate the file or range of files to process.

The syntax is:

```
HP4REMK [nn[-nn]]
HP4REOD [nn[-nn]]
```

For example, to **remake** and **reoder** file 5, type:

```
HP4REMK 5
HP4REOD 5
```

To **remake** and **reoder** files 1 to 11, type:

```
HP4REMK 1-11
HP4REOD 1-11
```

If no command line argument is given, the programs will prompt the user for the number of the file to **remake** or **reorder**.

Location

On HP-UX, **hp4remk** and **hp4reod** are stored in the **HP4GL/bin** directory along with the main HP ALLBASE/4GL program files. A manual page is provided for each in the **man** directory.

On MPE XL, **hp4remk** and **hp4reod** are stored in the **PUB.SYS** group along with the main HP ALLBASE/4GL program files. The **Evaluation Pack** version of HP ALLBASE/4GL will store **hp4remk** and **hp4reod** in the **BIN.HP4DEMO** group along with the main HP ALLBASE/4GL program files.

Demonstration

Gather the students around a terminal as you demonstrate the use of **hp4remk** and **hp4reod**.

Copy a set of S-files to a temporary directory or a temporary group; on HP-UX, determine the size by using **du** and **ls -l**; on MPE XL, use **listf,2**. Note the size of the larger S-files.

Delete an application from the S-files. Then run **hp4remk** and **hp4reod** on all 11 S-files. The exercise will take some time to complete so continue on with the module theory in the meantime. Note the sizes when the programs have finished.

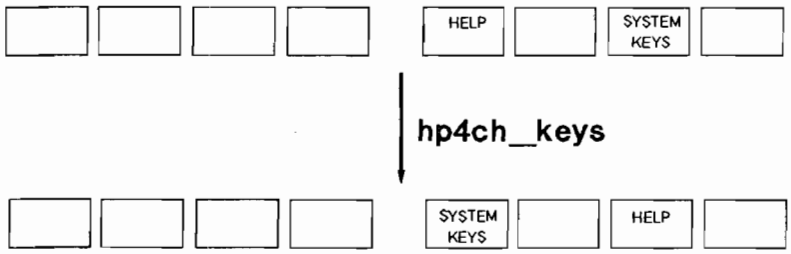
Intentionally blank

OUT14 - SLIDE: External Utilities - hp4ch_keys

THE OUTSIDE WORLD

External Utilities

hp4ch_keys(HP-UX only)



Australian Software Operation
out14

HEWLETT PACKARD

Student Notes

External Utilities - hp4ch_keys

The unsupported utility **hp4ch_keys** is only available, and only necessary on HP-UX.

Formerly, HP ALLBASE/4GL was known as **HP TODAY**; in that version, the actions `System Keys` and `Help` were located on function keys (f7) and (f5) respectively.

With the HP ALLBASE/4GL release, it was decided to swap those actions around. The **hp4ch_keys** utility has been provided to automatically swap `System Keys` and `Help` in end-user's applications.

Environment

hp4ch_keys requires the environment variables **HP4GLPATH** and **HP4SPATH** to be set. If these are not set, it prompts the user for the value of **HP4GLPATH**, from which it determines the value of **HP4SPATH**, assuming a standard installation environment.

hp4ch_keys checks that the environment variables point to valid directories; it also ensures that **hp4stoa** and **hp4atos** are available.

Basic Operation

hp4ch_keys is an HP-UX **shell script** which uses **hp4stoa** to unload the function keys from the application, processes the ASF file using a special program written in the HP-UX language called **awk**, and then uses **hp4atos** to write the new version of the ASF file back into the application.

The original ASF file containing the old function key layout is preserved.

No damage occurs if **hp4ch_keys** is run twice; it will only swap the `System Keys` and `Help` function keys if they are in the old **HP TODAY** layout.

No action is taken if only one of the function keys is found in the user's application.

No other function keys are affected.

hp4ch_keys is able to swap function keys in runtime environments as HP ALLBASE/4GL function keys are not generated items. The source is always present, even in a runtime

application. If a user is running the application while **hp4ch_keys** processes the function keys, the change will not be apparent until the user exits the application and restarts.

Usage

hp4ch_keys will accept a command line argument to indicate the name of the application to be processed. If no argument is given, the program will prompt the user for the name of the application. **hp4ch_keys** checks that the application specified exists before processing.

