SmartWare Developer

COPYRIGHT INFORMATION

Printed: 1997/06/10 - FH vandenHeuvel

34 St.Patrick Street • Suite 200
Toronto, Canada • M5T 1V1

Table of Contents

Table of Contents, i

Chapter 1: Introduction, 13

The ANGOSS Developer, 13
Getting Started, 13

Chapter 2: The Developer Tutorial, 15

Introduction, 15

Part I - New Applications, 16
Creating an Application, 16
Starting the Application, 18
Looking Around the System, 19
Changing the System Title, 21
Accessing the Developer System, 22
The Developer Menu, 23
Inserting a Menu, 23
Inserting a Data View, 26
The Custom-View Editor, 27
Defining Key Fields, 28
The Data View, 30
Data View Commands, 31
Exit the Application, 37

Part II - Data Views, 38
Creating a Data View - Review, 38
Repositioning a Menu Item, 40

Modifying the Employees Data View, 41

Part III - The Print Menu, 48
Printing, 48
The Print Menu, 49
Custom Reports, 49
The Report Generator, 50
Sending Data to the Spreadsheet, 52
Sending Data to the Word Processor, 54
Creating a RAD Report, 56
Executing a RAD Report, 60

Part IV - System Administration, 62
User Modes, 62
Users, 65
Hierarchy Diagram, 68
Procedures and Job Streaming, 70

Summary, 72

Chapter 3: Developer System Menus, 75

Accessing the Developer, 75
First Level Menu, 75

Inserting an Item, 76
Menu, 76
Database, 77
Spreadsheet, 78
Wordprocessor, 78
Program, 79
OS_Call, 80
Text_File, 81
New_Application, 82
Archive, 82
System_Defaults, 82
Print, 83
Print_Crystal, 83

Keyboard_Macros, 84
Job_Stream/Procedure, 84
Return, 84
Data_Entry, 85
User_Utilities, 85
Totals, 85
Update, 86
Graph, 86
Report, 87
Edit_Area, 87
Spreadsheet, 88
Word_Processor, 88
Communications, 88

Modifying an Item, 88
Browse_Fields, 89
Default_Edit_Areas, 90
DV_Info_(sysdef), 90
Edit_Area, 90
FPR_Template, 91
FPR_Keys, 91
Graph, 92
Information, 92
Input_Screen, 92
Add_Key, 93
Delete_Key, 93
Item_Information, 93
Library, 94
Link_Definition_(DB), 94
Link_Definition_(SS), 94
Link_Definition_(WP), 95
Menu, 95
Characteristics, 95
Editor, 96
OSA_Script, 96
Password, 96

Program, 96
Program_(Loading/Unloading), 97
Report, 97
Crystal_Report, 98
Screen_and_DB's, 98
System_Defaults, 99
Text_File, 99
Totals, 99
View_SYSDEF_Program, 100
Worksheet, 100

Deleting an Item, 100

Moving an Item - Reposition, 101

Utilities, 101
Remember, 101
OS, 103
Preferences, 103
Changes_Log, 104
Icon_Editor, 104
Quit, 105

Chapter 4: RAD Objects and Definition Files, 107

Menus, 107
Menu_Characteristics, 107
Item_Information, 109

Data View, 110
Screen Information, 110
Totals Definition, 116

Spreadsheet View, 117
Spreadsheet Information, 117
Edit Areas Definition, 120

Document View, 122
Document Information, 122

DWL Definition, 124

Report Components for Data Views, 125
The Components, 126
Execution Parameters, 130

Chapter 5: Management and Utilities, 133

Users, 135
User Modes, 135
Users, 136

System Defaults, 136

Hierarchy Diagram, 137

Archiving, 138
Creating the Destination View, 139
Creating the Archive Definition, 140
Datafiles Button, 141
User Interface Button, 143
Query Button, 144
Running an Archive Job, 144
Viewing the Log and Archived Data, 144

Help Files and Documentation, 144
Help Files, 144

Documentation, 146

Chapter 6: Advanced Application Development, 149

SPL Programming, 149
RAD SPL Programming, 150

Programmer Overview, 151
Creating a New Application, 151

Start Up, 152
Running, 153
Summary, 154
Accessing SmartWare, 154
Custom Commands, 155

Processing Routines, 157
Audited Programs, 157
Changing Modules, 159

Calling an Application from a Program, 160

Libraries, 161
Application Libraries, 161
Menu Libraries, 162

Programs On Non Modular Specific Menus, 162

Loading/Unloading Programs, 163

Event Handling Functions, 164

Navigation Menu Programs, 167

Running Other Software and Switching Modules, 167
Other Software Packages, 168
Changing Modules, 169

Customizing Login and Start Programs, 170

Passwords on Data Files, 171
Programming the Application to Supply Passwords, 171

Chapter 7: Miscellaneous Information, 173

RAD Colors, 173
Menus, 173
Data Views - Screen Colors, 173

Icons, 174
Adding Icons, 174
Icon Editor, 175

How Data Paths are Handled, 176

Moving Systems, 177
DOS Startup Batch File, 178
Information Set, 178
System Database Paths, 179

Upgrading a User's System, 180
Restructuring Modified Data Files, 180
Copying Program and Definition Files, 181
Test the System, 182

Appendix A, 183

Function Listing, 183
ACTIVE_LEV, 183
ADD_LOG, 184
ASK_FILENAME, 184
ASK_FILENAME_DIR, 184
CHAP, 185
CHECK_FILE_NAME, 185
CHECK_PATH, 186
CHOICE_TITLE, 186
CLEAR_KEYBOARD, 186
CLOSE_CUR_MENU, 187
CLOSE_ALLWIN, 187
COMPLETE_LOG, 194
COUNT_WORDS, 195
DECODE_EDA, 195
DEF_EDIT_AREAS, 196
DISPLAY_MESSAGE, 196
DO_CALC, 196
EDIT_AREAS, 197
ERRORREP, 198
FIND_COMMAND, 198
FINDRECORD, 198
FREE_CELL, 199

GET_ANGII_VER, 199
GET_CHOICE, 200
GET_FILE, 201
GET_LINE, 203
GET_PATH, 204
INT_PATH, 204
KEY_FIELD_PROMPT, 205
KRUNCH_SCREEN, 205
LOADDATAVIEW, 206
LOCK_REC, 206
MENU, 207
MENU_CALL, 208
MENU_TITLE, 209
MODULE_CALL, 209
MODULE_RETURN, 210
ONE_RECORD, 210
OPEN_FILE, 211
OPEN_LOG, 212
ORDER_COMMAND, 212
OS_APL, 213
PASSWORD, 213
POP, 214
PRINT_DOC, 214
PRINT_T_FLD, 215
PUSH, 215
READ_CHOICE, 216
READ_HEADER, 217
RELOAD, 218
REORDER, 218
RESTORE_ALLWIN, 219
RP_EXEC, 219
RUN_CRYSTAL, 220
RUN_QUERY_STR, 221
SCROLL, 222
SORT_STRING, 222
SPLIT_PFE, 222

SUSPEND_APP, 223
TEMP_EXIT, 223
TOOLS_FILE, 224
UP_DB_STAT, 224
WRITE_NAME, 225
WRITE_NUM, 225
WRITE_ULOG, 226
ZAP_DB, 227

Appendix B, 229

Public Variables, 229
General-Purpose Variables, 229
Login Choice Variables, 232
Current Choice Variables, 233
Current Menu Variables, 234
Menu Stack Variables, 235
Status Array Layout, 236
Status Information for Each Module, 236
Current View Variables, 237
Miscellaneous Status Variables, 238

Appendix C, 241

System Files, 241
File Name Extensions, 241
Input Screen Files, 245

Appendix D, 247

Actions, 247
ASVIEW, 247
BROWSE, 247
CA, 248
CALC, 248
CRW, 248
DELETE, 248
DOCV, 249
DV, 249
EDITAREA, 249
ENTER, 249
FCT, 250
FIND, 250
GRAPH, 250
KRUNCH, 250
MACRO, 251
M, 251
MERGE, 251
ORDER, 251
OS, 252
OSAPL, 252
PCR, 252
PROC, 252
QUERY, 253
REPGEN, 253
RP, 253
RT, 253
RTAS, 254
RTNS, 254
SENDSS, 254
SSR, 254
SUSPEND, 255
SV, 255
SYSDEF, 255

TEXT, 255
UPDATE, 256
UPDATESS, 256
UPDATEWP, 256
X, 256
XAUD, 257
ZAP, 257

Appendix E, 259

Converting Applications, 259
Conceptual Overview, 259
Creating the RAD Application, 259
Copying Files and Converting Menus, 260

Index, 265

Chapter 1: Introduction

The ANGOSS Developer

The ANGOSS Developer is a Rapid Application Development system, or RAD, based on the SmartWare Office Automation software. With it you can create small, simple applications - perhaps just to help organize your personal SmartWare files - or large applications that businesses use for any number of purposes.

Although the ANGOSS Developer is a strong tool for Programmers because it frees them from mundane and repetitive tasks, programming knowledge is not a requirement. It is really quite easy to use and you'll be surprised at how quickly you can build full-blown custom applications.

Applications created with the ANGOSS Developer, called RAD Applications, make use of SmartWare's four application modules, namely the: Database, Spreadsheet, Wordprocessor, and Communications modules. And since SmartWare is available on Windows, X Windows, DOS, and Unix, RAD applications automatically inherit the ability to run on any of these platforms with little or no modification. An excellent feature not just for mixed platform environments - which it handles effortlessly - but also for developers who plan to create applications for a variety of clients using different hardware and operating systems.

Getting Started

To get the feel of a RAD application, run the Sample application - or better still, try the tutorial in the next chapter. For the most part, the remaining chapters and appendices are reference material so it's a good idea to get the basics down before plunging deep into the reference sections. The tutorial is fairly quick and it will show you how to create a new application and then add objects to it. There's a sprinkling of native SmartWare instruction which you can move through quickly if you know the software already. If you don't, you'll be introduced to some important concepts and commands but the best place for SmartWare instruction is with its own set of manuals.

If you have already developed SmartWare applications and you wish to convert them into RAD applications, refer to Appendix E.

NOTE: Because SmartWare is often used as an office automation tool, the term "*Office Automation*" is sometimes used to describe it. The two names are interchangeable. Similarly, RAD applications are sometimes referred to as "*Enhanced Office Automation Applications*" or just "*Enhanced Applications*".

Chapter 2: The Developer Tutorial

Introduction

In its most basic form, RAD is a development system that makes it easy to combine custom views, worksheets, documents, reports, and programs into a controlled application.

Intended only as an introduction to RAD development, this tutorial guides you through the steps needed to build a simple *Human Resources* application - it will give you an idea of what the ANGOSS Developer - and the systems created with it - can do. Note that some familiarity with SmartWare is useful but not necessary - to people who know SmartWare well, some parts of the step-by-step instructions can be accelerated.

Some points to note before starting:

- The example application, called RADHR, will be created in a directory of the same name. Approximately 1 Meg of disk space must be free for this application.
- Figures displayed in the tutorial are usually snapshots of screens running under Windows. While the interface does not look exactly the same on other platforms, the basic steps used in creating the RADHR application are the same.
- Non Windows users should add the ANGOSS bin directory to the *path* statement. If you are unclear on how to set the path statement, refer to your operating system's documentation for information on *Environment Variables*.

Part I - New Applications

Creating an Application

First of all, start SmartWare.

If you're running under **Windows**, turn off SmartWare's pulldown menus by selecting the command sequence:

```
Tools > Preferences > Terminate-Pulldowns
```

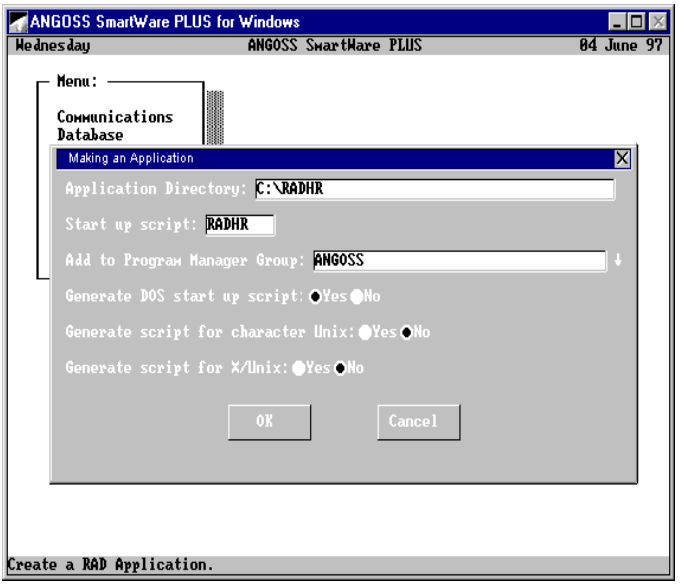
New applications can only be created from the standard SmartWare command list.

Select the commands:

```
Remember > Make-Application
```

This displays the Make Application dialog box.

Figure 2.1.1, The Make Application Dialog Box



As shown in figure 2.1.1, enter C:\RADHR into the Application Directory text box and RADHR into the Start up script text box. Unix users should substitute the "C:\" with an appropriate directory.

Windows Icon

If you are running Windows, enter a *Program Manager* group into the *Add to Program Manager Group* text box - the *Make Application* command will create the folder or group if necessary. In the example above, the RADHR icon will be placed in a folder called ANGOSS.

Command Line Script/Batch File

If you're not running Windows, the *Add to Program Manager Group* text box does not appear. On operating systems other than Windows, *scripts* or *batch files* are used to start RAD applications. The remaining three fields on the dialog box determine whether start up scripts will be created for these platforms.

Create an icon, script and/or batch file - you should at least create the start up device appropriate for your system. For instance, the dialog in figure 2.1.1 shows that both a Windows icon and a DOS batch file will be created.

When you have finished, select the **OK** button.

SmartWare will start copying files to the new application directory.

Starting the Application

Depending on the platform and the start up scripts created, the RADHR application can be started in several ways.

In Windows

Open the ANGOSS folder or group.

Select RADHR.

In DOS

First move to the RADHR directory. Assuming you are on correct drive, enter the following commands:

```
cd \radhr
radhr
```

NOTE: If you receive an *"out of environment space"* message, refer to your DOS manual on increasing this space.

In Unix

First move to the RADHR directory and then enter:

```
radhr
```

In X Window

First move to the RADHR directory and then enter:

```
xradhr
```

RADHR will prompt for a user ID. Enter the "*root*" user ID (this ID will be explained later):

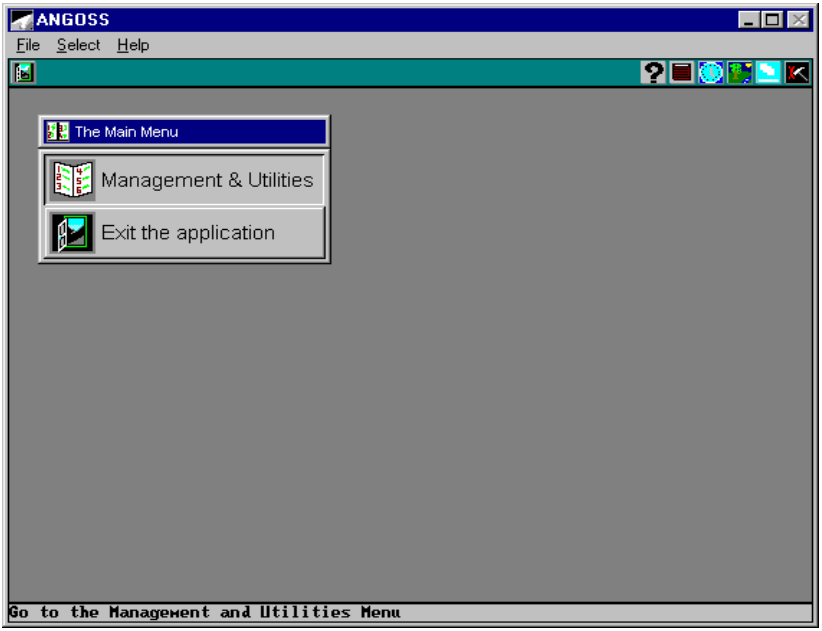
Enter the user ID: root

The first time into a system, the start up sequence will update system files. This will not happen on subsequent start ups.

Looking Around the System

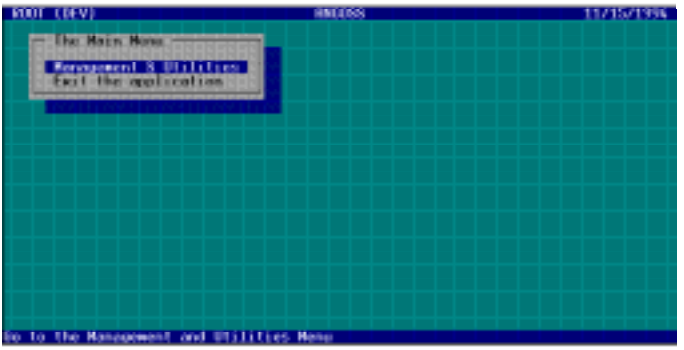
At this point you should have a screen that contains a menu, The *Main Menu*, with two items on it. In a *graphics environment* like Windows, your screen should look similar to figure 2.1.2.

Figure 2.1.2, The Main Menu in Graphical Mode



In a *character mode* environment, the screen looks different but, as you can see from figure 2.1.3, the makeup of the main menu is the same. Of the differences between the two modes, the existence of *pulldown menus* and *tool bars* in the graphics mode version are the most obvious.

Figure 2.1.3, The Main Menu in Character Mode



Graphical or not, the main menu is a "*navigation*" menu. You can select items from navigation menus with any of these four methods:

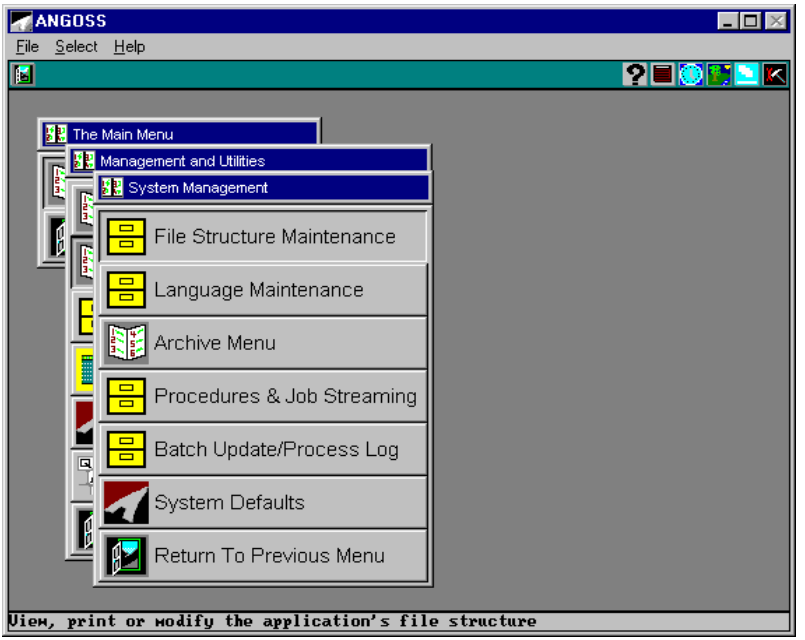
- Using the up/down cursor keys, highlight an item and press Enter.
- Using the space bar or backspace key, highlight an item and press Enter.
- Double click on an item with the left mouse button.
- Press the first letter of an item - if more than one item exists with the same first letter, the highlighter will cycle through those items.

Try this by selecting *Management & Utilities*.

A second menu should appear partially covering the main menu.

Now, select the *System Management* item. Your screen should look something like this:

Figure 2.1.4, The System Management Menu



Changing the System Title

Let's make our first change to RADHR by entering a more descriptive system title.

Select *System Defaults* from the *System Management* menu.

This time the menus are cleared and the screen contains four fields with a flashing cursor in the *System Title* field.

Delete *ANGOSS* from the *System Title* field. Replace it with *Human*

Resources.

After pressing **Enter**, the cursor moves to the next field.

We're not concerned with the other fields so, press the **Esc** key (or the right mouse button).

A dialog box then appears requesting whether or not we want to "*Recalculate SYS-DEF.VW each start up session?*". Recalculating the System Defaults each time adds to the start up time. Generally this recalculation is not necessary.

Press **Enter** or click on the **No** button.

When the navigation menus reappear, notice that the system title at the top of the screen has changed.

Ok, so far we have not actually used the Developer system. The Management & Utilities menu is basically a system administration tool set, but developers will undoubtedly use many of its functions - there will be more on this later.

Accessing the Developer System

One of the nice things about a RAD Application is that you build the system while running it. You can see and try out changes immediately.

To start, return to the *Main Menu* by escaping or selecting the *Return to Previous Menu* item (twice).

You should now have just the *Main Menu* on the screen with the highlighter on *Management & Utilities*.

To access the Developer:

Simultaneously press the key combination: **Ctrl A**

To prevent unauthorized access, the Developer system is password protected.

Enter the default password into the dialog box: **1621**

The Developer's menu takes a moment to appear - this loading pause as well as the password entry request only occur the first time the Developer is accessed in a session.

NOTE: the Developer password can be changed by a *Management & Utilities* menu item.

The Developer Menu

In a **graphics environment**, the Developer menu is a *pulldown* type that is located near the top of the window. Items on the pulldown are:

```
Insert Modify Delete Reposition Utilities Help
```

Pulldown menus are accessed with the mouse. Alternatively, the *Alt* key activates the menu, cursor keys or an item's first letter make a selection.

In **character mode**, the Developer menu is a command list located near the bottom of the screen. Items on the command list are:

```
Insert Modify Delete Reposition Utilities Help
```

The space bar and backspace keys cycle the highlighter through the menu items. Enter selects the highlighted item. A single mouse click or the first letter of an item can also be used.

Because of the rotary nature of these menus, command lists are also referred to as *ring* menus.

Inserting a Menu

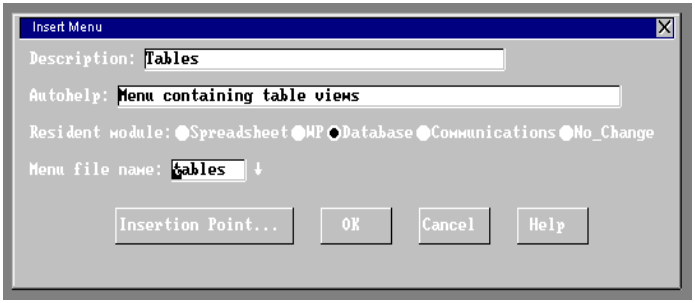
Now that the Developer has been accessed, we can add a new item to the Main Menu.

Select **Insert**.

Insert presents a list of items that can be placed on navigation menus. In this case, we want to add a sub menu to the Main Menu.

Select **Menu**.

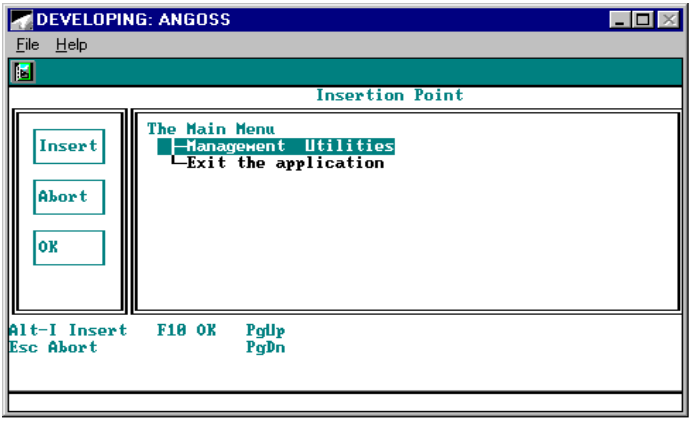
Figure 2.1.5, The Insert Menu Dialog Box



Enter the values shown in figure 2.1.5 for the *description*, *autohelp*, and *menu file name* fields. Since this will be a database level menu, leave the resident module selection on *Database*.

Now, select the **Insertion Point** button.

Figure 2.1.6, The Insertion Point Screen



In the menu map to the right of the command buttons, you can use the up/down arrow keys or the mouse to position the highlighter on an item.

Select the **Insert** button (or use the **Alt I** key combination).

Then select **OK** (or press **F10**).

You are then returned to the *Insert Menu* dialog box.

Select **OK**.

Finally, you are given the option to enter comments about the modification - these comments are stored in a log (refer to the reference chapters). You can also choose to finish or to continue working in the developer system.

For now, let's just select **Finished**.

And that's all there is to it. The Developer completes the command and then returns you to normal execution mode.

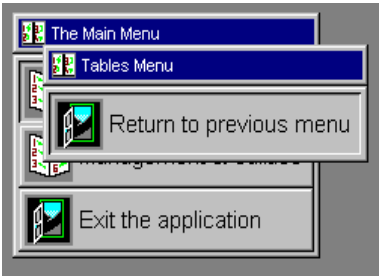
The *Main Menu* now contains the *Tables Menu* item we created. Notice that the auto-help text is displayed along the bottom of the application.

To open the *Tables Menu*.

Select **Tables**.

As you can see, the Tables Menu contains a single choice: *Return to Previous Menu*.

Figure 2.1.7, The Tables Menu



Inserting a Data View

The next step is to add a *data view* to this menu.

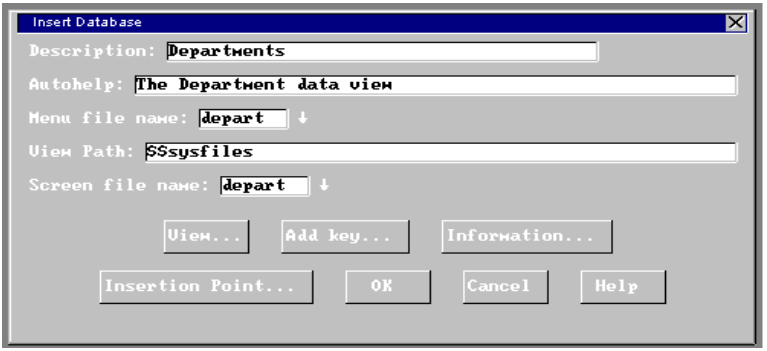
Again, access the Developer System with: **Ctrl A**

Select **Insert**.

As before, Insert presents a list of items that can be placed on navigation menus. In this case, we want to add a *Database* to the Tables Menu.

Select **Database**.

Figure 2.1.8, The Insert Database Dialog Box



Enter the values shown in figure 2.1.8:

Enter the Description: *Departments*.

Enter the Autohelp line: *The Department data view*.

Enter the Menu File Name: *depart*.

Enter the Screen File Name: *depart*.

Leave the view path field as *$$sysfiles*.

Now, select the **View...** button.

A second dialog box appears that gives you the option to create a view similar to one that has already been created however, we're going to create a view from scratch.

Select the **OK** button without entering a value in the text box.

The Custom-View Editor

The Developer system creates a custom view called *depart*, places the description at the top left, and then presents the SmartWare custom-view editor command list:

```
Option: Attach Create Delete Edit Move Input-Order Paint Replicate
```

NOTE: If you're already familiar with the custom-view editor you can skip this section - the data file, called *depart*, will contain two fields: *Code* (A8) and *Description* (A50).

Now, we must create a data file that stores information entered by users.

Select **Create**.

This presents the Create command list.

Select **Data-File**.

The custom-view editor now requests a data file name.

Enter the name *depart*.

Records in a data file can all have the same storage lengths regardless of the information's size within each record. This is referred to as *Fixed-Length*.

Select **Fixed-Length**.

Finally,

Select **No-Password**.

At this point, we're going to add fields to the data view and file. Using the cursor keys or the mouse, move the cursor to line 3, column 4 (the cursor position is shown below the command list).

Select **Create**.

Again, this presents the *Create* command list.

Select **Field**.

The custom-view editor now requests a field name.

Enter the name **Code**.

Usually, fields are placed on the data file.

Select **Data-File**.

This presents a new screen where the field's type, width, title placement, and other options are defined. In this case, we just want to use the default values.

Press **F10**.

Ok, now let's create one more field. Move the cursor down to line 5, column 5.

Select **Create**.

At the Create command list,

Select **Field**.

Now the field name.

Enter *Description*.

Place this field on the data file as well.

Select **Data-File**.

In the Field Definition Menu, leave the type as *Alpha* and move the pointer down one to the *Field Width* option.

Enter the number *50* (don't forget to press **Enter**).

Select **F10**.

We should now be at the custom-view editor's top level menu. Since these two fields are all we'll need for this data view, let's exit the editor.

Select **F10**.

This brings us back to the *Insert Database* dialog box.

Defining Key Fields

Creating *key fields* can make data file *sorts*, *queries*, and *searches* very fast when performed on those fields.

NOTE: Keys are immediately updated so that, when a file is ordered to a key and a record is added or modified, its position in the file is adjusted when the record is released.

RADHR provides the opportunity to define key fields from the *Insert Database* dialog box.

Select the **Add Key...** button.

A field prompter appears. The *Code* field is a natural key field choice since it will probably be searched or queried often. The cursor keys move the pointer around the prompter, make sure it points to *Code.*

Press **Enter** or simply double click the mouse on *Code*.

When the *Key Definition* menu pops up, just accept the defaults by pressing **F10**.

Now, select the **Insertion Point** button and insert the data view above the Return *to Previous Menu* item.

You are then returned to the dialog box.

Since that's all we need to do, select **OK**.

RADHR then takes a moment to update its internal information. When this is complete, you are again given the option to enter comments about the modification into the Changes log. For now,

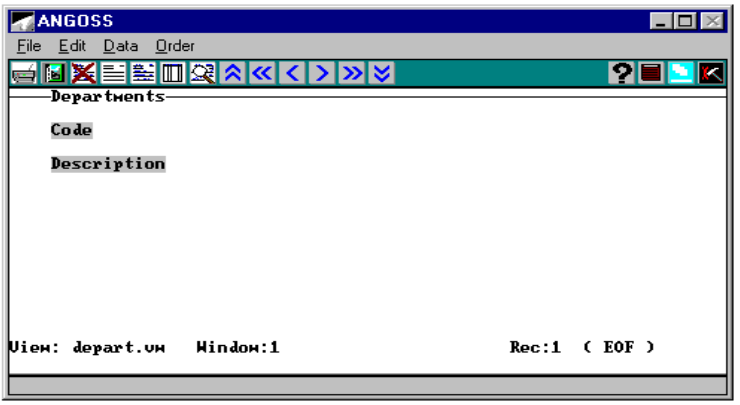
Select **Finished**.

RADHR returns us to the *Tables* menu with *Departments* as the first option.

The Data View

To see the view that we have created, select *Departments* from the *Tables* menu. Our screen should look something like this:

Figure 2.1.9, The Departments View



The screen in figure 2.1.9 is generically referred to as a *Data View*. Data views actually contain three components:

- Data File This file or files defines the record structure and holds information entered by users. In this case, we created the file as depart and added its fields in the custom-view editor.
- Custom-View Sometimes called a Screen, this file defines how the Data File (or Data Files - more than one can be attached) appears to the user. It too was created with the custom-view editor.
- Command Set This menu provides the user with an interface to the data. The Developer system created it with the most common database commands but you can customize it by removing or adding commands.

Data View Commands

For the moment, we are going to put the developer system aside (until Part II of the tutorial) and use the data view commands just as any user would. Remember, the Graphical interface uses pulldown menus while the Character interface uses command lists - in this case, both are referred to as a *View Menu*.

The first thing we need to do is add some records.

Graphical:

Open the **Edit** pulldown menu and select **Enter**.

Character:

Select **Enter** from the command list.

Because we're in data entry mode, the data view commands disappear. Instead, a *function key list* appears near the bottom of the screen and the cursor flashes in the *Code* field.

Enter *FOFFICE* into the *Code* field and *Front Office* into the *Description*.

A second blank record is displayed.

Enter *SHIP* into the *Code* field and *Shipping Department* into the *Description*.

A third blank record is displayed.

Enter *HOFFICE* into the *Code* field and *Head Office* into the *Description*.

Now add three or four of your own records.

When you have finished, press the **Esc** key.

To move through these records, experiment with the **F5** and **F6** keys and the **Ctrl Home** and **Ctrl End** key combinations. Graphical environment users can use the horizontal arrow icons on the toolbar (if you press and hold the left mouse button on one of these tools, you can read the autohelp message at the bottom of the window - moving the mouse along the toolbar without releasing the left button shows the current tool's *autohelp*).

Notice that the values following the word *Rec:* at the bottom right of the screen change as you move through the file. This indicates the current record number.

Ok, let's look at the remaining Data View commands. The following group of commands - *Browse, Delete, Krunch, Find, Update,* and *Return* - are straight forward. Try these commands out yourself. Note, the location of these commands for graphical users is listed in parenthesis.

Browse Mode (Data pulldown menu)

If you select the Browse command, records are displayed across the screen and the cursor keys become active. Browse toggles between the two display modes.

Delete and Krunch (Edit pulldown menu)

Delete de-activates or re-activates the current record. A red flag at the bottom right indicates the record's delete status. Note that deleted records are not actually removed from the data file until the *Krunch* command is used.

Finding Records (Data pulldown menu)

You can search for records with the *Find* command. This command provides a fast binary search through the file's key fields or a sequential search on any field.

Updating Records (Edit pulldown menu)

The contents of records can be changed with the *Update* command. The rules for moving around the record and editing fields are the same as those of the *Enter* command.

Return (File pulldown menu)

Use this command or **Esc** to exit the file.

The *Order* and *Query* commands require a little more explanation. In particular, these commands include a data file that retains definitions and indexes created by the user. The easiest way to explain this data file is to create a query first.

NOTE: Due to the number of options behind the *Print* command, it will be discussed later in a section of its own.

Querying the File

Files can be temporarily limited to a subset of records by executing a query. Query definitions determine what the subset will be. Let's try this now.

Graphical:

Open the *Data* pulldown menu and select *Query*.

Character:

Select *Query* from the command list.

RADHR presents the *Create Query* dialog box.

Enter *Office* into the *Definition Name*.

With this query, we will create an index having only office codes. The description should reflect this.

Enter *Access office codes only* into the *Description* field.

To create the query:

Select the **Define...** button.

Now the *query-by-example* (QBE) screen is displayed with the cursor flashing in the first field.

We want this query definition to select just the records that contain *OFFICE* in the *Code* field. To do this, we use a special symbol, the exclamation point, that means "*contains*" followed by the search data (refer to the ***Database Manual*** for complete instructions on query usage). For example:

Enter *!OFFICE*.

The cursor moves to the next field. We don't need to add a second search criteria so let's just quit.

Select **F10**.

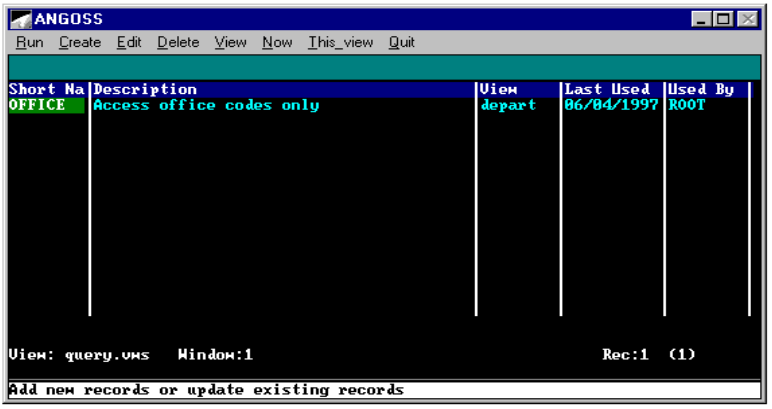
This takes us back to the *Create Query* dialog box.

We are done here, so select **OK**.

The Query Definition Data File

Now that a query definition has been created, we can look at the data file that maintains the query definition and index file names.

Figure 2.1.10, The Query Definition File



Normally, you can use the cursor keys to move to different records but, in this case, we only have one definition. Notice that the record shows the name of the query, its description, the view it was defined for, the date it was last used, the last user, and, if you move the highlighter all the way to the right, the last modified date and who modified it.

The pulldown menu or command list operations are:

- *Run* - Executes the currently highlighted query and returns to the active view (in this case, the *Departments* view).

  Note that, if the query has been run before, RADHR provides the opportunity to rerun it or use the previous run results - that is, order the file to the last index created by this query.

- *Create* - Similar to the steps we took earlier when creating the *OFFICE* query. Note that you can use an existing query definition as the basis for a new one.

- *Edit* - Edits the highlighted query.

- *Delete* - Removes the highlighted query definition and its reference in this data file.

- *View* - Displays the highlighted query definition.

- *Now* - Creates and executes a query but does not place a record in the Query Definition data file.
- *This_View* - Presents only those queries created for the current Data View.
- *Quit* - Returns to the *Data View* command list.

Let's run the query.

Select **Run**

The query executes and creates an index. When this is complete, a *Query Summary* box appears indicating the number of records searched and the number of matches found, as well as, the *index's* full name (the index name is taken from the query definition name).

Use **Esc** to clear this summary box.

Notice how, the *Department* view's status line (second line from the bottom) shows that the file is ordered to the *OFFICE* index. A quick look with browse mode on should reveal that the file displays only those records with the word *OFFICE* contained within the *Code* field.

Ordering the File

There are two ways that the sequence of records in a data file can be altered from the order in which they were entered.

- *Keys* can be used to immediately arrange the data file alphabetically (or numerically) based on the contents within each record's key field.
- *Sorts* too, are used to arrange the data file but are generally performed on non key fields or on a combination of fields. When a sort definition is executed, an *index* file is created (remember that a query also produces an index). Note that sorting data files with a large number of records may take a considerable amount of time.

The *Order* command presents a secondary menu containing the data file's key fields plus two other options: *Special_Sort* and *Physical*. Let's take a look at it now.

Select *Order*.

On the department file, *Code* is the only key field, so this Order command lists:

```
Code, Special_Sort, Physical
```

Physical Order

The *Physical* option is the simplest to explain. It simply returns the file to the order in which records were entered.

Select *Physical*.

If your data file was still ordered to the *OFFICE* query prior to executing the *Physical* command, you should now notice that the file is in its original order. The status line should also have changed.

Key Field Order

There are three sub-options when ordering to a *key field*. The first arranges the entire file based on the value in each record's key field. While it is also possible to limit the file to a range or even a single value, the department view is not the type of file we would do this with. Since that is precisely what the other two options do, we will discuss them in a more suitable Data View. For now, let's just order the whole file.

Select *Order*.

The only key field in this file is *Code*.

Select *Code*.

We want to place all the records in *key field order*.

Select *Whole_File*.

Notice that, when the command has completed, the view's status line shows that the file is in key field order.

Sort Index

Like the *Query* command, *Special_Sort* has a data file that retains the names of user-defined *sort definitions* and *indexes*. The steps are similar.

Select *Order*.

Select *Special_Sort*.

The RAD application presents the Create Sort dialog box. With this sort, we're going to order the data file to the description field.

Enter *Descript* into the *Definition Name* field.

Enter *Sort on department descriptions* into the *Description* field.

Following that:

Select the **Define...** button.

Now RADHR presents a field prompter pointing to the first field. With the cursor keys or mouse, move the pointer to the *Description* field.

Press **Enter**.

We can choose to have the sort place records in a lowest-to-highest or highest-to-lowest sequence. Let's change this to *Descending*.

Press the **D** key.

To finish, press **F10**.