The syntax is:

```
hp4ch_keys [application_name]
```

For example, to process the application **example**, type:

```
hp4ch_keys example
```

Location

hp4ch_keys is stored in the **HP4GL/contrib/bin** directory along with the other unsupported programs. A manual page is provided in the **contrib/man** directory.

The EXTERNAL Command

In this lab, you will use the **EXTERNAL** command to implement two simple interfaces to programs outside the HP ALLBASE/4GL environment. There is also an optional exercise involving writing and reading pipes or message files.

HP-UX: Passing and Returning Parameters

This exercise will concentrate on how an HP ALLBASE/4GL application can pass information to and from an **EXTERNAL** program.

Your application will call an **EXTERNAL** program to return the value of the **HP4DATAPATH** environment variable.

The module will initially involve a process, a screen, a message and a named scratch pad.

First, define a scratch-pad called **HP4DATAPATH**.

Next, create a query type message called **pause** to stop the process until the user presses Return; give the message the appropriate text.

Create a screen containing the appropriate titles and just one display field to show the value of the **HP4DATAPATH** environment variable. The primary move for the field should be to/from the scratch-pad.

Create a process called **get_datapath**, as follows:

```
01 SCREEN datapath
02 MESSAGE pause
03 EXTERNAL get_datapath P-HP4DATAPATH
04 SHOW *REFRESH
05 MESSAGE pause
06 EXIT
```

Create the external script **get_datapath**, as follows:

```
#!/bin/sh
#
#       Name:         get_datapath
#       Author:       Instructor
#       Date:         29/2/1990
#       Purpose:      To return to HP ALLBASE/4GL the value of HP4DATAPATH
#
#


#
#       Determine the length of the variable
#

len=`expr length $HP4DATAPATH`

#
#       Send back the word EXTERNAL to file descriptor 3
#

echo "EXTERNAL" >&3

#
#       Send back the length and the value to file descriptor 3.  The data
#       will be written into the first parameter on the command line
#       in the EXTERNAL command after the name of the program; i.e. the
#       scratch pad HP4DATAPATH

echo "1 $len $HP4DATAPATH" >&3

#
#        Exit and set the return status to 1
#

exit 1
```

MPE XL: Setting a File Equation

The users wish to be able to re-direct HP ALLBASE/4GL reports anywhere. You decide to implement this using the **EXTERNAL** command to call an external command file; the command file will set a **file equation** according to the user's entries.

Create a screen with the following fields:

Field Name	Length	Edit Code
Report name	8	X
Device	8	X
Output Priority	2	N
Number of Copies	3	N

The primary movement field for each field should be a named scratch pad named according to the screen field. Associate a range with the **Output Priority** field, to force a value between **1** and **13**. Associate a range with the **Number of Copies** field, to force a value between **1** and **127**.

```
01 SCREEN   file_equation
02 LINK *JOINER="," 3 P-device P-priority P-number_copies P-options
03 EXTERNAL fileqn P-report_name P-options
04 MESSAGE pause
```

The external command file **FILEQN** is shown below:

```
PARM F1=* F2=*
comment
comment         A command file to get the information sent out from
comment         HP ALLBASE/4GL and set up a file equation to re-direct
comment         the report output on MPE XL.
comment
comment         Name:           fileqn
comment         Author:         Instructor
comment         Date:           29/2/1990
comment


comment
comment         If no parameter has been supplied, give an error message
comment         and exit
comment
```

```
if "!F1" = "*" then
        echo "You must supply the report name and the options"
else

        comment
        comment         Assign the first parameter to the variable repname
        comment

        setvar repname "!f1"

        comment
        comment         Assign the second parameter to the variable options
        comment

        setvar options "!f2"


        comment
        comment         Now make a file equation
        comment

        file myrept=!f1;dev=!f2

endif
```

The word **myrept** indicates the name of the **Formal File Designator**, to which the report will be re-directed. Enter that name for one of the reports developed in a previous module.

Now run the application and enter some details into the screen. Then escape to MPE XL by pressing `System Keys` `More Keys` `Op. System`. Type **listeq** to confirm that the file equation has the values just entered; return to HP ALLBASE/4GL by typing **exit**.

Run a report to check that the report is sent to the correct device.

Now enter some different values to re-direct the report to a different device.

EXTERNAL *REFRESH

The users have requested a simple way to access an external editor from their HP ALLBASE/4GL application. We will implement this operation using the **EXTERNAL *REFRESH** command to call the editor; if using HP-UX, call the editor **vi**; if using MPE XL, call the editor **EDITOR**.