We are then returned to the *Create Sort* dialog box.

Press **OK**.

Refer to the ***Database Manual*** for complete instructions on sort usage.

RADHR displays the *Sort Definition* data file and command set. Notice that this screen is identical to the *Query Definition* data file described earlier.

To execute the sort:

Select **Run**.

The sort executes and creates an *index*. As with the query earlier, the view's status line (second line from the bottom) shows that the file is ordered to the *DESCRIPT* index.

Exit the Application

Ok, let's completely exit the system - we'll restart it in Part II.

Select *Return* (on the File pulldown menu) to get back to the *Tables* Menu.

Select *Return to previous menu* to get back to the *Main Menu*.

Select *Exit the application*

Confirm that you want to return to the operating system (OS).

Part II - Data Views

Since you're now somewhat familiar with the ANGOSS Developer and RAD applications, descriptions of steps in the second part will be brief. This should make it faster to move through the tutorial.

Creating a Data View - Review

Follow the steps used in ***Starting the System*** at the beginning of the tutorial to restart RADHR. The *Main Menu* should appear with the highlighter on the *Tables Menu* item.

Move this down so that it highlights the *Management & Utilities* item.

Start the Developer with the **Ctrl A** key combination and enter the password, **1621**.

We're going to add an *Employees* file to the *Main Menu*. Considering that you already know how to create a Data View, these steps should be easy.

Select *Insert*.

Select *Database*.

Enter the description, *Employees.*

Enter the autohelp line, *The Employee data view*.

Enter the Menu file name, *emp*

Enter the Screen file name, *emp*

Select the **View...** button.

Don't create a similar view, just select **OK**.

At this point we're in the custom-view editor. First create the data file.

Select *Create*.

Select *Data-File*.

Enter the file name *emp*.

Select *Fixed-Length*.

Select *No-Password*.

Now create a field.

Select *Create*.

Select *Field*.

Enter the field name *Last Name*.

Select *Data-File*.

The *Field Definition* Menu should be displayed. Notice that the field width's default value of eight is too small for many surnames. Let's change it.

Move the cursor to *Field width* and enter *25*.

Press **F10**.

For now, let's quit and save the view.

Press **F10**.

Don't create a *key field*.

Select **Insertion Point...** and make the insertion below the *Tables Menu*.

Select **OK**.

Select **OK**.

Select *Finished*.

Our *Main Menu* should now look like this:

Figure 2.2.1, The Main Menu

Repositioning a Menu Item

It might make more sense if the most commonly used option is in the first position on the menu so, let's move the *Employees* view. There are small differences between the graphical and character versions in the way you go about moving menu items. In graphical mode, you select the menu item after the *Reposition* command is issued while, in character mode, you must highlight the menu item before the Developer system is started.

Graphical:

Start the Developer with **Ctrl A**.

Select *Reposition.*

Select *Employees.*

Character:

Highlight the *Employees* item.

Start the Developer with **Ctrl A**.

Select *Reposition*.

The remaining commands are the same.

Move the insertion point highlighter to *Tables Menu*.

Select *Insert*.

Select **OK**.

Don't bother with the *Changes Log*, select *Finished*.

Modifying the Employees Data View

The Employees data view currently contains just one field. To add more fields:

Select *Employees*.

Start the Developer with **Ctrl A**.

As usual, RAD's command list pops up but, when modifying a Data View in character mode, the view's command list is displayed too. This is because we may want to make changes to either the data view's *command list* or the *custom-view* and *data file* - remember, a data view consists of three parts.

This time, we want to modify the Employees custom-view (also referred to as *screen*) and data file.

Graphical:

Select *Modify*.

Select *Database*.

Select *Screen_and_DB's*.

Character:

Select *Modify*.

Select *Screen_and_DB's*.

Data views may have data files loaded in the background. In this case, we don't have any background files so it's irrelevant to specify whether we should unload them or not. Usually however, background data files are not unloaded.

Select **No**.

First of all, let's move the *Last Name* field down two lines.

Select *Move*.

Select *Item*.

Select *Field*.

The field prompter pops up and displays our only field.

Press **Enter**.

Move the cursor to Line 5, Col 5.

Press **Enter**.

Creating Employees Fields

Here is a list of the fields (not including the *Last Name* field) that we will be creating.

Employees Fields

Name	Type	Width
Employee Number	counter	5
Employment Date	date	10
First Name	alpha	12
Address	alpha	30
City	alpha	20
Telephone	alpha	12
Department	alpha	8
Salary	numeric	7

Ok, let's place the *Employee Number* field where the Last Name field used to be.

Employee numbers need to be unique and a good way to ensure that, is to use a counter field. This type of field increases in value each time a record is entered.

Move the cursor to Line 3, Col 5.

Select *Create*.

Select *Field*.

Enter the field name *Employee Number*.

Select *Data-File*.

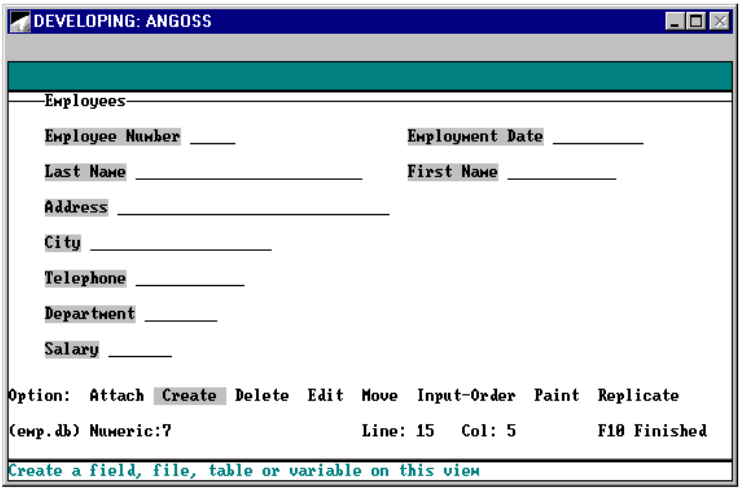
The *Field Definition* Menu should be displayed. We're going to change the field type to *Counter* and the field length to *five*.

With the space bar (or mouse), move the highlighter to *Counter*. Move the pointer to *Field width* and enter the number 5.

Press **F10** to exit the *Field Definition* Menu.

Using the field list above, create the remaining fields. Your custom-view might look something like this:

Figure 2.2.2, The Employees Custom View



Extended Field Options

Aside from just the *type*, *width* and *title placement*, a variety of other field options can be altered. To demonstrate this, let's edit the *Salary* field and take a look at the extensions offered by the *Field Definition* Menu.

Select *Edit* from the Custom-View Editor's first level command list.

Select *Field*.

In the field prompter, point to *Salary* and press **Enter**.

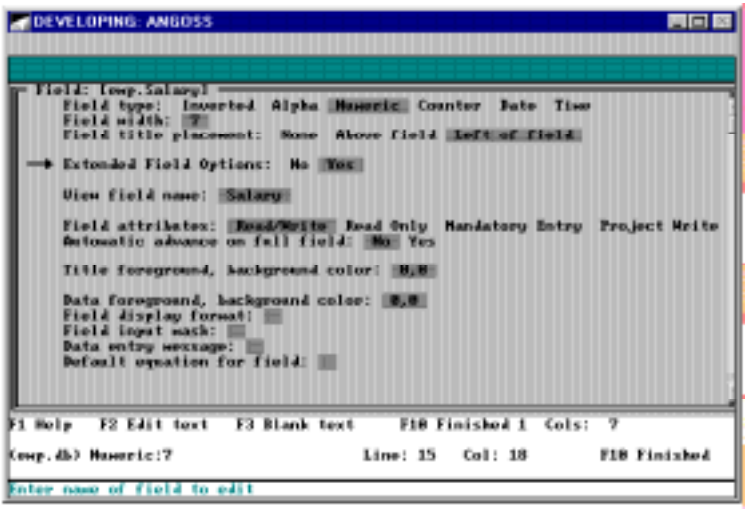
Leave the field's screen width at seven by pressing **Enter**.

Move the pointer down to *Extended Field Options.*

Using the **space bar**, move the highlighter to *Yes*.

The Field Definition Menu expands.

Figure 2.2.3, The Field Definition Menu



Note that extended field options affect the field on the custom-view but not on the data file (this is also true for the *Field Title Placement*). If the data file is also

attached to another custom-view, these options can be set to entirely different values. Think of the custom-view as a way to display fields and control input to the data file.

Items on the *Field Definition* Menu are, for the most part, self-explanatory (though you may wish to use *F1 Help* or refer to the ***Database manual*** for more information). In many cases, valid options can be listed with the **F6** key. One point regarding the *View Field Name*: you can change the field name on the custom-view but the original name is retained on the data file. While this may be necessary sometimes, take care changing the name because it adds a level of complexity that may later make it difficult to diagnose problems.

For the *Salary* field, we want to make two changes: the display format should be changed to a currency type; we'll also add an entry message. Continuing from our last steps.

Move the pointer to *Field display format*.

Press **F6** for a list of options.

Enter the characters *R$*.

Move the pointer to *Data Entry Message*.

Enter *Yearly Salary Amount.*

Press **F10** to exit the *Field Definition Menu.*

Data-File Menus

Now let's add a menu to the *Department* field. In this case, we want to access the codes from the department file. To do this we must first attach the *Department* data base to the *Employees* custom-view and then create a *Data-File* menu.

Select *Attach*.

Select *Data-File*.

From the file prompter select *depart.*

That takes care of attaching the *Department* file. Note that the Data-File menu we are about to create can only appear when a user updates the field containing the menu.

Select *Create*.

Select *Menu.*

From the field prompter select *Department.*

Select *Data-File*.

From the attached data file list select *depart*.

At this point we are presented with fields from the department file. These are the fields that will appear on the menu.

From the field prompter, use **F6** to select both *Code* and *Description*.

Press **F10**.

The value returned when a user selects a particular item from the menu must be specified.

From the field prompter, select *Code*.

The menu's size and location can be set. Let's just use the default.

Press **F10**.

As was stated earlier, data-file menus only appear when updating the field containing the menu. In fact, data-file menus also require the user to a press the key combination: **Alt F5**. To make users aware of this, let's edit the field and add an entry message.

Select *Edit*.

Select *Field*.

In the field prompter, point to *Department* and press **Enter**.

Leave the field's screen width at eight by pressing **Enter**.

Move the pointer down to *Extended Field Options*.

Using the space bar, move the highlighter to *Yes*.

Move the pointer to *Data Entry Message*.

Enter *Use Alt F5 to access the Department menu.*

Press **F10** to exit the *Field Definition* Menu.

Input Order

When updating a record, the input order determines the sequence for editing fields. By default the sequence that fields were created in is used. To change this, use the Input-Order command.

Select *Input-Order*.

Press the key combination **Alt R**.

Press **F10**.

The **Alt R** key combination forces the sequence to start at the top left field, move across the screen, then down to the next line and so on.

Ok, that's all we need to do to the Employees custom-view. Let's exit the custom-view editor and update the system information.

Select **F10**.

Select *Finished*.

Adding Keys

When a data view is first created, the *Create* dialog box provides an option to add keys. This doesn't happen when a data view is modified. Instead, the Developer system provides a separate command list option to modify keys.

Start the Developer system (**Ctrl A**).

Select *Modify* (graphical users must then select *Database*).

Select *Add_Key*.

Point at *Employee Number* and press **Enter**.

Press **F10** to accept the defaults.

Select *More Development* to keep the Developer system active.

Again, select *Modify* (graphical users must then select *Database*).

Select *Add_Key*.

Point at *Last Name* and press **F6**.

Point at *First Name* and press **F6**.

Press **Enter**.

Press **F10** to accept the defaults.

Select *Finished*.

Notice that the second time we added a key, we selected two fields. This was done to refine the ordering of records when more than one employee has the same last name. With keys of this type - that is multiple field keys - the first field is referred to as the *major* key field and the others as *minor* key fields.

Adding Employees

Add some records with the **Enter** command - any employee names will be fine - but add more than one employee per department. You should see that the *Employee Number* is automatically entered and incremented for each record. Remember, when entering data into the *Department* field, the **Alt F5** key combination presents a data file menu.

Before moving on to Part III, take some time to look at the *Find*, *Order*, and *Query* commands and their sub-commands.

Part III - The Print Menu

Printing

At the end of Part III, we'll see how developers can create sophisticated reports that are accessed but not modified by users. However, users are not limited to the developer's reports. They can create their own custom reports or send data to the spreadsheet and word processor modules. Let's begin with the *Print Menu*, employing it as any user would.

The Print Menu

If you have quit RADHR, restart it, go into the *Employees* data view, and select *Print* (*File* menu for graphical users). The print menu should look something like this.

Figure 2.3.1, The Print Menu

The first option, *Print Current Record*, does just that. It prints the record as it appears on the screen with both field titles and data. The remaining three options use a definition data file similar to the ones used by the *Query* and *Special_Sort* commands (discussed in Part I). Of course, the definitions themselves are quite different.

Custom Reports

To start with, let's create a custom report.

Select *Custom Report*.

Enter the *Definition Name* as: *resource*.

Enter the *Description* as: *Human Resource List*.

Select the **Define...** button.

The Report Generator

At this point, RADHR accesses the SmartWare report generator in order to create a report definition. Its command list appears as:

```
Option: Form Table Page Edit-Fonts Remove-Fonts
```

We're going to define a simple table report that lists the contents of each record's *Employee Number*, *Name*, and *Department* fields in column format. If you're already familiar with the SmartWare report generator, you can skip this and go on to the next section.

The first thing we need to do is to indicate that the report contains a table. This is accomplished through the Page Definition screen.

Select *Page*.

Move the *Page Definition* pointer to the last option, *Is there a Table on the Page.*

Move the highlighter to *Yes.*

A number of sub-options appear regarding the size and spacing of the table on the page. For our purposes, the default values are adequate.

Press **F10**.

Now let's define the table.

Select *Table.*

Select *Columns.*

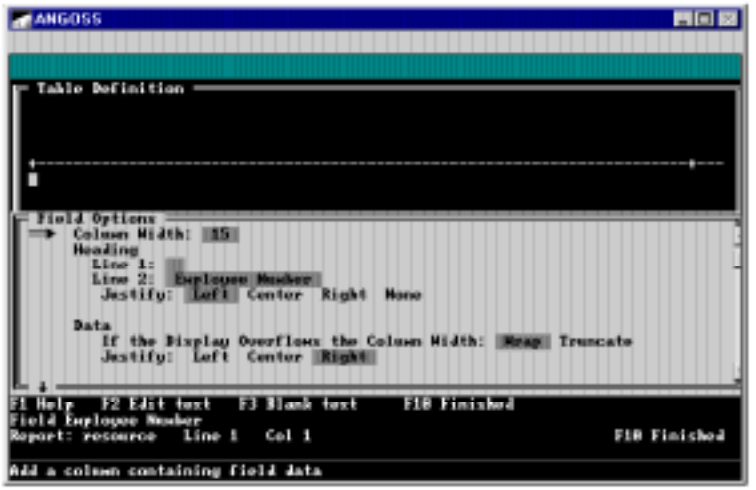
The *Columns* command list allows us to add and arrange fields, calculations and text on the report. In this case, we're simply going to add fields.

Select *Field*.

Select *Employee Number* from the field prompter.

That should produce the *Field Options* screen.

Figure 2.3.2, The Field Options Screen



Again, let's accept the default settings.

Press **F10**.

This field now appears on the dotted line; the field name above the line serves as the column title on the printed report; the numbers below indicate the column length and type (numeric). The cursor flashes on the last character of the field - let's move it to the right and add the next field.

Press the *right arrow* key twice.

Select *Field*.

Select *Last Name* from the field prompter.

Press **F10** to accept the *Field Option* defaults.

Continue this process by adding the *First Name* and the *Department* field. To finish:

Press **F10** three times.

Select **OK**.

This takes us to the *report definitions* view (similar to that of the query).

Select *Run*.

A dialog box appears with a number of options. You can send the report output to the printer or screen or disk. You can specify whether all records should be printed or just the results of *break point totals* - we did not define break points on this report, so *Totals Only* is not a valid choice. The remaining options determine the start/end pages and the number of copies to print - pressing **Enter** on each of these without typing any numbers results in default settings where one copy of the report, from first to last page, is printed.

The SmartWare report generator is an extensive subject covering *form*, *table*, and *combination* reports plus options within these reports such as *labels*, *break points*, *calculations*, *titles*, and *fonts*. To learn more about the report generator, refer to the ***Database Manual***.

Sending Data to the Spreadsheet

To send data from the current view to a worksheet:

Select *Send to Spread Sheet* from the print menu.

Enter the *Definition Name* as: *salary*.

Enter the *Description* as: *Table of employee salaries*.

Select the **Define...** button.

In the *Send* dialog box, the *Record Format* field determines the vertical or horizontal appearance of data in the worksheet. *Send Fields* determines the data that gets sent. If this is set to *List-Fields*, the desired fields should be entered into the *Select Fields* text box. The *Summarized* option uses SmartWare's *send crosstab* command. Refer to the ***Database Manual*** for instructions on defining crosstabs.

In this case, we're going to send the *name* and *salary* fields from all records in the file.

Leave the *Record Format* set to *Row* and move to the *Send Fields* option.

Select *List-Fields*.

Move to the *Select Fields* text box.

Press **F6** to produce the field prompter.

Move the pointer to the *Last Name* field.

Press **F6** (mouse users can click on the field).

Move the pointer to the *First Name* field.

Press **F6**.

Move the pointer to the *Salary* field.

Press **F6**.

Press **F10**.

Select **OK**.

We should now be back at the *Create Send* dialog box.

Select **OK**.

This takes us to the *send definitions* view (again similar to that of the query).

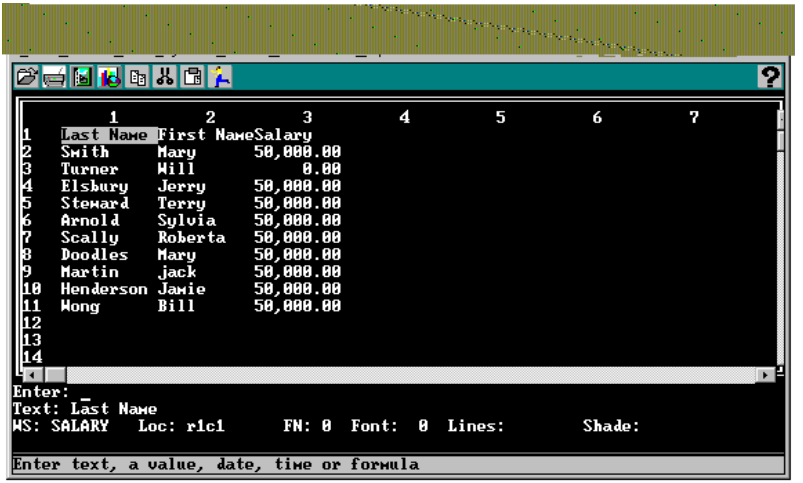
Select *Run*.

Select **OK**.

Select **OK**.

Your worksheet should look something like this:

Figure 2.3.3, Send Worksheet



You should now be in the spreadsheet module's entry mode and are free to make changes to the worksheet. To return to the print menu:

Graphical:

Select *Exit* from the *File* pulldown menu.

Character:

Press **F10**.

Select *Resume_application*.

Sending Data to the Word Processor

This process, called *Mail Merge*, creates a "*form letter*" for each record in the current view. The idea here is that fields (such as *First Name*) can be substituted into a generic letter.

To do this:

Select *Mail Merge* from the print menu.

Enter the *Definition Name* as: *memo*.

Enter the *Description* as: *Memo for upcoming meeting*.

Select the **Define...** button.

In the Create Merge dialog box:

Press **F6** to produce the field prompter.

Move the pointer to the *First Name* field.

Press **F6** (mouse users can click on the field).

Move the pointer to the *Last Name* field.

Press **F6**.

Press **F10**.

Select **OK**.

We should now be back at the *Create Merge* dialog box.

Select **OK**.

This takes us to the merge definitions view.

Select *Run*.

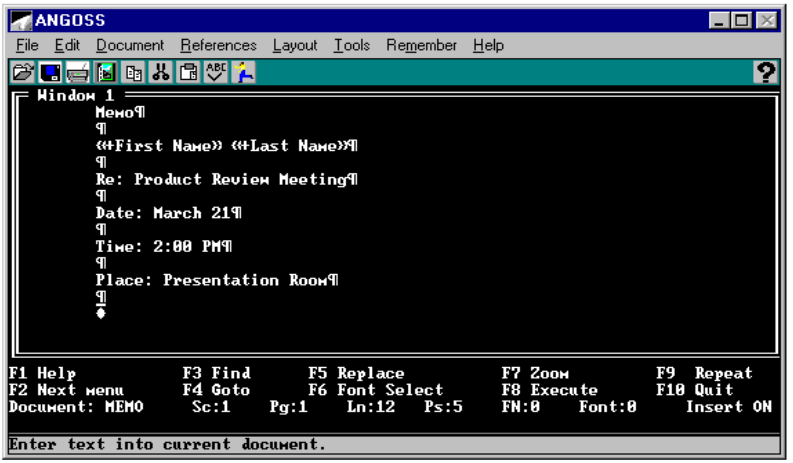
Select **OK**.

You should now see the selected field names within an otherwise blank document. These fields are surrounded by double arrow characters (which indicate that they are *Merge Variables*) and prefixed with plus signs. The plus sign is actually a merge variable option that causes the variable to be omitted if the input string is empty.

Merge variables can be placed anywhere in a document. As well, merge variables can be reused. To manually insert merge variables, use the **Ctrl J** key combination to enter the merge start symbol, type a valid variable name, and press **Ctrl K** to complete it. For a complete description of merge variables and the entire merge process, refer to the ***Word Processor*** manual.

Enter whatever text you wish to have in your memo. For example:

Figure 2.3.4, The Merge Document



When you have finished creating the form letter:

Select *Exit* (**F8** in character mode).

RADHR then provides the opportunity to print the letters immediately or leave it for later. Print output can be sent to the printer or to a file.

Since this *mail merge* is now set up, it can be used at any time without modification - simply select it and specify that you do not want to modify the definition.

Note that a *mail merge* uses only those records in the current index.

Creating a RAD Report

While developers can insert reports at any database menu, one common place to do this is on a data view's print menu. The type of report we're about to look at is fixed by the developer (i.e., users cannot modify the report). It has the ability to combine a *report definition*, an *input screen*, a *query* and a *sort*, as well as, an *SPL program*.

Furthermore, options that are determined at run time, such as the number of copies to print, can be preset or prompted.

To begin, move the highlighter to the last item on the print menu and then start RAD.

Select *Insert*.

Select *Additional*.

Select *Print*.

Select *Report*.

In the *Insert Report* dialog box:

Enter the *description* as: *Department Salaries*.

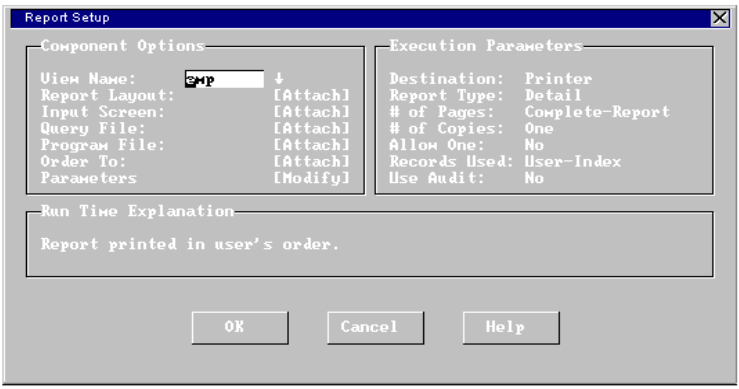
Enter the *Autohelp* line as: *Salaries paid to employees by department*.

Enter the *Default component name* as: *sal_rp*.

Select the **Report Components** button.

The Report Setup dialog box is then displayed.

Figure 2.3.5, The Report Setup Dialog Box



Basically, this dialog box provides access to the report components and the execution parameters. The first step here is to create a *Report Layout*.

Select the **[Attach]** button beside *Report Layout*.

The *Attach Report Layout* dialog box defaults the layout file name to *sal_rp*.

Select *New*.

This takes us to SmartWare report generator (we used it earlier in the *Custom Reports* section). As before, we will indicate that the report contains a table.

Select *Page*.

Move the *Page Definition* pointer to the last option, I*s there a Table on the Page.*

Move the highlighter to *Yes.*

Press **F10** to accept the remaining defaults.

Now let's define the table.

Select *Table.*

Select *Columns.*

Select *Field.*

Select the *Department* field.

Press **F10** to accept this column's defaults.

Move the cursor a space beyond the *Department* field.

Select *Field.*

Select the *Last Name* field.

Press **F10** to accept this column's defaults.

Move the cursor a space beyond the Last Name field.

Select *Field.*

Select the *First Name* field.

Press **F10** to accept this column's defaults.

Finally, move the cursor a few spaces beyond the *First Name* field.

Select *Field.*

Select the *Salary* field.

Change the *Column Width* to *9*.

Press **F10** to accept this column's settings.

Press **F10** to complete the column setup.

OK, that takes care of the columns. What we need now is a *breakpoint* placed on the *Department* field. This breakpoint will be used to sum the salaries of each person within a department (that's why we added more one employee per department).

Select *Breakpoints*.

Select *Add*.

The *Department* field should now be highlighted.

Press **Enter**.

Press **F10** to accept the defaults in the *Totals Option* box.

Using the cursor keys, move the highlighter to the *Salary* field.

Press **S** to sum that field.

Press **F10**.

Press **F10** to complete the Breakpoints.

You can add a *Grand Total* to sum all the salaries and a report title if you like. For now, let's just quit the report generator.

Press **F10**.

Press **F10**.

That brings us back to the *Report Setup* dialog box. For our breakpoint arrangement to work, we must insure that the file is ordered correctly.

Select the **[Attach]** button beside the *Order To* item.

The *Attach Order To* dialog box defaults the sort file name to *sal_rp*.

Select *Sort*.

The field prompter should now be displayed.

Select the *Department* field followed by the *Last Name* field.

Press **F10** to complete the field selection.

Press **F10** to accept the *Sort Definition* defaults (*Ascending Order*).

Again, that brings us back to the *Report Setup* dialog box.

Select the **[Modify]** button beside the *Parameters* item.

The *Execution Parameters* dialog box displays a number of options that affect the report during run time. Of these options:

Set the *Destination* field to *Ask-User*.

Set the *Records Used* field to *Whole-File*.

Select **OK** to close the *Execution Parameters* dialog box.

Select **OK** to close the *Report Setup* dialog box.

Finally, we return to the *Insert Report* dialog box.

Select an insertion point and complete the dialog box.

Select *Finished* to complete the Developer session.

Executing a RAD Report

Let's test our report. We should now be on the *Print* menu.

Select the *Department Salaries* item.

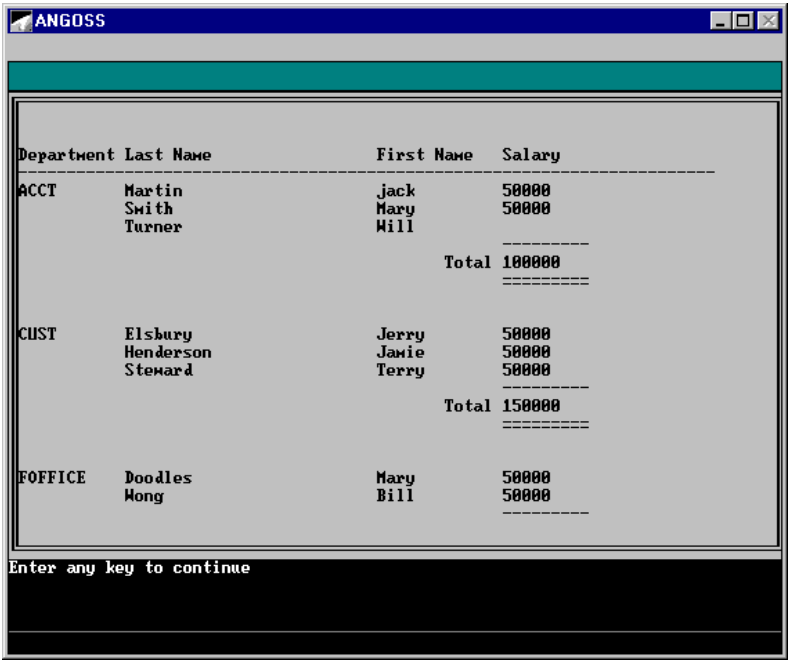
A dialog box pops up that tells us the number of records that will printed. We can also select a destination for the report. When testing, it's a good idea to send the output to the screen or text screen. The text screen option is much faster but it does not display the report exactly how it appears when printed.

Select *Screen* or *Text-Screen*.

Select **OK**.

Text screen output of the report should look similar to this:

Figure 2.3.6, Text Screen Output



To summarize, the following stages describe how the report worked at run time:

- **Whole File** - The *Employees* data view was ordered physically to insure that all records would be included in the report.
- **Sort** - The file is then sorted first by the *Department* field and then by the *Last Name* field.
- **Destination** - The data is sent to the *report layout* program to be displayed on the screen.
- **Layout** - The *report layout* program displays each record in the current order. When a new value for the *Department* field is encountered, records

from the previous group are summed together based on the values in the *Salary* field. This total is displayed before the next department group is started.

Now, exit out of the RADHR application and take a break.

Part IV - System Administration

In Part IV, we'll set aside the Developer system and investigate the application's administration capabilities. Generally, access to these tools is provided by the *Management & Utilities* menus. Before we start, you may want to backup or copy the RADHR directory tree.

To begin:

Start *RADHR*.

Select *Management & Utilities*.

User Modes

RAD applications can be set up so that access is controlled in groups. This is useful in multi-user situations where, for example, one group of users is limited to an *order entry* menu while another group is confined to a *shipping* menu.

By default, RAD applications come with three user modes: *Developer*, *Administrator*, and *User*. So far, we have been using the developer mode (since it's the only mode that gives us the ability to build an application).

Let's take a look at this now.

Select *User Management*.

Select *User Mode Maintenance*.

Enter the password, **1621**.

The Screen should contain something like this:

Figure 2.4.1 User Mode Maintenance

Here, the *Main Menu*'s items are listed under *Description*. The columns beside this show the item access for each user mode except *DEV* mode (developers have access to everything). The slashes in the *User* column indicate that anyone belonging to the group *User*, will not see the *Manage & Utilities* item on the *Main Menu* and therefore will not have access to that menu.

To add a new User Mode:

Move the highlighter to the *User* column (any row will do).

Select *Usermode*.

Select *Add*.

In the *Add Usermode* dialog box:

Enter the *Short name* as: *Group*.

Enter the *description* as: *Level 1 Users*.

Select **OK**.

Select ***Yes*** in the backup warning message.

You should now have a new user mode with the same capabilities as *User*. In addition to the *Management & Utilities* item, let's disable the *Tables Menu.*

Move the highlighter into the *GROUP* column and the *Tables Menu* row.

Select *Access*.

Select *Disable*.

On the *Access* menu, you may have noticed that items can be *Enabled* or *Passwords* attached. In fact, aside from access limitations, there are a number of other menu

aspects that can be modified by user mode. The *Defaults* menu allows you to change *menu types, colors, help exposure*, a *user mode's starting menu* (it doesn't have to be the *Main Menu*), and more. Menu aspects can be changed globally, on a per-menu-basis, or by item. If you're a developer, you can also change menu and item information through the *All Users* command.

Ok, that takes care of the *Main Menu*, but what about the other menus? There are two ways to go about making user mode changes to other menus: one way is to select *Next Menu* from the *File* menu, the second is described below.

Select *File*.

Select *Quit*.

Select *Return to Previous Menu*.

Select *Return to Previous Menu* (returns to the *Main Menu*).

Select *Employees*.

Press the **Ctrl U** key combination.

Note that the *User Mode Maintenance* screen can be accessed with the **Ctrl U** macro on any navigation or view menu.

We're going to disable the *Krunch* command for *GROUP* users.

Move the highlighter into the *GROUP* column and the *Krunch* row.

Select *Access*.

Select *Disable*.

Select *File*.

Select *Quit*.

Finally, there is one more *user mode* command we should try.

Exit the *Employees* view and return to the *Main Menu*.

Press **Ctrl U**.

Select *Usermode*.

Select *Switch*.

Select *GROUP*.

Select **OK** in the message dialog box.

Select *File*.

Select *Quit*.

We've temporarily switched to the *GROUP* user mode. As you can see, the *Main Menu* is restricted to the *Employees* and *Exit* items. If you check out the *Employees* view, you will not find the *Krunch* command.

To switch back, perform the same steps but select *DEV* rather than *GROUP*.

Users

In multi-user situations, individual users have their own passwords and working directories. So far, we've been using the ROOT user which belongs to the DEV user mode.

To set up a user:

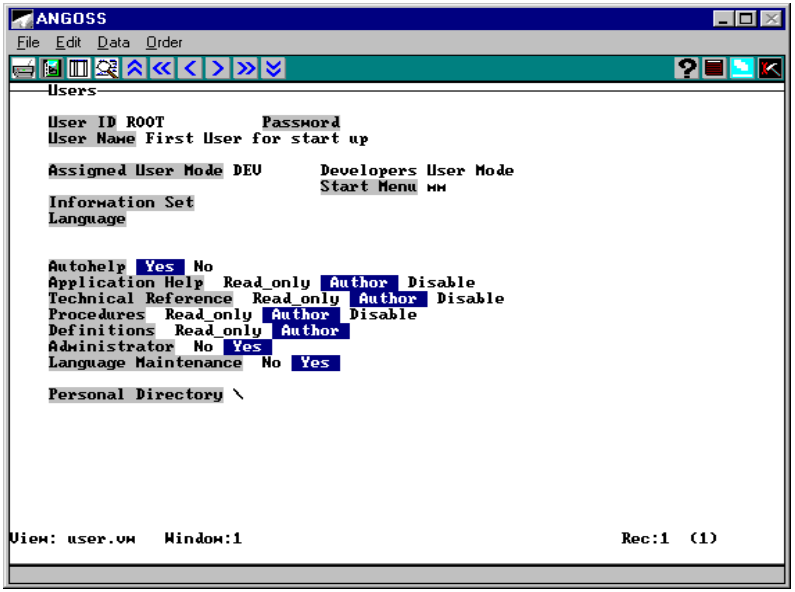
Select *Management & Utilities*.

Select *User Management*.

Select *Users*.

The screen should look something like this:

Figure 2.4.2, User Management



This is actually a data view which currently contains only the *ROOT user* record. Let's add a hypothetical administrator named *Warren.*

Select *Enter* (graphical users will find it on the *Edit* pulldown menu).

Now, enter the following data into these fields:

Field Data

User ID: *WARREN*

Password: *555LT*

User Name: *Warren Piece*

In the *Assigned User Mode* field:

Press **Alt F5**.

Select *ADMIN*.

The *Information Set* determines the default paths used by the system for this particular user (currently there is only one set of paths). In the *Information Set* field:

Press **Alt F5**.

Select *standard*.

Because we haven't set up any language information, skip the *Language* field. Select the following options for these fields:

Field	Select
Autohelp	Yes
Application Help	Author
Technical Reference	Read_only
Procedures	Author
Definitions	Author
Administrator	Yes
Language Maintenance	Yes

This group of fields determines whether a user can create and/or access *help files*, *procedures*, and *definitions* (such as user defined queries), as well as, perform administration and language maintenance tasks. Since Warren is an administrator, it seems likely that he will need the above capabilities.

The *Personal Directory* field defaults to the *user ID name*. This is a reasonable value.

Press **Enter**.

If the directory *\WARREN* does not exist, RADHR requests permission to create it.

Select *Yes*.

That takes care of setting Warren up as a user in RADHR. Exit the view and return to the *Management & Utilities* menu.

At this point, each time he starts the system, he must login with his user ID and then enter the password. You can however, have the login occur automatically by setting the environment variable, *ANGIIUSR*, to his user ID. Refer to your operating system manual for information on these variables. Note: in Unix, the environment variable is *USER*.

Hierarchy Diagram

With small applications like RADHR, it's not difficult to remember the overall structure of the system. However, in large systems, a tool like the *Hierarchy Diagram* can be useful.

We'll take a brief look at it despite the size of our application. From the *Management & Utilities* menu:

Select *Menu Hierarchy Diagram*.

Because it takes time to create, the diagram is not automatically kept up to date. To bring it up to date:

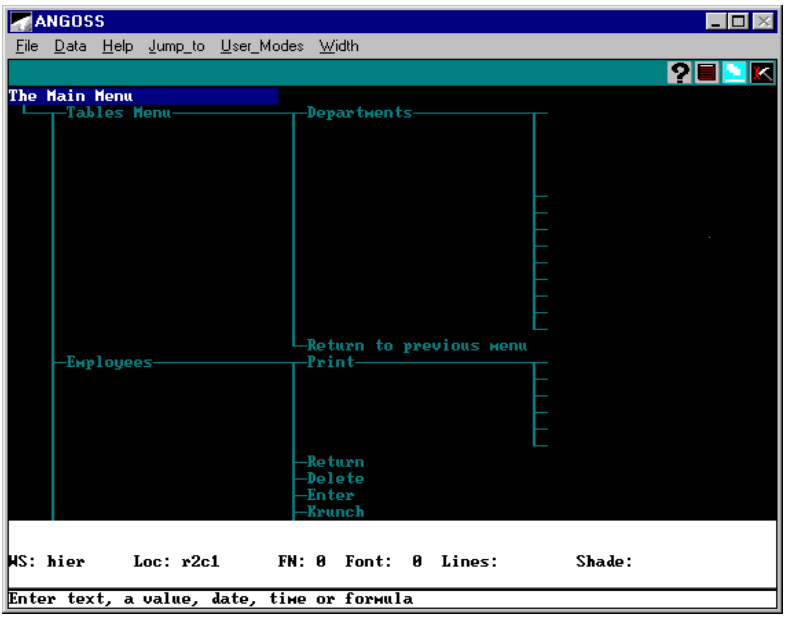
Select *File*.

Select *Generate*.

Accept the *Generate* dialog box defaults and select **OK**.

The process takes a moment and then redraws the screen with the items we have added.

Figure 2.4.3, Menu Hierarchy Diagram



With the cursor keys, you can move to other items in the diagram. Try experimenting with menu selections. Among other things, there are tools that allow you to jump directly to an item, view a menu as it appears in the application, display the *action/object/function* of all items, and demonstrate the direct route from the top menu to the currently highlighted item.

When you have finished experimenting:

Select *File*.

Select *Return*.

Procedures and Job Streaming

These two items provide an automated way of stepping through application items. *Procedures* direct a user through each step, allowing them to interact but not stray from the pre-set course. *Job Streams* work a little differently: all interaction occurs before items are run, thus providing a mechanism for unattended processing.

Since the item types used in *procedures* is not limited, as it is with *job streams*, let's setup and run a simple procedure.

Procedure Setup

From the *Management & Utilities* menu:

Select *System Management.*

Select *Procedures & Job Streaming*.

Each *Procedure* and *Job Stream* contains a record in this file.

Select *Enter* (*Edit* menu in graphics mode).

Enter the *Short Name* as: *test.*

Enter the *Description* as: *Test Procedure*.

Select *Procedures* as the type.

Skip the *Users* box and move to the *User Modes* box below it.