Do the following steps:

1. Create an **Alpha-Numeric Constant** called **editor**; give it the value **vi** or **EDITOR**

2. Create a function called **call_editor**, as shown below:

```
01 SCREEN get_filename
02 EXTERNAL *REFRESH C-editor V-filename
03 EXIT
```

3. Create a screen called **get_filename**; this will contain just one field, prompting the user for the name of the file to be edited.

4. Set up a call to the function from either your main menu or from a function key.

5. Test the operation of the system. Create a new file using the editor. Return to HP ALLBASE/4GL and then re-edit the file.

Questions

Why do we need to use the *REFRESH option here?

External Utilities

hp4stoa & hp4atos

Perform the following steps:

1. Log in to your application and then escape to the operating system.
2. Check the HP ALLBASE/4GL environment by using **env** on HP-UX or **showvar** on MPE XL.
3. Unload your application to an **ASF** file using **hp4stoa**; the **ASF** file should have your name. On HP-UX, it should have the extension **.asf**; on MPE XL, it should be in the **ASF** group.
4. Examine the **ASF** file using an editor; modify some function keys, a message or the text on a screen. Be careful to maintain the format produced by **hp4stoa**.
5. Read the **ASF** back into the S-files using **hp4atos**.
6. Back in your application, re-generate the item you changed (if necessary) and run the application to see the effect.
7. Escape back to the operating system and copy the **ASF** file to a new file.
8. Edit the new **ASF** file to change the name of the application; you should only change the **header** information.
9. Try to read the new **ASF** file into the S-files. Note the error message produced by **hp4atos**.
10. Go back into HP ALLBASE/4GL and log in as the administrator. Define the new application and then attempt to load the new application once more.
11. Return to HP ALLBASE/4GL, log in to the new application, do a **Generate ALL** and test it to ensure it is all complete.

hp4xref

Perform the following steps:

1. Read through the manual page for **hp4xref** supplied at the end of the module notes.
2. Run **hp4xref** to produce a report for your application. Save the report in a file.
3. Browse through the **report file** using a pager such as **more** or **print**.

Using HP ALLBASE/4GL with a Pipe or Message File

This exercise is optional. In this exercise, you will use HP ALLBASE/4GL to write to a pipe or message file and read from the same pipe or message file. The **writer** will take data from a typical data entry screen and send it to the pipe or message file. The **reader** will read the data from the pipe or message file and display it on a screen using the **SCROLL** command.

Follow the steps below:

- Develop a shell script on HP-UX called **mk_pipe** to create a pipe OR develop a command file on MPE XL called **bldmess** to create a message file. Call the shell script or command file using the **EXTERNAL** command in the **writer** process; pass the name of the pipe from HP ALLBASE/4GL, using the user's login name in the filename.

- Develop a process to operate as the **writer** to the pipe or message file. Use the word **Finished** as the sentinel.

- Develop a process to operate as the **reader** for the pipe or message file. Use the word **Finished** as the sentinel.

- Develop two screens, one for the **writer** and one for the **reader** as specified above.

- Attach both processes to the same menu in your application.

- Run the application on two different terminals. Ensure that both applications terminate correctly.

Possible solutions:

```
#!/bin/sh
#
# make_pipe
#
# A script to make a pipe
#
# Author:  S. Hiscox
# Date:  22/2/1989
```

```
#
#-------------------------------------------------------
#

MKNOD=/etc/mknod

if [ $# -ne 1 ]
then
          echo "$0:  You must supply the name of the pipe."
          exit 1
fi

#
# Get the name of the pipe from the command line
#
# It will be /tmp/pipe.<user_name>
#

pipe_name=$1


#
# If the pipe already exists, remove it
#

if [ -p $pipe_name ]
then
        rm -f $pipe_name
fi

#
# Now make the pipe
#

$MKNOD $pipe_name p

exit 1
```

```
P-read_from_pipe
01 DEFINE %f% line_from_file
02 MODE *READ  pipe
03 LINK  2 "/tmp/pipe." *USER *P02
04 MOVE *P02 *FILENAME
05 SCREEN read_pipe
06 MESSAGE ready_to_read
07 FILE *READ pipe.line_from_file ; ENTER 11
08 IF F-%f%.pipe.%f% = "Finished"
   THEN MESSAGE end_of_pipe ; EXIT
```

```
09 SCROLL F-line_from_file.pipe.line_from_file
10 ENTER 7
11 IF *IOSTATUS = N-end_of_file THEN ENTER 7
12 IF *IOSTATUS = N-no_pipe THEN MESSAGE no_pipe
13 IF *PASS = "Y" | *PASS = "y" THEN ENTER 7


P-write_to_pipe
01 MODE *WRITE pipe
02 LINK  2 "/tmp/pipe." *USER *P02
03 MOVE *P02 *FILENAME
04 EXTERNAL make_pipe *FILENAME
05 FILE *BUFFER pipe
06 MOVE *P02 *FILENAME
07 SCREEN write_to_pipe
08 IF *P50 = "exit" THEN ENTER 13
09 MOVE S-total F-total.pipe
10 FILE *WRITE pipe
11 ENTER 5
12 NOTE Clean up : send the sentinel 'Finished'
       down the pipe and then remove it.
13 MOVE "Finished" F-payee.pipe
14 FILE *WRITE pipe
15 FILE *REMOVE pipe
16 EXIT
```

Sample Program Fragments

Return Path on HP-UX:

For HP-UX, the external program must write to **file descriptor 3**.

This is a very simple task for a shell script or **C-program**; it is not so easy for a **Pascal** program or using another language.

Shell Script

The following code fragment illustrates how to return information from an HP-UX shell script.

```
echo "EXTERNAL" > &3
echo 1 5 Hello > &3
echo 2 10 This is it > &3
```

The **echo** command normally sends the information to the screen but **&3** re-directs it to **file-descriptor 3**.

C-Program

The following code fragment illustrates how to return information from an HP-UX C-program.

```
fptr = fdopen(3,"w");
fprintf(fptr, "EXTERNAL");
fprintf(fptr, "1 %d %s", strlen(str1), str1);
fprintf(fptr, "2 %d %s", strlen(str2), str2);
```

where **str1** and **str2** are variables of type **char**, containing **Hello** and **This is it** respectively.

fptr is a **file pointer**; it has opened **file descriptor 3** with **write access**.

Return Path on MPE XL:

For MPE XL, the external program must write to the file **HP4EXTP**; this must be a variable length record binary message file.

This is not straight-forward for any programming language on MPE XL.

C-Program

The following code fragment illustrates how to return information from an MPE XL C-program.

```
sprintf(fname, "&HP4EXTP&");
HPFOPEN(8, &filenum, &status, 2, fname, 3, &domain, 11, &access);
sprintf(fname, "EXTERNAL!");
FWRITE((short)filenum, (char *)fname, (short)-9, CONTROLCODE);
FWRITE((short)filenum, (char *)str1, (short)-strlen(str1), CONTROLCODE);
FWRITE((short)filenum, (char *)str2, (short)-strlen(str2), CONTROLCODE);
```

fname is a character array of length 128. **filenum** and **status** are integers. **domain** is an integer with value **2**, **access** is an integer with value **1**.

str1 and **str2** are character arrays containing **Hello** and **This is it** respectively.

Some error checking code has been eliminated for simplicity.

References

Read the Developer Reference Manual entries for the **EXTERNAL** command very thoroughly.

Further information can be obtained from some **ASO Support Newsletter** articles. In particular, Simon Hiscox wrote an article describing how to return parameters from a **Pascal** program running on an HP 9000 Series 300 computer running HP-UX; the article is in Newsletter #11. An article by David Williams, in Newsletter #3, also discusses parameter passing with external programs.

Sample output from hp4stoa

A simple **hp4stoa** file could contain the following items:

```
n-computer
config        test
(
        stoa_ver        'B.01.00';
        sfile_ver       3.6;
        unload_date     '90/02/11';
        unload_time     '10:52:55';
        system          'HP-UX';
        os_release      '6.5';
        machine         '9000/370';
)

field_spec action
(
        sdesc   'action';
        ldesc   'AUTHOR: developr';
        date    '90/01/12';
        time    '11:39:42';
        secured N;
        length  1;
        repeated            1;
        min_entry           1;
        edit_code           U;
        justification       N;
        pad_char            ' ';
        range   '';
        table   '';
        help    '';
)

process call_pascal
(
        sdesc   'call_pascal';
        ldesc   'AUTHOR: simon'
                'Call the external pascal program úpcase';
        date    '89/10/16';
        time    '20:45:36';
        logic_details
        (
                1 'SCREEN call_pascal';
                2 'EXTERNAL upcase *S01  V-return_value';
                3 'SHOW *REFRESH';
                4 'MESSAGE pause';
        )
)
```