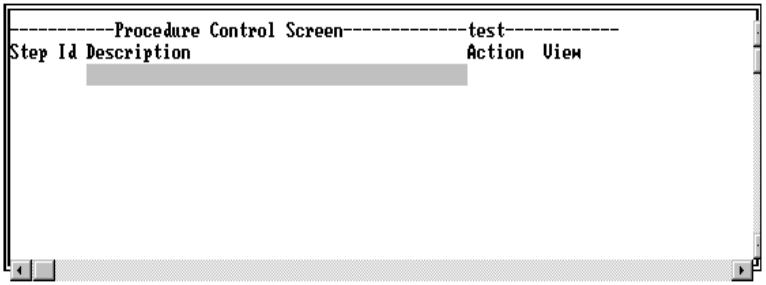
Enter *DEV*.

Save the record.

Select *Steps* (*Edit* menu in graphics mode).

This produces the hierarchy diagram with a *Control Screen* placed on top of it.

Figure 2.4.4, Procedure Control Screen



To Enter the first step into the Control Screen:

Select *Insert* (*Edit* menu in graphics mode).

Select *Access-Application.*

Move down through the hierarchy to *Tables Menu.*

Move right to *Departments*.

Move right to *Print*.

Move down to *Update.*

Choose *Select* from the menu.

For the second step:

Position the highlighter on step 2.

Select *Insert* (*Edit* menu in graphics mode).

Select *Access-Application.*

Move down to *Employees.*

Move right to *Print*.

Move down to *Browse.*

Choose *Select* from the menu.

For the final step:

Position the highlighter on step 2.

Select *Insert* (*Edit* menu in graphics mode).

Select *Access-Application.*

Move down to *Employees.*

Move right to *Print.*

Move down to *Update.*

Choose *Select* from the menu.

Choose not to make it the same step.

Select *Save* (*File* menu in graphics mode).

Select *Return* (*File* menu in graphics mode).

Now, return back to the *Management & Utilities* menu.

NOTE: Setting up a *Job Stream* is nearly identical. The difference lies in the fact that some items cannot be placed in a *Job Stream* because those items require direct interaction.

Procedure Execution

While it is possible to insert individual Procedures or Job Streams onto navigation menus, in this case, we'll just execute it from the view.

Select *Run a Procedure or Job Stream*.

Choose *Select.*

The *PROCEDURE CONTROL SCREEN* pops up between each step, giving users the ability to *continue, back up, skip, attain help,* and *abort.*

Summary

That's it. We've covered many of the **basic aspects** of developing a RAD application, but there's a lot more to experiment and create with. Try adding *spreadsheet views* and *document views* to the application; if you have some programming skills, the *audited program* option automates the task of tracking processes; for organizing

large amounts of data, the *archiving* feature may be valuable; add *help files* to the application and automatically generate documentation...

For a complete list of items that can be inserted on menus, refer to ***Chapter 3 - Developer System Menus***.

Chapter 3: Developer System Menus

Accessing the Developer

The Developer system can be accessed with the **Ctrl A** key combination when running a RAD application. In order to prevent unauthorized access, a Developer System password is requested. By default, the Developer system password is **1621**.

Note that once the password has been entered, the RAD application assumes the user has developer status and does not request the password again during that session.

To change the Ctrl A access key, edit the associated value in the ANGOSS.CFG file with a text editor. Passwords can be modified from Quick Key Access Passwords in the administrator submenu.

The following information is reference material. If you are unfamiliar with the Developer System or with RAD applications, we recommended that you at least *read* through the tutorial.

First Level Menu

The Developer presents the following menu:

```
Insert Modify Delete Reposition Utilities Help
```

These commands are the basic tools used to expand and modify an application. For the most part, they operate on the current menu or object (such as a data view). To operate the developer system on a different menu or object, move to that item prior to invoking the Developer System.

Insert	Add items to the current menu or object.
Modify	Change the specifications of a selection, menu, or objects associated with the current menu.

Delete	Removes an item from the menu. If any items (such as views, reports, or programs) exist which are related to the deleted selection, the Developer requests permission to delete each one of these as well.
Reposition	Temporarily deletes an item. The next time the Developer System is invoked, the Developer provides the option to re-insert the item, effectively moving it to a new location on the menu or to an entirely different menu.
Utilities	Accesses another menu containing various system development utilities.
Help	Provides help on the Developer System.

Inserting an Item

When Insert is selected, the Developer presents a list of item types that can be added to a menu. After choosing one of these types, a dialog box appears. The dialog box for each type contains both common elements and type-specific elements.

For example, all insert dialog boxes contain a description text box and an autohelp text box. The text entered into the description field is used as the menu item name. Autohelp appears at the bottom of the application window whenever the menu item name is highlighted.

Further, all insert dialog boxes contain an Insertion Point button which allows you to place the new item anywhere on the current menu.

A description of each item type follows. Note that some items are found under the Additional option under specific circumstances. For example, insert a Totals action appears on the Additional option only when a data view is active.

Menu

Location: Insert
Scope: All

Sub menus are added with this command.

Menus can reside in any of the four SmartWare modules regardless of the parent menu's location. Switching modules can take time therefore, if a menu contains items from a single module, it may be more efficient to select that module. If the menu has choices that access multiple modules, consider using the No_Change option. This minimizes module changes by not changing unless a menu option requires it to.

ANGOSS identifies menus by its unique file name. Note that only legal filename characters may be used.

See also:

Inserting a Menu, page 23

Menus, page 107

Database

Location: Insert

Scope: All

Inserts a data view - that is, a custom-view, a data file, and a view menu.

Along with the standard elements, the Insert Database dialog box requires a menu file name, a path for the view, and a screen (view) file name. It also includes buttons that lead to the SmartWare custom-view editor, the add key prompter, and the data view information screen.

The general approach for creating a data view is to fill in the dialog box text fields and then select the View button. This brings up the custom-view editor where data file(s) and fields are created. When this process is complete, key fields are added. If you are not familiar with the custom-view editor or the key field concept, please refer to the SmartWare Database manual.

Adjusting aspects of the data view's appearance and control is accomplished with the Information button. Refer to the Screen Information section in Chapter 4.

It is possible to insert a data view on another data view's view menu. If this is the case (i.e., you've invoked the Insert Database option from an existing data view), the Insert

Database dialog box will request information for a database-to-database link definition.

See also:

Inserting a Data View, page 26

Data View, page 110

Spreadsheet

Location: Insert
Scope: All

Inserts a spreadsheet view - that is, a worksheet and a view menu.

Along with the standard elements, the Insert Spreadsheet dialog box requires a menu file name. It also includes a button that leads to the spreadsheet information screen.

It is possible to insert a spreadsheet view on a data view's view menu. If this is the case (i.e., you've invoked the Insert Spreadsheet option from an existing data view), the Insert Spreadsheet dialog box will request information for a database-to-spreadsheet link definition.

See also:

Spreadsheet View, page 117

Wordprocessor

Location: Insert
Scope: All

Inserts a document view - that is, a word processor document and a view menu.

Along with the standard elements, the Insert Wordprocessor dialog box requires a menu file name. It also includes a button that leads to the document view information screen.

It is possible to insert a document view on a data view's view menu. If this is the case (i.e., you've invoked the Insert Wordprocessor option from an existing data view), the

Insert Wordprocessor dialog box will request information for a database-to-wordprocessor link definition.

See also:

Document View, page 122

Program

Location: Insert
Scope: All

Inserts an SPL program or function call.

While it is possible to create an application in which no programming is required, there may arise situations where specialized functions must be added to an application. For details on SPL programming, refer to the SmartWare Project Processing manual.

There are three branches to the Program option:

Execute

This is a straight-forward SPL program. It is well suited to simple, quickly-written programs. Note that sophisticated programs, particularly critical ones, should be audited (see below).

Audited

Whenever an audited program is executed, it's recorded in the Batch Update/Process Log file. An audited program has both an input screen, displayed before the program is run, and an audit/error report, displayed when the program has completed. Both the input screen and the report are stored with the Process Log record.

An additional feature of audited programs is that they may be included in a job stream, allowing an end-user to initiate a number of processing routines in series. Note, however, that audited programs are only available in the data base module.

Function Call

Any active function that accepts one argument can be called. Typically, these functions reside in an Application Library or a library bound to a menu.

The Insert Execute and Insert Audited dialog boxes contain the program name field while the Insert Function Call dialog box contains the function name field.

The Parameter field, too, is different. In the case of programs, the parameter is stored in the $$fct public variable which can then be used to determine a specific branch within a common program. This is how a single shared program can be used from multiple menu items. With function calls, the parameter is the value of the argument.

The program dialog boxes also contain a Program button which provides access to the SPL editor. If the specified program does not already exist, a template is generated. For audited programs, access to the input screen editor is also provided.

See also:

SPL Programming, page 149

OS_Call

Location: Insert
Scope: All

The OS Call creates a command or menu selection that calls the Operating System (DOS or Unix). Such a call can be used to perform commands or to invoke non-ANGOSS applications. In DOS applications, you may choose to have SmartWare removed from memory before running the OS command. By unloading SmartWare first, memory restrictions are effectively removed, however, execution of the OS command will be faster when SmartWare is left resident.

There are two OS Call options:

Command

SmartWare remains resident. A maximum of one OS command may be used.

Script

Inserts and creates an OS script (batch file). In DOS, this will unload SmartWare from memory before executing the script.

This option allows numerous lines to be used in specifying the commands to execute. Third-party software may be called freely from within the script. When execution of the script is complete, SmartWare is reloaded and the user is returned to the menu, record, or cell that had been on the screen when the script was executed.

The dialog box's Script button takes you to the text editor, where you may create or edit the script.

Text_File

Location: Insert

Scope: All

Creates a command or menu selection that causes the contents of a text file to be displayed on the screen.

In general, a text file can be used any time you want to quickly display information to the user. During the prototyping phase of development, text files could explain features which have yet to be included. Text files may also be useful in a finished application. Since the text file name is actually an expression, the contents of database fields or spreadsheet cells can be used as part of the name.

The dialog box's Expression field determines the name of the file. If the same text file is to be displayed every time the choice is selected, then the file name and extension must be enclosed in quotation marks (e.g., "message.txt"). If the command is issued from a data view, you may enter an expression using fields from the view (e.g., str([Order])|".txt"). Either way, this expression must evaluate to a string.

The Text-Editor button allows you to create the initial file.

The Access mode dictates which users will be able to modify the text file when it is displayed.

Normal mode means that only developers (users whose user mode is "Dev") will be given the ability to edit the file.

View_Only mode will not even allow developers to edit the file.

Edit_Always grants editing privilege to all users. When editing privileges are granted, the F7 and F8 keys are active, allowing access to the Text-Editor and the Word Processor respectively.

New_Application

Location: Insert
Scope: All

Creates a command that runs another RAD application. When the user exits this application, he or she will be returned to the menu, record, or cell that had been current when the New_Application action was selected.

Note that the end-user must be registered in both systems with the same User ID, since this information is automatically transferred.

Archive

Location: Insert Additional
Scope: Database Navigation Menus

Inserts an archive job on the current menu. This option only appears when a navigation menu in the database is active. If you do not specify an archive job in the Insert Archive dialog box, the resulting item will simply execute the Run an Archive Job selection from the Archive Menu.

See also:

Archiving, page 138

System_Defaults

Location: Insert Additional
Scope: Database Navigation Menus

Inserts the System Defaults view on the current menu. This option only appears when a navigation menu in the database is active.

See also:

System Defaults, page 136

Print

Location: Insert Additional
Scope: Database

Inserts a print action. This option only appears in the database module. If a data view is active, the sub option of inserting a Print Current Record action will also be available.

A RAD report operates on a custom view and consists of one or more of the following components: an input screen, a query, a sort/key order, an SPL program, or a SmartWare report definition. When inserting a report, the dialog box requests a default name for these components and provides access to the Report Component dialog box.

See also:

Printing, page 48

Report Components for Data Views, page 125

Print_Crystal

Location: Insert Additional
Scope: All

Inserts a print action that executes a Crystal report - available in Windows only.

When inserting a new Crystal report, you can launch Crystal Reports with the button on the Insert Dialog. You should set the $$crw_path variable in the angoss.cfg file to point to the directory where Crystal Reports is installed. If it is not set, RAD looks in the C:\CRW directory and the D:\CRW directory.

Note: Crystal Reports must be installed in order to create or modify these reports. However, the runtime files installed with SmartWare will allow users to print Crystal reports without requiring the actual program.

Reports (.rpt files) should be saved in the home directory of the application. This is important when you first launch Crystal Reports, because it may not be in the correct directory.

Note that the order or index of the current view does not affect the printing of a Crystal report nor does Crystal Reports have access to custom views. This is because the connection between SmartWare and Crystal Reports is through an ODBC driver. One way of working around these limitations is to create a temporary database (using the DATA QUERY command for example) which reflects the current order and index and includes any calculations you want to use. Then make your Crystal report access the temporary table.

Keyboard_Macros

Location: Insert Additional
Scope: Database Navigation Menus

Inserts the Macros view on the current menu. This option only appears when a navigation menu in the database is active.

Job_Stream/Procedure

Location: Insert Additional
Scope: Database

Inserts a job stream or procedure on the current menu. This option only appears if the database is active. If you do not specify a job stream or procedure in the Insert Job Stream/Procedure dialog box, the resulting item will simply execute the Run a Procedure or Job Stream selection from the Management and Utilities Menu.

See also:

Procedures and Job Streaming, page 70

Return

Location: Insert Additional
Scope: All

Inserts a Return action on the current menu.

Data_Entry

Location: Insert Additional
Scope: Data View

Appears when a data view is currently active. It lists the standard data entry actions. With the exception of the Zap action, all of these are automatically included on a data view when originally inserted.

The list includes: Browse, Delete, Enter, Find, Krunch, Order, Update, and Zap.

For a detailed description of the action of each of these functions, refer to the Users Manual in the Documentation view (Management & Utilities menu option).

User_Utilities

Location: Insert Additional
Scope: Data View

Appears when a data view is currently active. These actions are automatically included on a data view when originally inserted.

The list includes: Query, Send_to_SS, Mail_Merge, Custom_Report, and Keyboard_Macros

For a detailed description of the action of each of these functions, refer to the Users Manual in the Documentation view (Management & Utilities menu option).

Totals

Location: Insert Additional
Scope: Data View

The Totals option only appears when a data view is currently active. This inserts a command or menu selection that sums the contents of selected fields over all currently available records.

See also:

Totals Definition, page 116

Update

Location: Insert Additional

Scope: Spreadsheet View

Available only in a spreadsheet view, the Update selection creates a command that allows the end-user to modify the displayed worksheet in a controlled fashion. Only blocks of cells which have been defined as “edit areas” will be accessible. These edit areas may be different for each user mode. Also, only numerical or text data may be entered while in Update mode; formulas cannot be entered.

To insert an Update action on a data view, refer to the Data Entry option.

See also:

Edit Areas Definition, page 120

Graph

Location: Insert Additional

Scope: Spreadsheet View

Available only in a spreadsheet view, the Graph option creates a command that generates a graph. Graphs can be set as any one of the SmartWare graph types and output can be “pre-wired” to a variety of devices (e.g., printer). The output decision can also be left to the end-user.

Specify the graph type, destination, and definition file name in the Insert Graph dialog box and then select the Graph Definition button to access the SmartWare graph definition screen. For details on graph types and defining graphs, refer to the Spreadsheet manual.

Report

Location: Insert Additional
Scope: Spreadsheet View

Prints a spreadsheet report. In the Insert Report dialog box, enter the report file name and then select the Report Definition button to access the spreadsheet report generator. For details on defining spreadsheet reports, refer to the Spreadsheet manual.

Edit_Area

Location: Insert Additional
Scope: Spreadsheet View

Allows the end-user to edit one or more specific areas of the spreadsheet. These areas are associated with the new command you are Inserting. They should not be confused with the edit areas defined for use with the Update action, and can, in fact, be completely dissimilar to them. Each command whose action is Edit_Area has a separate set of areas associated with it. In contrast to this, the Update command always uses the edit areas associated with the end-user's user mode. In summary, the set of edit areas associated with the Edit_Area action are command-specific, while the edit areas associated with the Update action are user-mode-specific. For details regarding the definition of these user-mode-specific edit areas, refer to the Edit Areas Definition section in Chapter 4.

One possible use for commands of this type is to allow the user to change parameters (e.g., interest rates) whose values affect calculations elsewhere on the spreadsheet.

See also:

Edit Areas Definition, page 120

Spreadsheet

Location: Insert Additional
Scope: Spreadsheet

Temporarily suspends execution of the application, and gives the user direct access to the spreadsheet module. Once the user has completed the direct-access tasks at hand, exit is used to return control to the RAD application.

Word_Processor

Location: Insert Additional
Scope: Word Processor

Suspends ANGOSS and gives the user direct access to the SmartWare word processor. Once the user has completed the direct-access tasks at hand, exit is used to return control to the RAD application.

Communications

Location: Insert Additional
Scope: Communications

Gives the user direct access to the communications module. Once the user has completed the direct-access tasks at hand, exit is used to return control to the RAD application.

Modifying an Item

The Modify command is used to alter menu items, menu characteristics, or the various data objects. Between the graphical and character modes of operation, Modify works a little differently.

Graphical

The options presented on the modify menu include the menu itself, the data object if present (e.g., data view), and all items from the current navigation

or view menu. Sub items on pull down menus, such as Enter (which appears under Edit), are indented below their parent items.

If a menu item can be modified beyond its menu information (known as Item Information), an arrow on the pull down menu points to a sub menu that allows modification of both the item information and the object or definition. For instance, if the item is an SPL program, the sub menu will contain: Item_Information and Program.

Character

The options presented on the modify menu include the menu itself, the data object if present (e.g., data view), and the current menu item. Note: to modify the current item, it must be highlighted prior to activating the Developer.

If the current menu item can be modified beyond its menu information (known as Item Information), an option to modify that object or definition will also be presented. For instance, if the current item is an SPL program, the option to modify the program will also appear.

A description of all possible modification types, and the circumstances under which they appear, follows.

Browse_Fields

Location: Modify Database
Scope: Data View

Appears whenever a data view is active. It modifies the fields that are displayed when in browse mode. Note that browse fields may be changed by usermode.

Default_Edit_Areas

Location: Modify Spreadsheet

Scope: Spreadsheet View

Appears while a spreadsheet is active. Default edit areas are blocks on the spreadsheet which the user can view or modify. Different edit areas may be defined for each user mode.

See also:

Edit Areas Definition, page 120

DV_Info_(sysdef)

Location: Modify (System Defaults)

Scope: Database

Appears when a menu item leads to the system defaults screen. It modifies the information pertaining to the system defaults view, in a manner identical to that of modifying the information of any other data view in the system.

See also:

Screen Information, page 110

Edit_Area

Location: Modify (Edit Area)

Scope: Spreadsheet View

Appears whenever the highlighted command has an action of Edit_Area. Be aware that you will be modifying a set of command-specific edit areas, not the user-mode-

specific areas, when you select this command. Note: the edit area behavior can be overridden.

See also:

Edit Areas Definition, page 120

FPR_Template

Location: Modify (DDL)
Scope: Data View

Appears when a data view is active and the command takes the end user to another data view. Since the command calls a data view from within a data view, a DDL is required. If the DDL type is file-per-record, this option will appear.

With a file-per-record DDL, a separate destination data file exists for each record in the source data file. If a data file does not already exist for a particular record, a new file is created. This is accomplished by copying an empty template data file. The FPR_Template command allows the user to access the custom view editor to modify this template view. For more details on FPR links refer to the DDL Definition section in Chapter 4. For instruction on using the custom view editor, refer to the SmartWare Database manual.

See also:

Screen Information, page 110

FPR_Keys

Location: Modify (DDL)
Scope: Data View

Appears when a data view is active and the command takes the end user to another data view. It modifies the keys used by the DDL. For details on FPR links refer to the DDL Definition section in Chapter 4.

See also:

Screen Information, page 110

Graph

Location: Modify
Scope: Spreadsheet View

Appears whenever the highlighted command causes a graph to display. It can change the graph type and invokes the graph definition screen. For details on graph types and defining graphs, refer to the SmartWare Spreadsheet manual.

Information

Location: Modify (Database, Spreadsheet, Document)
Scope: Data View, Spreadsheet View, Document View

Appears whenever there is a data view, spreadsheet view, or document view active. It leads to an information screen containing loading, color, location, file name and usage information regarding the view. For details on these three screens, refer to the Screen Information, Spreadsheet Information, and Document Information sections in Chapter 4.

See also:

Chapter 4: RAD Objects and Definition Files, page 107

Input_Screen

Location: Modify (Audited Program)
Scope: All

Appears when the highlighted choice invokes an audited program. Audited programs are associated with input screens. For details on audited programs, refer to the RAD SPL Programming section in Chapter 6.

See also:

Input Screen Files, page 245

SPL Programming, page 149

Add_Key

Location: Modify Database

Scope: Data View

Appears only when a data view is active. It allows key fields to be added to the current data view. For an explanation of key fields, refer to the SmartWare Database manual.

Delete_Key

Location: Modify Database

Scope: Data View

Appears only when a data view is active. It allows key fields to be deleted from the current data view. For an explanation of key fields, refer to the SmartWare Database manual.

Item_Information

Location: Modify

Scope: All (menu items)

Contains information required for the RAD App to generate actions when menu items are selected. Among others, this information includes the description, action, object, and function of an item.

In pulldown menus, when Modify is selected, the presence of an arrow beside an item name indicates that the Item_Information option is on a sub menu. If the arrow is not present, selecting the item itself produces the Item_Information's dialog box.

See also:

Item_Information, page 109

Library

Location: Modify
Scope: All

Appears only when a library is bound to a menu. To bind or remove a library from a menu, refer to the Menu_Characteristics section in chapter 4.

See also:

Menu_Characteristics, page 107

Link_Definition_(DB)

Location: Modify (DDL)
Scope: Data View

Appears when a data view is active and the highlighted choice leads to another data view. Link_Definition_(DB) modifies the DDL definition. For more details, refer to the DDL Definition section in Chapter 4.

See also:

Screen Information, page 110

Link_Definition_(SS)

Location: Modify (DSL)
Scope: Data View

Appears when a database view is active and the current choice leads to a spreadsheet. It modifies the path, file name, template and transfer information of the DSL. For details, refer to the DSL Definition section in Chapter 4.

See also:

Spreadsheet Information, page 117

Link_Definition_(WP)

Location: Modify (DWL)

Scope: Data View

Appears when a data view is active and the highlighted choice leads to a document view. It modifies the path, file name, template and data transfer information of the DWL. For details, refer to the DWL Definition section in Chapter 4.

See also:

DWL Definition, page 124

Menu

Location: Modify

Scope: All

Appears any time Modify is selected. Its two sub options provide access to the Characteristics and Editor options.

See also:

Menus, page 107

Characteristics

Location: Modify Menu

Scope: All

Modifies the general characteristics of the current menu. These include titles, colors, and the audit status of the menu, among other attributes.

See also:

Menu_Characteristics, page 107

Editor

Location: Modify Menu

Scope: All

Reads the current menu's MNU file into the text editor. Because it is possible to damage an MNU file, DO NOT EDIT it if you are unsure of the MNU file format.

OSA_Script

Location: Modify (OS CALL)

Scope: All

Available when the current choice is an OS call (i.e., runs a DOS batch file or Unix script). OSA_Script loads the script into the text editor.

Password

Location: Modify

Scope: All

Passwords can be added or removed from menus.

Program

Location: Modify (Program)

Scope: All

Appears when the current choice executes an SPL program. Modification takes place in the text editor followed the option of compiling the program.

See also:

Chapter 6: Advanced Application Development, page 149

Program_(Loading/Unloading)

Location: Modify (Program)

Scope: Navigation Menus

Appears when the current choice leads to a data view, spreadsheet view or document view that uses an override program.

See also:

Chapter 4: RAD Objects and Definition Files, page 107

Chapter 6: Advanced Application Development, page 149

Report

Location: Modify (Print)

Scope: Database/Spreadsheet Module

Appears when the highlighted command causes a database report or spreadsheet report to be printed.

Database:

This leads to the Report Component dialog box. For more information on RAD reports, refer to the Reports section in Chapter 4.

Spreadsheet:

This leads to the spreadsheet report generator. For instruction regarding this report generator, refer to the SmartWare Spreadsheet manual.

See also:

Report Components for Data Views, page 125

Chapter 4: RAD Objects and Definition Files, page 107

Crystal_Report

Location: Modify (item)
Scope: All

To change which report is executed or the report target, modify the Item_Information for the particular item. To modify the report, choose Crystal_Report.

Note: Crystal Reports must be installed in order to create or modify these reports. However, the runtime files installed with SmartWare will allow users to print Crystal reports without requiring the actual program.

Screen_and_DB's

Location: Modify Database
Scope: Data View

Appears whenever a data view is active. The SmartWare custom view editor is used to modify the view. For details on the custom view editor, refer to the SmartWare Database manual.

Before modification takes place, the Developer offers the option to unload background data files. Background files refer to standard or custom views which may also be active. Unloading background files may be necessary if fields are added to or deleted from a data file that is attached to a background view. It also requires less memory during modification. However, leaving background views loaded takes less time and may be handy if the current view references background views (e.g., for testing FILELOOKUPs with F5).

See also:

Data View, page 110

System_Defaults

Location: Modify (System Defaults)

Scope: Database

Appears when the highlighted choice leads to the System Defaults view. The SmartWare custom view editor is used to modify the view. For details on the custom view editor, refer to the SmartWare Database manual.

See also:

System Defaults, page 136

Text_File

Location: Modify (Text File)

Scope: All

Displayed when the current choice causes a text file to be displayed. It modifies the access mode of the text file and the text file name expression.

Totals

Location: Modify (Totals)

Scope: Data View

Appears when the highlighted choice runs a totals calculation. For details on creating and modifying a Totals Definition, refer to the Totals Definition section in Chapter 4.

See also:

Totals Definition, page 116

View_SYSDEF_Program

Location: Modify

Scope: Database Module

Appears when the current choice leads to the application system defaults. It displays the program which is responsible for setting the system default public variables from the system default view. For details on the System Defaults view, refer to the System Defaults section in Chapter 5.

See also:

System Defaults, page 136

Worksheet

Location: Modify

Scope: Spreadsheet View

Appears when a spreadsheet view is active. It temporarily suspends the RAD application so that modifications can be made to the worksheet directly with the spreadsheet module.

Deleting an Item

The Delete command removes an item from the current menu, and optionally deletes any files associated with the item.

When Delete has been selected, the Developer first verifies that you wish to delete the item. Following this, you will be asked a series of questions regarding the deletion of each file associated with the item. For example, if the item calls a menu, you will first be asked to delete the menu file and then prompted to delete each of the files associated with that menu. In other words, the deletion process is recursive, moving from the current choice downwards in the menu hierarchy until the “terminal nodes” are reached.

Moving an Item - Reposition

It may be desirable to alter the order in which items appear on menus or to move an item from one menu to another. The Reposition command is used to perform this task.

Note: In character mode, select the item before initiating the Developer. In graphics mode, select the item from the Reposition pulldown menu.

Menu Order

To move an item to a different position on the same menu, select the item (as stated above), choose the appropriate location in the Insertion Point screen, and select Insert.

Moving to another Menu

Moving an item to another menu is a two step process. To do this, select the item (as stated above), pick the Insert on Other Menu button from the Insertion Point screen, and exit the Developer. Then move to the appropriate menu, restart the Developer, and re-insert the item.

Note: Attempting to exit the application without re-inserting an item produces a warning. You should then return to the application and re-insert the item. You may choose to exit anyway, in which case the outstanding item will be lost, but any files associated with the item will not be deleted from the hard disk.

Utilities

Utilities provide a number of useful tools for application development. The following describes each of these tools.

Remember

Location: Utilities
Scope: All

This command gives you access to a number of utilities which deal with the creation, execution and editing of SPL programs. A couple of enhancements over the Smart-

Ware Remember commands, such as compiling all project files and editing application libraries have been added. For information on the SmartWare Remember commands and on the SmartWare Programming Language (SPL), refer to the SmartWare Project Processing manual and Chapter 6: Advanced Application Development, page 149, in this manual.

When the Remember selection is selected, a command list appears with the following options: Application_Library, Compile_All, Execute, Beautify, and Lint.

Application_Library

The Application_Library command allows you to edit or create one of two program files that contain library functions for the current module. All functions placed into the libraries may be accessed by other project files and calculations.

The two libraries are High use and Low use - frequently used functions should be kept in the high use library. Refer to Application Libraries, page 161.

Compile_All

Compiles all SPL programs in a specified directory. You have the option of compiling with or without debug information.

Execute

The Execute option allows an SPL program to be run. (This should not to be confused with the Developer's Insert Program command, which inserts a menu item that, in turn, causes a program to be run.)

Beautify

This program corrects indentation problems in SPL programs.

Lint

This program identifies unused variables and functions in SPL programs.

OS

Location: Utilities
Scope: All

The OS command provides temporary access to the operating system. To return to ANGOSS, simply type “exit” at the operating system prompt and press Enter.

Preferences

Location: Utilities
Scope: All

Provides access to a number of global preferences.

ANGOSS_Configuration

Opens the text editor on the ANGOSS.CFG configuration file.

Database, Spreadsheet, Wordprocessor, Communications

Opens the current SmartWare module specific preferences editor (e.g., Tools Preferences Database). For more information on these preferences, refer to the appropriate SmartWare manuals.

Global

Opens the SmartWare global preferences editor (i.e., Tools Preferences Global). For more information on these preferences, refer to the SmartWare System manual.

HardWare

Opens the SmartWare hardware preferences editor (i.e., Tools Preferences Hardware). For more information on these preferences, refer to the SmartWare System manual.

Menu_Colors

Colors can be modified for all navigation and view menus in your RAD application. The Menu_Colors command presents a dialog box where the five color types can be modified. Since this is a global operation, color values always default to a -1 value. For more information, refer to RAD Colors, page 173.

Background_Graphic

In graphic mode, the background graphic (i.e., the graphic behind navigation menus) can be changed with this command.

Changes_Log

Location: Utilities
Scope: All

Views the log that records system changes. This log contains the User ID, date, time, and comments for these changes.

Icon_Editor

Location: Utilities
Scope: All (Graphic mode)

Provides access to the Icon Editor. For more information, refer to Icons, page 174.

Database
Spreadsheet
Wordprocessor
Communications

Location: Utilities
Scope: All

This command causes a RAD application to be temporarily suspended, providing direct access to the current SmartWare module. Exit is used to return control to the application.

Though not recommended, you can also suspend execution of the RAD application at any time by using the Ctrl Z key stroke. This key stroke is also useful when testing your own SPL programs.

You will not be able to execute a SPL program while the RAD application is suspended. For this purpose, use the Developer's command sequence Utilities / Remember / Execute.

Quit

Location: Utilities
Scope: All

Exits the Developer System. You can also do this with the Esc key or the right mouse button.

Chapter 4: RAD Objects and Definition Files

RAD applications are made up of objects which were initially created with the Developer System's Insert command. Many of these objects are simple and do not require extended information - some have additional settings, definition files, and/or sub objects. This chapter describes those objects.

The following information is reference material. If you are unfamiliar with the Developer System or with RAD applications, we recommended that you at least *read* through the tutorial.

Menus

In general, there are two separate aspects of menus that can be modified: *the characteristics of the menu itself*, such as its color or type; and *the information that specifies what an item selection will do* including the object it acts upon.

Menu_Characteristics

This dialog box contains general information regarding the current menu when the Modify Menu_characteristics command sequence is selected. Altering its data will affect the way in which the menu appears or operates.

Short Name

Contains the system name of the menu. Its contents cannot be changed.

Module

Indicates the active SmartWare module for the menu. Its contents can only be altered if the menu is a navigation type. The NoCh option does not force a module switch regardless of the current module.

 # Items

Displays the total number of options on the current menu. Its contents are automatically adjusted when choices are added or deleted.

Type

Displays the menu type. It cannot be changed.

Help Text Pointer

Indicates the overview help text file name.

Title

Displayed title of navigation menus or command list prompt. With navigation menus, this text also appears within all automatically-generated documentation as a main heading for the current menu section. In multilingual systems, this text is overridden by corresponding translated text.

Color fields

This group defines the menu's text, highlighter, and border colors.

Library to Activate

Optionally, an SPL program can be activated when the menu is selected. It is unloaded when returning from the menu.

Tool Bar Placement

Indicates the location of the tool bar when in graphics mode.

Audit Usage

Indicates whether the menu is to be audited. If so, each time a user selects an item from an audited menu, the details of the occurrence are recorded in a special audit log. This log can be viewed from the User Management menu.

Item_Information

This dialog box contains information regarding the functionality of a menu item when modifying it. Altering this data may affect how the menu item appears or operates.

Short Name

Contains the system name of the menu on which the item appears. Its contents cannot be altered.

Module

Specifies the SmartWare module with which the current menu is associated. To change the module, refer to the Menu_Characteristics section.

# of Items

Displays the total number of options on the current menu. Its contents are automatically adjusted when choices are added or deleted from the menu.

Type

This field displays the menu type.

Description

Contains the description that appears on the menu. For users not assigned to the default language, a translated description will override its text.

Action

Contains the menu item's action code. A menu item's action dictates what happens when the item is selected. The F6 key produces a pop-up menu containing all valid action codes. For details regarding these actions, refer to Appendix D of this manual.

Object

Contains the item on which the action operates. For example, if the action is M (call a Menu), this field must contain the short name of the menu to call; an action of X (eXecute a program) invokes the program listed here. If the action does not require an object, this field will be blank.

Function

Supplements the information in the Object field. The use of this field varies, depending on the action. Refer to Appendix D for details.

Icon

Specifies the item's icon in graphics mode. The F6 key produces a file list selector. The Icon Editor can also be accessed.

Autohelp

Displayed autohelp text on the bottom line of the screen. The autohelp text can be no more than 80 characters in length. In multilingual systems, translations may override the contents of this field.

Help Text Pointer

Contains the filename of the help text associated with the item.

Data View

A Data View consists of a custom view, one or more data files, and a view menu. There are also a number of definition files that are or may be associated with a data view.

Screen Information

Data view Screen Information can be accessed when creating a data view or when the Developer's Modify Database Information command is selected while at a data view.

The Screen Information editor is actually a custom view record (each data view in a system has a corresponding record in that system's scr_info view). Since it is a custom view record, the editing commands used here are the same as those of a normal data view.

Note that, depending on the size of your screen, this view may be divided up into a number of pages.

The following section describes each of the Screen Information fields:

Screen Name

The current custom view's file name.

Path

The location, on disk, of the custom view and usually of attached data files.

The path field should contain a public variable - this makes it easier to relocate data files and allows for multiple location data file support. Note that this field requires an expression so that a variable must be entered or, in the case of a hard coded path, a quoted string.

Immediately below the Path field's data, is the result of the expression.

Short Name of menu...

This unique name identifies a particular data view to the system. In fact, it is synonymous with Data View Name. Note that the Screen Name could not serve this purpose because it is possible to have a number a data view's using a single custom view.

Data Files

This table lists all of the data files that are attached to the data view. The first column lists the data file name. The next column, Relation, indicates whether the data file is the DRIVER or is Driven. If the data file is Driven, then the next two columns show the two link fields, one from the driver data file and the other from the driven data file. To change information in this table, modify the data view itself - the Developer will then automatically update the table.

Update/Delete Expression

This field allows for the entry of a logical expression that will determine if users can update or delete records. Typically a field's data is used within the expression so that, if the expression evaluates to True, the current record cannot be updated or deleted.

Border

When the border is on, scroll bars are available. In graphics mode, the Tool Bar is activated as well.

Load/Unload Program

A project file can be used to override or modify the standard loading process. Do not enter anything in this field unless you are familiar with override programs. Refer to the Loading/Unloading Override Programs section in Chapter 6 for details.

Input Program

Note: for compatibility only - refer instead to the Event Handling Functions section in Chapter 6 for details. An Input Program can be used to override the standard view menu that is responsible for responding to end-user key strokes. Do not enter anything in this field unless you are familiar with custom view menus.

Browse Fields

Displays the list of fields used during browse mode. To modify the browse field list, select the Modify Database Browse fields command.

Auto Browse

Indicates whether the data view should initially be presented in browse mode. This may also be changed by user mode.

Key Fields - Style

Specifies the menu type, either bar or prompter, for key field selection in character mode. This affects the FIND and ORDER menus presented to the user.

Key Fields - Case Conversion

A case conversion modifier can be applied to any of the key fields. As a convenience to end users, these modifiers can automatically convert their input during the FIND and ORDER commands to match the field's format. For example, data in the [Last Name] field is stored in proper case, the user performs a find on this field and enters the name DOE, the application automatically converts this to Doe and finds the appropriate record.

The modifiers are single character entries:

U	upper case
L	lower case
P	proper case
0 - 9	use input mask <n> (e.g., $_mask1)

other no change

As well as the U, L, and P types, the digits 0-9 specify an input mask. The input mask is defined by a public variable $_mask<n>, where <n> = 0 to 9. This public variable can be defined by declaring and assigning it a value in an application library or by creating a [mask<n>] field in the System Defaults and entering a mask string into the field.

Note that, when a mask is used as a modifier, it makes sense that the field on the custom view, too, has a mask or that a rule be used to convert the data. For instance, a Rule can be developed to automatically convert a field to upper case, lower case, or proper case. To do this, modify the custom view and create an error checking rule with a formula similar to:

```
==(IF DBPUT("[Last Name]",PROPER( [Last Name]) )THEN [Last Name] ELSE -1)
```

A rule message, such as Case Conversion Failed, should also be added in case the DBPUT function fails to assign the new value to the field, which, in theory can only happen if the field is READ ONLY or the data does not conform to other rules on the field.

To summarize, in order to compensate for the fact that the keys are sensitive to upper and lower case, you can ensure that the data is always entered in a particular format using a mask or rule. Then, by using the Key Type fields, you can tell the FIND and ORDER commands to also conform to the same format.

The result is that the end user need not be concerned with upper and lower case.

Key Fields - Alias and Suppression

Specifies the name that will appear on a FIND or ORDER field list regardless of the actual key field name. If no alias is specified, the key field name is used.

A key can also be suppressed from the ORDER or FIND menus. To do this, enter one of the following into the alias field:

- removes from both the ORDER and FIND

-f removes from the FIND.

-o removes from the ORDER.

If an alias is required on a suppressed key, it may be entered after the -f or -o parameter (e.g., -fNew Name).

Default Key (1-15)

Sets the default order of the file when the data view is accessed. If left blank, the file will be ordered physically. This may be overridden by user mode. The number from 1 to 15 represents the key field number.

Referenced Data Files...

This table lists the data files (standard views) or custom views that are to be automatically loaded in the background. You may require files to be loaded so that the filelookup function can access them or because a program may access them. The following columns are located in this table:

File Name

Contains the file name of the referenced custom view or data file that is to be loaded in the background.

Type

Indicates that the referenced file is either a data file (standard view) or a custom view. The letters D and V represent these two options respectively.

Key Field

May contain the name of a key field for ordering the file.

Path

Contains the path name in which the data file or view resides. This field defaults to the correct path, but may be altered if required. As with all paths, remember to use a public variable name to encode the name where possible.

Default Screen Colors

Indicates the screen colors of the data view. When a field is blank, the color for that screen element will be the one that was set and stored in the custom view. If a color field is set to -1, the screen color is derived from the menu. Any other value (between 0 and 15) sets the color to that value. However, colors specified at the field level override those entered here.

DDL Definition

Inserting a data view on a view menu (a data view is active) presents an option to link these views. This is known as a Database to Database Link (DDL). A DDL definition

is accessed from either the Insert Database dialog box or the Modify Link_Definition_(DB) command.

There are three DDL types: Simple, Binary_Query, and File_Per_Record. Each is described below.

Simple DDL Definition

A Simple DDL causes the destination data view to be loaded. Optionally, an automatic search for a specific record in the destination data view can be invoked. This search will use information from specified fields in the source data view. The destination data view will be presented to the user with its records in search key order.

The definition screen offers two input items:

Destination Field - Enter a key field name on the destination data view that will be searched upon arrival. If it doesn't matter which record is displayed when the destination is reached, leave this blank.

Lookup Expression - Enter an expression (probably using field names from the source data view). The result of the expression is used to search the Destination Field.

Binary Query DDL Definition

A Binary Query DDL is similar to a Simple DDL except that a subset of records may be selected.

The definition screen offers three input items:

Destination Field - Enter a key field name on the destination data view that will be searched upon arrival. In a Binary Query DDL, this field may not be left blank. Use F6 for field selection.

Lookup Expression - Enter an expression (possibly using field names from the source data view). The result of the expression is used to search the Destination Field. Use F6 for field selection.

Additional Conditions - Enter a logical expression using fields from the destination view. In order for a record to be selected, this expression must evaluate to be True. If no additional conditions are required, leave this field blank.

File Per Record DDL Definition

A File per Record DDL is quite different from the other two DDL types. It allows a set of identical data files to be created; a separate directory is created for each record in the source data file. The path name of the directory in which the destination data file resides is determined from specified data in the source record. An empty, template will be copied into the directory the first time it's accessed.

The definition screen offers three input items:

Destination Field - Enter a key field name on the destination data view that will be searched upon arrival. If it doesn't matter which record is displayed when the destination is reached, leave this it blank. Use F6 for field selection.

Lookup Expression - Enter an expression (probably using field names from the source data view). The result of the expression is used to search the Destination Field. Use F6 for field selection.

Path Expression - Enter an expression, possibly using field names from the source view. The result of the expression must be a string of alphanumeric characters, specifying the path of the destination data file. This field cannot be left blank. This expression, like any other in the ANGOSS system, follows the SPL expression conventions. Hence, if you wish to hard code a path name, such as C:\MYSYS\FILES, you must enclose it quotation marks. On the other hand, if you refer to the contents of a public variable, it should not be enclosed in quotation marks.

Totals Definition

A Totals Definition is used to sum the contents of selected fields over all currently available records in a data view. The Query or Order command can be used to select a subset of records to sum. When the user selects the command whose action is TOTALS, the calculation will be performed, and the results displayed on the screen.

To create a Totals option on a data view, enter the calculation definition file name into the Insert Totals dialog box and select the Totals Definition button. The Calculate Total Definition dialog box appears. Modifying an existing Totals option takes you directly to the Calculate Total Definition dialog box.

Totals Definition Dialog Box

For each total calculated, enter a description of the calculation in the Titles column. The description will preface the result of the calculation.

The Format column indicates how to display the results of the calculation. For example, to display a result with 2 decimals, right justified, and preceded by a dollar sign, enter the code 2r$. For details on format codes, refer to the FORMAT function in the SmartWare Formula Reference manual.

In the Expression column, type an expression utilizing one or more fields from the database (press F6 to list fields). The expression is evaluated using data from each record - these calculations are then summed in the final result.

Spreadsheet View

A Spreadsheet View consists of a worksheet and a view menu. There are also a number of definition files that are or may be associated with a spreadsheet view.

Spreadsheet Information

Spreadsheet View Information can be accessed when creating a spreadsheet view or when the Developer's Modify Spreadsheet Information command is selected while at a spreadsheet view.

The following section describes each of the Spreadsheet Information fields:

Menu

The system name of the spreadsheet view's menu. This cannot be edited.

Spreadsheet Name

Contains the worksheet name. If a DSL is active this field will be calculated, that is, its value is determined when a user selects the spreadsheet view from a data view menu.

Path

The path of the directory in which the worksheet resides. Generally the path is coded, via public variables, to facilitate relocation, or to allow a different spreadsheet to be loaded for each user (for example, a user's personal directory, $$persdir, is specified). If a DSL is active, this field will be calculated and its contents will not be changeable.

Associated Spreadsheets

The associated spreadsheet fields allow you to supply menu names of other spreadsheet views. These will each have their own Spreadsheet Information Screens. Found on each of these screens is the path and filename of a worksheet that is to be activated when the main worksheet is loaded.

Loading Program

Optionally, an SPL program can be used to override the standard spreadsheet view loading process. Leave this blank if you are not familiar with SPL within RAD systems. Refer to the Loading/Unloading Override Programs section in Chapter 6 for details.

Edit Program

Note: for compatibility only - refer instead to the Event Handling Functions section in Chapter 6 for details. Optionally, an SPL program can be used to override the standard spreadsheet view controller. Your custom edit program will be responsible for responding to user's key strokes.

Edit Areas

Contains the default Edit Areas. These may be overridden for specific usermodes.

Edit areas are bounded sections of the spreadsheet. The cursor may not be moved outside the predefined area except by switching to another edit area. The Developer's Modify Spreadsheet Default_Edit_Areas command can be used to physically select blocks of spreadsheet cells to be defined as edit areas. Note: the predefined event handling function, normal_ss_event, overrides edit area behavior.

Colors

Allows for color settings of various worksheet objects.

DSL Definition

Inserting a spreadsheet view on a data view menu (rather than a navigation menu) presents an option to link these two views. This is known as a Database to Spreadsheet Link (DSL). A DSL definition is accessed from either the Insert Spreadsheet dialog box or the Modify Link_Definition_(SS) command.

The definition dialog box offers the following fields:

Short Name - Contains the name of the DSL definition.

# of Links - This calculated field simply displays the number of data items that are to be transferred to and from the spreadsheet (i.e., the number of items in the Links table).

SS Name Expression - The spreadsheet name expression must contain an expression which evaluates to a string containing the spreadsheet's name. This expression may use field names, functions, or literal text. An example using all three might be:

```
left([customer name], 6)|"SS"
```

Path Expression - Enter an expression which evaluates to a string specifying the path name of the destination spreadsheet.

Template Expression - The template is a spreadsheet, which is copied into the appropriate directory the first time the destination spreadsheet is accessed. Hence it can be used to set certain default conditions regarding the spreadsheet involved in the DSL. This field should contain an expression, possibly utilizing fields from the database, which will evaluate to a string specifying the path and file name of the template. IF or CASE statements may be used if required. Refer to the SmartWare Formula Reference manual for details.

Links - This table lists all of the data items (fields or cells) that will be transferred to or from the spreadsheet. The three columns of the table are explained below:

Action	This column specifies in which direction the transfer of data is to occur. DSL indicates Database to Spreadsheet, and SDL indicates the reverse. A DSL transfer occurs when the spreadsheet view is invoked. An SDL returns data to the database upon exit of the spreadsheet view.
DB Field Name	This column must contain field names on the data view. If information is being sent to the spreadsheet (DSL), a value is read from the database field and written to the corresponding spreadsheet cell. This happens when the DSL command is invoked by the end user. If, however, an SDL action is specified, a value is read from the spreadsheet cell and written to the corresponding database field when the end user returns from the spreadsheet. Square brackets are not required when specifying field names. Note: the field names specified in a SDL (spreadsheet to database links) must NOT be read only (the RAD App will not be able to affect those fields). Use Project-Write instead.
SS Cell	This column must contain a cell reference. This may be in the format of r1c1 or a named cell. This cell will be written to with the value contained in the database field when the action is DSL (database to spreadsheet link). This occurs when the user invokes the command which leads to the spreadsheet. If an SDL action is specified, the cell reference is read and written to the corresponding data file field when the user exits the spreadsheet.

Edit Areas Definition

Edit areas are bounded sections of the spreadsheet. The cursor may not be moved outside the predefined area except by switching to another edit area.

There are two basic types of edit areas:

The first type applies to the spreadsheet view as a whole; these edit areas are utilized whenever the end-user selects the Update command (or any command with an action of UPDATESS) from the spreadsheet view com-

mand list and governs cursor movement when at the command list. Default edit areas may be modified by selecting the Developer System command, Modify Spreadsheet Default_Edit_Areas from a spreadsheet view. These defaults may be overridden, by assigning a different set of edit areas to users belonging to each user mode, with the User Mode Maintenance system (refer to the User Mode Maintenance section in Chapter 5).

The second type of edit area is associated with a particular spreadsheet view command. Such commands have an action of Edit_Area. When the end-user selects such a command, he or she is allowed to edit the spreadsheet, just as with the Update command. But instead of using the default or user-mode-specific edit areas, a special set of edit areas, associated with the command itself, are used. This is useful if you want to create a command that allows the user to modify a certain cell(s). For example, if a spreadsheet is an amortization schedule, you may create a command to change the interest rate. This type of command may be added to a spreadsheet view by selecting the Insert Additional_Function Edit-Area command. The edit areas associated developer's command is defined by selecting Modify Edit Area.

Note: the pre defined event handling function, normal_ss_event, overrides edit area behavior.

Each of the above Developer commands present the same dialog box - be careful that you know which type of edit areas you are defining:

Area Reference

Enter a block or cell reference into this column. This reference will define the extent of this particular edit area. Pressing the F4 key will cause the current spreadsheet to be displayed, so that the highlighter may be used to mark a block on the spreadsheet. The F6 key allows you to define the edit area as corresponding to a previously-named cell or block. It is recommended that names be used instead of row-column references, since such names will be automatically adjusted when the spreadsheet is modified. Blocks may be named by selecting the Modify Spreadsheet Command. This will give you direct access to the spreadsheet module, allowing you to use the Sheet Name Define command to set up block names.

Outside Area Lock

These two columns define the position on the screen at which the edit area is to be displayed. If these are left blank, the edit area will be positioned at the top left corner

of the screen. The # Rows column specifies the number of spreadsheet to be displayed above the edit area. The # Columns column specifies the number of spreadsheet columns to be displayed to the left of the edit area on-screen. These borders might contain descriptions or column headings, or may be left blank merely for aesthetic purposes.

Inside Area Lock

These two columns specify a number of rows or columns inside the defined area to lock. Locked rows or columns will not scroll when the user moves the cursor past the edge of the screen. This is useful for ensuring that row or column titles are always visible. An outside area lock will also perform this function, but there are two instances when an inside area lock must be used. First, if a command which causes a new row (or column) to be inserted into an edit area is included on the command list, at least one row (or column) must be inside-area-locked to ensure that the edit area is increased in size, rather than being moved, to make room for the new row/column. Second, the spreadsheet supports criteria blocks, which specify criteria regarding which cells are to be included in various operations. Both the edit area block used in the operation and its corresponding criteria block must include the same set of row or column titles. Thus it makes sense to include these titles in the edit area, and then use an inside area lock to prevent the user from altering them.

In general, it is wise to use the two types of area locks in the following manner: use outside area locks to position the top left corner of the edit area on the screen. Include row and/or column titles within the edit area, and then use the inside area locks to prevent the user from altering these titles.

Document View

A Document View consists of a document and a view menu. There are also a number of definition files that are or may be associated with a document view.

Document Information

Document View Information can be accessed when creating a document view or when the Developer's Modify Wordprocessor Information command is selected while at a document view.

The following section describes each of the Document Information fields:

Menu

The system name of the document view's menu. This cannot be edited.

Document Name

This field contains the file name of the document that is to be loaded upon invocation of the view. If a DWL is active (see below), this may be defined as a calculated field, in which case it cannot be altered.

Path

Enter into this field the path of the directory in which the document is located. Generally the path is coded (using public variables). This feature can be used to cause a different document (same name but different directory) to be loaded for each user who accesses the view. If a DWL is active this field becomes a calculated field, and you cannot change its contents.

Loading Program

Optionally, an SPL program can be used to override the standard document view loading process. Leave this blank if you are not familiar with SPL within RAD systems. Refer to the Loading/Unloading Override Programs section in Chapter 6 for details.

Edit Program

Note: for compatibility only - refer instead to the Event Handling Functions section in Chapter 6 for details. Optionally, an SPL program can be used to override the standard document view controller. Your custom edit program will be responsible for responding to user's key strokes.

Colors

These fields indicate the colors that should be used to display the document. If left blank, default colors will be used.

DWL Definition

Inserting a document view on a data view menu (rather than a navigation menu) presents an option to link these two views. This is known as a Database to Wordprocessor Link (DWL). A DWL definition is accessed from either the Insert Wordprocessor dialog box or the Modify Link_Definition_(WP) command.

A DWL allows the transfer of data from the database to the document. You may indicate that the document name and path is to be calculated based on information in the database, or specify this data explicitly.

The definition dialog box offers the following fields:

Short Name

Contains the name of the DWL definition.

# of Links

This calculated field simply displays the number of data items that are to be transferred to the document (i.e., the number of items in the Merge Variables list).

Doc Name Expression

This field must contain an expression. The string resulting from evaluation of this expression will specify the file name of the document to be loaded.

Path Expression

Enter an expression which evaluates to a string specifying the path name of the document.

Template Expression

The template is a document, which is copied into the appropriate directory the first time the document view is accessed. Hence it can be used to set certain default conditions regarding the document involved in the DWL. This field should contain an expression, possibly utilizing fields from the database record, which will evaluate to a string specifying the path and file name of the template. IF or CASE statements may be used if required. Refer to the SmartWare Formula Reference manual for details.

Merge Variables

This table lists all of the data items that will be transferred to the document. You may leave this table blank if no data needs to be transferred. The columns of this table are described below:

Variable Name

The variable name column can contain public variable names that will be treated as formula type merge variables in the destination document. The values of these variables will be set in the database, and then accessed in the word processor. To avoid conflict with other public variables, the first two characters of all such variable names have been preset to m$.

To insert a merge variable into a document, you must enter a start merge character (looks like << and is obtained by Ctrl J); an equals sign, to indicate a formula merge variable; the merge variable name; and finally, the merge finish character (looks like >> and is obtained by Ctrl K). For example, <<=m$name>>.

NOTE: Due to an idiosyncrasy in the word processor, you must include at least one replacement variable. A replacement variable is a variable that is merged from a text file or screen input rather than calculated. The Developer has provided an easy work around. Include a merge variable <<+dummy>> anywhere in your document (The plus sign tells the word processor not to leave a blank line if the merge variable is empty).

DB Expression

This column contains expressions, the results of which will be stored in the corresponding Merge Variables. These expressions may include database field references, formulas or literal data.

Report Components for Data Views

Overview

Unlike a SmartWare database report, which contains only a report layout definition, a RAD data view report may consist of a number of other component types and may not necessarily even contain a report layout. The possible components are: report layout, query, input screen, order type, and program. As well, a group of Execution

Parameters options provides the developer with the ability to preset certain run time alternatives such as forcing output to the printer.

While setting up or editing this type of report, a run time explanation displayed at the bottom of the Report Setup dialog box summarizes the interaction of the various components and parameters.

Setting up a Report

Reports can be located on either a navigation menu or a data view. To insert a report, select the Insert, Additional, Print, and Report commands (this last command, Report, is not issued from a navigation menu). After completing the Insert Report Dialog box entries, select the Report Components button.

The Report Setup dialog box contains the Component Options on the left, the Execution Parameters on the right, and the Run Time Explanation below those two.

The Components

To create, detach, or delete individual component definitions, select the appropriate Modify or Attach button from the Component Options section on the dialog box. Note that the View Name and Execution Parameters options can be changed but not deleted.

At run time, the order in which components are executed is as follows: load the view if required, set the parameters, load the input screen, run the query, sort the file, run the program, and finally, print the report.

Note that different RAD reports can share components. For example, you may have one RAD report that shares its query or sort with another.

View Name

Determines the view that the report will operate on. This name is actually a menu short name, not a custom view name. When first inserting a report, this option is automatically set to the current view if applicable (i.e., a data view is active). Since a report cannot be run without a data view, a valid view must be entered.

Report Layout

Report Layouts allow you to design the visual appearance of a report by defining columns, titles, etc. These layouts are ultimately created or edited in the database report generator. Refer to the SmartWare Database manual for details on this editor.

You may choose not to use the standard report layout but instead use LPRINTs or report definitions from within an attached program. Note that the report itself, if defined, is the last of the components to be executed at run time. This offers the potential to reference variables and functions that were defined in the program or input screen. Note also that the developer can bypass an attached report by setting the #abort variable from within the program.

Input Screen

Input Screens are generally used to input values into variables which can then be used in queries, programs or reports. It can however, be used just to describe a process before the end user executes it. Ultimately, this option executes the Input Screen editor in order to create or modify the Input Screen. Refer to the SmartWare Project Processing manual for details on this editor.

If an Input Screen is defined, the end user will be presented with it before any other components occur. This means that variables defined in the Input Screen can be used in the program, query, or report layout.

NOTE: that input screens used in reports must contain an input variable, of the table type, setting the value of the $sure variable to Yes or No. Normally, the system will automatically generate this.

Query File

Executes the Query editor in order to create or modify the Query. Refer to the SmartWare Database manual for details.

Variables used in the query may be set by the input screen. A report Query will be executed after the Input Screen but before the Sort, Program, or Report components. A query will also nullify an order-to-key operation, performed by the report's Order component or the end user.

Program File

While this SPL program can be used for almost any purpose, there are three main reasons to use one. First, you may want to replace the report layout with an LPRINT program or have a branch to various other report layouts. Second, you may require

additional processing to support the report. Third, you may need to set variables or store functions that are used in the report layout.

When creating a new program, the system will automatically generate a skeletal program that contains special variables and a message function. The three variables: $$clm, ##mhnd, and #audit; should not be used and definitely not modified. The variable $$fct is used to indicate the purpose that the program is being called for - there are two times that the program is executed: during the execution of the report and to activate variables and functions during the development stage.

The developer can also use the $$fct variable in the same way it is used in normal RAD programs, that is, the program can branch based on the value of $$fct set by a menu choice. This allows the program to be executed for other related purposes.

The rest of the program's variables, except #abort, can be read but should not be modified; most of them are typically used to control an LPRINT report. They indicate the report defaults and/or user selections. The following describes these:

#abort	The only value returned from this program back to the report interpreter. It is a boolean variable that, when set to one, will abort the remainder of the report. In practical terms, this means that the report layout will not be executed.
#copies	This numeric variable indicates the number of copies to be printed.
$destination	This string variable indicates the destination of the report. Its value will be the first letter of either Printer, Screen, Text-Screen, or Disk.
$dest_file	This is the path and the name of the file that the report will print to if the destination is set to Disk.
#end_page	The last printed page of the report is indicated by this numeric variable. If its value is 0, up to and including the last page will be printed. See also the #start_page variable.

#no_interrupt	This boolean variable can be used to determine whether the program is allowed to stop program flow. For example, when a Job Stream or Documentation Generation is running, the program should not stop and wait for user input. Under these circumstances, this variable will be set to true.
$one_record	This string variable can be set to either Y (yes) or N (no).
$records_used	This string variable indicates the initial records used to operate on. Its value will be the first letter of either Whole-File or User-Index.
$report_type	This indicates whether the report will print each record (of the current index) or just the totals. Its value will be the first letter of either Detail or Totals-Only.
#start_page	The first printed page of the report is indicated by this numeric variable. If its value is 0, the report will begin with the first page. Its compliment is the #end_page variable.

The rec_message function is used to send a message to the screen and/or audit report. To display a message and wait for a key press, the first parameter (display) must be set to 1. Note that the value of #no_interrupt will override this capability, and similarly, if the Audit execution parameter is set to No, an audit report message will not be produced. The second parameter is the message itself.

If these variables and functions are used correctly, you will have no problems running the report, whether it is executed from a menu, by a job stream, or by the document generator. However, when using LPRINTs, you should be aware of the number of pages printed when a document generation is performed. In this case, the document generator sets the start and end pages both to 1 so that a sample of any given report is a maximum of one page within the document. If the page variables are ignored, an extremely long sample report may be generated.

NOTE: New program code should be placed immediately after the line: INSERT PROCESS HERE in the proc() function. If the purpose of the program is to set variables and/or functions used in the report, declare these as public. It is not necessary to lock the variables since the program will remain loaded, leaving the variables active, until the report is completed.

Order To

There are two order types to choose from: key and sort definition. Both options use the SmartWare selectors. Refer to the SmartWare Database manual for details on key and sort order.

A report Order will be executed after the Input Screen and Query but before the Program or Report components. Note that an order to key option will be ignored if a report query is attached.

Execution Parameters

These parameters allow the developer to preset certain conditions for the report during run time. Notice that some of the parameters offer an Ask-User option. This means that, when the report is executed, a dialog box is built that presents all parameters set to Ask-User. For instance, if Destination is set to Ask-User, the end user will be able to choose whether the report output will be sent to the printer, screen, or disk.

To access the Execution Parameters dialog box, select the appropriate Modify button in the Components section. Alternatively, mouse users can cycle through individual options by clicking on that option in the Execution Parameters section (for example, click on the word Printer).

Destination

Report output can be sent to the Disk, Printer, Screen, or Text-Screen. If the option is set to Disk, the user will be required to enter a path and file name.

Report Type

This parameter causes the report to print all records or just the totals. A run time error is produced if Totals-Only is selected but no break points with totals or grand totals have been defined on the report layout. This option is ignored whenever Current Record Only is on.

# of Pages

This is the number of pages that will be printed. When set to Ask-User, the end user will be required to enter both the start and end pages.

# of Copies

This is the number of copies that will be printed.

Allow One

If this option is set to Yes, only the current record will be printed. If set to Ask-User, the end user will be given the option to print the current record only. This option is only offered when the report is attached to the current view.

Note that a Yes setting will cause the system to ignore the Report Type and Records Used parameter settings.

Records Used

The records initially operated on by the report components can be all the records in the file or the current subset. Unless the report operates on the current view, Whole-File will be automatically selected. On the other hand, if it is attached to the current view, you may decide to retain the user's current order (User-Index) or ignore the user's current order (Whole-File).

This option may be ignored depending on the Allow One parameter setting. As well, if this parameter is set to User-Index, an order-to-key, specified by the Order component will be ignored.

Use Audit

When this parameter is set to Yes, a record will be entered in the Process Log each time the report is executed. Note, that regardless of this parameter's value, execution through job streaming will be audited.

Chapter 5: Management and Utilities

This chapter covers the administration aspects of a RAD application. Unlike the Developer, this part of the application is not independent of the application. Most items discussed can be accessed from the Management & Utilities menu structure.

Note that the User Mode tools control access to this menu structure. Since passwords can be assigned to menus, developers can deny administrators access to specific items by moving those items to a new menu.

The following information is reference material. If you are unfamiliar with the Developer System or with RAD applications, we recommended that you at least *read* through the tutorial.

The Management & Utilities menu structure contains:

Documentation

Maintains documentation list and generates documents from help text.

Menu Hierarchy Diagram

Generates, displays or prints an application map.

Run a Procedure or Job Stream

Selects from a list of defined menu steps and executes. The Developer can also place these items on navigation menus.

Macro Definitions

Creates, loads, and maintains a list of macro sets.

User Data Base

Maintains the user list.

User Mode Maintenance

Maintains user groups. Adjusts user mode settings. Equivalent to Ctrl U.

Activity Log

Displays log of user activity on selected menus (determined by a Menu Characteristics setting).

Quick Key Access

Sets Developer, User Mode Maintenance, and Passwords Language Maintenance passwords.

File Structure Maintenance

Maintains database files and views. Provides access to the Info_set screen.

Language Maintenance

Maintains alternative language item descriptions for all menus. Equivalent to the Ctrl L key combination.

Procedures & Job Streaming

Creates and maintains a list of defined menu steps.

Batch Update/ Process Log

Maintains a list of log entries. Displays error reports and input screens from audited programs and reports.

System Defaults

Contains changeable values that affect the behavior of the RAD App.

Archive Job

Creates and maintains a list of archive Definitions.

Run an Archive Job

Executes a preset archive defined in the Archive Job Definition.

View Archive Log and Data

Views log on previously run archive jobs and provides tools for maintaining archived data.

A number of the major topics are discussed in the following sections.

Users

User Modes

Access to RAD applications menus can be controlled in groups to which users are assigned. By default, RAD applications come with three user modes: Developer, Administrator, and User. Others can be added.

The User Mode Maintenance screen is accessed with the Ctrl U key combination or from the User Management menu. In order to prevent unauthorized access, the User Mode password is requested. Note that, once the password has been entered during a session, the application assumes the user has the required status and does not request the password again.

By default, the User Mode password is **1621**.

Both the Ctrl U access key and the password can be changed. The access key is defined in the ANGOSS.CFG file. RAD passwords can be modified from the administrator submenu called Quick Key Access Passwords.

The User Mode Maintenance screen is arranged in columns where menu items appear in the left most column and user group (not including DEV) access status appears in the following columns. Only one menu is handled at a time.

The User Mode menu items provide the following actions:

File Next_Menu	Moves to next menu
File Quit	Quit User Mode Maintenance
Access Disable	Disables group access to menu item. Disabled status is represented by eight slash marks (////////).
Access Enable	Enables group access to menu item.
Access Password	Assigns a password to an item. The password appears in the user group's column. To remove a password, enter a blank string.

Usermode Add	Creates a new user group.
Usermode Remove	Removes a user group.
Usermode File_Fix	Clears user mode information from all menus.
Usermode Switch	Switches to specified user group for testing. Generally this can only be performed when activating User Mode Maintenance from the Main Menu.
Defaults All Users	DEV mode only. Provides alternative access to Menu_Characteristics and Item_Information.
Defaults Global	Sets global menu defaults (color, type, etc.) for each user group.
Defaults Current_Menu	Sets current menu defaults (help text access, macro loading) for each user group.
Defaults Edit_Areas	Sets spreadsheet Edit Areas for each user group.

Users

Each user on the system is assigned an ID, a password, a personal directory, and is placed in one of the user mode groups described above.

The Users data view is accessed from the User Management menu.

System Defaults

System defaults are data items which are used by the system. Fiscal periods, tax rates, the company address, cheque number counters, and default account codes are examples of typical system default items. Four system defaults are provided when a RAD application is first created. These are the system title, the current year, the current period, and the number of periods per year. Custom default items may also be added.

When a RAD application is initiated, a series of locked public variables are set according to field values in the first (and only) record in sysdef.db. The variables can then be referenced in calculated fields, spreadsheet formulas or programs. After modifying the contents of the System Defaults view, the RAD App requests whether these variables should be recalculated each time the application is started.

The System Defaults option is on the System Management menu (found on the Management and Utilities menu). The related Developer Modify commands are described here:

System_Defaults

Modifies the sysdef view in SmartWare's custom view editor. Every field on this view is interpreted by the RAD App as a system default. When modification has completed, the Developer generates an SPL program which is run each time the application is started. The program sets public variables, whose names are derived from the field names in the view, to the values in the system defaults view. Variable names are prefixed with $_ (dollar sign underbar) and spaces in field titles are replaced with an underbar (e.g., $_tax_rate). Name your fields with this in mind.

DV_Info_(Sysdef)

Modifies the View Information record pertaining to the system defaults view. Through this record, other views or data files can be activated whenever the System Defaults view is loaded. This provides capability to create calculated fields and error checks for sysdef.

View_SYSDEF_Program

Since the system defaults program is automatically generated, there is no provision to edit it directly. However, this command will allow you to review the derived public variable names in the program.

Hierarchy Diagram

Displays an application map.

The cursor keys and mouse are used to navigate through the diagram. Menu items provide the following responses:

File Generate	Generates the diagram.
File Print	Prints the diagram.
Data Find	Searches diagram for supplied text.

Date Mode	Toggles between Description and Action/Object/ function listing.
Data Trace	Displays direct path from the top menu to the current item.
Data View	Displays the current menu as it appears in the application.
Jump_to	Jumps directly to the actual item.
User_Modes	Accesses user mode maintenance.
Width Compact	Adjusts displayed item width to 10 characters.
Width Full	Displays full item width.
Width User-Defined	Adjusts displayed item width to the number of characters supplied.

Archiving

Archiving facilitates the transfer of information from one set of data files to another. It is used to remove information from operation files. It is also designed so that developers can easily transfer data around their application.

Archive definitions are created in the Archive Definitions view. This view can be accessed by selecting the following menu items:

Management and Utilities > System Management > Archive Menu

An Archive definition identifies a source view and a destination view. The data structures of the two views must be identical, with a few minor exceptions discussed below. Note that the views may have many data files attached (relational).

An Archive definition also specifies if and how the data from both driver and driven files are to be transferred. A series of three questions - for the end user to answer at run time - can be defined to control the process. A query definition selects the records to be transferred.

Archiving can append data to the same data file or files each time it is run. This is used to move data from one area of the application to another. For example, "*input*" records can be moved into master files, or, records can be moved into a static history file.

Archiving can also create a data file or a set of data files by copying a blank version of the specified destination view. This is typically used when you want to remove data from the system so it can be off loaded from the main storage device to a removable media such as tape.

Creating the Destination View

Before defining the archive definition, you must create the destination view. Use the Developer to insert a new database view. Here are some tips on creating the archive destination view:

- Create the destination view SIMILAR to the source view.
- Use the REPLICATE DATAFILE command to create the destination datafile(s). This will copy the field structure.
- Change any counter fields to numeric fields.
- Not all data files on the view may need to be archived. If you do not replicate files (i.e., the source and destination both reference the same datafiles) then the data file will not be processed. When the destination view is accessed, the original data file is used. This is useful for lookup files that will not change over time.
- The archive routine will load the source and destination views simultaneously, so no file name or pathing conflicts may exist.

Here are some special considerations if the archive job is not appending. (i.e., A new set of data files is created each time the archive is run).

- The destination view (.vw) will be copied, without changes, to a subdirectory beneath $$archdir (normally $$homedir|"arch").
- Paths for data files that are not processed must be calculated in the Edit Links section of the destination view definition. (e.g., =$$syspath|"product").
- Paths for data files that are processed must be blank so that the location of the data file matches the view.
- A special loading program will be automatically created and attached to the destination view. When users access the view, they will be able to choose

which archive job to access. Developers will have the option of accessing the template.

Creating the Archive Definition

Each record in the archdef.vw stores an archive definition. To create a new archive definition select the Enter command. Use Update to modify existing archive definitions.

The Enter and Update commands both lead to a dialog box where the definition can be created or modified.

Name	Archive definition ID.
Description	A 40 character description.
Source View	Contains the view name from which data will be transferred. Use F6 or click the arrow for a list.
Destination View	Contains the view name which will receive the data.
Transfer All Records	If this field is “No”, then a query is performed to select which records to transfer. If it is “Yes”, all records will be transferred, thus saving the time taken to run a query. If you are not appending records (see below), the data can be moved with an operating system copy, which also greatly enhances performance.
Append to Existing	If this field is “No”, then new datafile(s) are created in an archive directory each time the query is run. “Yes” indicates that the records will be transferred to the same datafiles, each time the archive is run.
Verify	If this field is set to “Yes”, then the destination and source records are compared to double check data was successfully transferred.
Datafiles	Click this button to reach the table which specifies details about how the data is transferred.

User Interface	This button leads to a view where you can enter end user questions, help text, etc.
Query	This button allows you to define the query definition.

Datafiles Button

The Datafiles dialogue box allows you to define the details of the data transfer.

The source datafiles, listed in the left column are matched against the destination data files. In most cases, the datafiles will automatically be correlated correctly. If required, you can rearrange the list by changing the destination datafile.

If the datafile names match, it is assumed that the destination view will reference the original files.

The Operation column describes the events that will occur when the archive process is run. To change this, press the related Edit button.

The Edit buttons lead to another dialog box. Each of these fields are described below. For example, imagine that you are archiving orders. The orders view has three data files, ordhead (header information), orddet (order item detail), and customer. Imagine that we want to transfer all three data files to the archive destination.

Operation

If this field is set to “Copy”, the original records remain. If it is set to “Move”, then the original records are deleted and purged. Using our orders example and assuming we are removing old orders, we would probably set ordhead and orddet to “Move”, and set the customer datafile to “Copy”. This would copy and delete the order information and only copy the customer information.

The following fields only appear when you are not transferring all data and the data file in question is a driven datafile.

Merge Options

This field only appears appending records. It allows you to specify how driven records from the source are merged into the existing destination records.

“Refresh” indicates that the destination data file is to be searched before

adding a record. If a record is found with the same key value, it will be updated (refreshed) with the new values read from the source. If a record is not found, it is added.

"Keep-Old" tells the archiving routine to search the destination file for a matching key. (Just like Refresh.) If a match is found, the destination data-file is not affected. If a match is not found then the record is added.

"Unconditional-Add" indicates that the destination file is not searched. Records are always added.

Putting this in context of our orders example, we would set "customer" to either "Keep-Old" or "Refresh". We would want to avoid duplicates in the destination version of "customer". The orddet datafile would be set to "Unconditional-Add" as we do not have to concern ourselves with duplicates.

Transfer one child

This option allows us to optimize the performance of the record only archiving routine. If this field is set to "Yes", then after one driven link is found (in the source), another match is not sought. For example, ordet should be set to "No" and customer should be set to "Yes". If we were to set ordet to "Yes" then only the first item on the order would be transferred. If we were to set customer to "No" then an unnecessary find operation occurs on each order, as it tries to find a second related customer record.

Report Missing Children

If this field is set to "Yes", then missing driven records, in the source file, are reported as warnings in the error report. In our example, orddet should be set to "Yes", because it is inconceivable that an order would not have any line items. If the users of our imaginary example leave the customer blank (say for cash sales), then we would expect orders with no customer number. In this case we would not want to report missing driven records (customer).

User Interface Button

These options control how the user interacts with the archive process and where and how the data is off-loaded.

Help Text

This is the text shown to the user during the archive process.

Defaults

Up to three default values (each can have a description) are accessible to the query definition via the variables $$default1, $$default2 and $$default3 (in that order on the list). This allows you to create defaults such as “Number of Months to Keep Data”.

Questions

These questions are asked of the user just prior to the commencement of the archive procedure. The user’s responses to these questions are accessible to the query definition via the variables $$answer1, $$answer2 and $$answer3 respectively. You must supply text in the form of a question or prompt and specify what type of data would constitute a valid response. A typical question is “Enter the latest date to archive”.

Off-Load Information

Specifies the commands used by the operating system to copy files to and from the archive directory. For example, in DOS, you might use the commands BACKUP and RESTORE. To specify the archive directory in the command syntax, use the string “{archdir}” in lieu of an actual directory specifier. This will ensure portability and flexibility in your archive definitions. For example the offload command may be “backup {archdir} a:/s” and the restore command may be “restore a: {archdir}/s”.

Automatic Off-Load indicates whether or not you wish to have the files in the archive directory off-loaded immediately following an archive operation.

Query Button

This opens the query editor where you determine the records that will be archived. Remember, you can use the $$default1, $$default2, $$default3, $$answer1, $$answer2 and $$answer3 variables that are described in the User Interface section above.

Running an Archive Job

Open the Run an Archive Job menu item, highlight the archive process you want to run, and choose Select from the menu. The Archive dialog will then allow you to enter the appropriate answers defined in the Archive Definition and give you a final option to abort the archive process.

When complete, the archive process presents statistics including the name, time started, variable values (answers and defaults), number of records selected, etc.

Viewing the Log and Archived Data

Open the View Archive Log and Archived Data menu item and move to the archive record you want - use Browse mode if you like. To view the archived data, select File > Load. To view the archive statistics, select Data > View Report.

Help Files and Documentation

Help Files

At any menu item, help can be accessed through the F1 key or question mark (?) icon.

This produces a list (similar to the hierarchy diagram) of the current menu's items and a control panel to its left. Users with the appropriate authoring privileges can modify the help text for any menu item. Note that Application help and Technical help have separate access rights set in the Users file.

Help Text Formatting - Dot Commands

When help text is read into the word processor by a RAD App, it is automatically formatted. The format will occur when you use the word processor to edit help text or when you generate a document.

Paragraphs must be separated by blank lines or dot commands. If they are not, the format program will assume they are the same paragraph. This allows you to ignore the margins while using the text editor or make text lines long or short, whichever is convenient. Formatting will automatically tidy up the text.

For appropriate spacing of the generated document, it is a good idea to leave the first line of the help text file blank.

The formatting process also automatically converts ASCII line graphic characters to word processor line graphic characters when generating a document. This allows the ASCII graphics to be normally displayed in the help text and still print properly from the generated manual. If you want to include a screen dump in your help text, capture the screen image and then read the text file images into your help text. Line graphic characters will appear properly in both the generated documentation and help text.

RAD Apps recognize dot commands. These commands allow you to change the formatting or fonts, and to insert keep marks (for page break control). When help text files are displayed on the screen, dot commands are omitted. When editing help text in the word processor, they appear as hidden text. When a document is generated, dot commands are omitted.

A Dot command must be the only item on a line and cannot have any preceding or trailing spaces. However, dot commands may be entered in either upper or lower case.

The following is a list of recognized dot commands:

..regform-<font>	Means REGular FORMat. It will format text to suit 80 column help screens and 8.5 x 11 printouts. This is the default mode, so if no dot command is used, the REGFORM format is adopted.
..regformi-<font>	Means REGular FORmat Indent. It is identical to REGFORM except that it causes the first line of each paragraph to be indented.

..pointform-<font>	Formats the text so that the first line of each paragraph overhangs the body of the paragraph. It is ideal for listing a series of points or describing the use of database fields.
..noform-<font>	Turns off any formatting. Text in this form remains as it appears in the text editor. This is required for screen dumps or other diagrams included in your help text.
...sameform-<font>	Changes the font without affecting the format.
...keepst	Use this to mark the start of a block of text that cannot be broken by a page break. It must be followed by a ..keepen.
..keepen	Used to mark the end of a block of text that cannot be broken by a page break. It must be preceded by a ..keepst.
..startkeep	Inserts a keep block starting at the dot command and continuing down for six lines. No keep end is required.

The <font> can be substituted with “F” and the font number or COMPRESS or NORMAL. You may also leave the font indicator off (e.g. ..regform).

Compress

The COMPRESS option will affect the graphic character conversion so that it converts graphic fonts to F7. The COMPRESS option should be used when the text you want to format contains ASCII graphics.

Documentation

The automatic generation of documentation is performed from the Documentation item on the Management & Utilities menu. This is a data view that contains previously generated documents.

This view's Generate command creates the document based on menu item descriptions, help text, and a number of other selectable items. Documentation can begin at

any menu level where only items below that menu are included. Furthermore, the Management & Utilities menus can be excluded.

Once a document is generated, edit the fonts to best suit your printer. By editing the template (Template command), you can change the default fonts, which will affect documents generated in the future. The fonts used are:

0	Normal text (help text)
1	Level 1 titles
2	Level 2 titles
3	Level 3 titles
4	Level 4 titles
5	Level 5 titles
6	Subtitles
7	Textual screen capture (COMPRESS) - upper character set
8	Sample reports
9	File and View layouts
10	Condensed File and View layouts

The default fonts use Standard 12pt and filled area (serif) fonts that work on all printers.

Note that, when the Documentation Generator is running, the application is actually executed and screen shots are taken. When complete, the Generator returns you to the Main Menu rather than the Documentation view.

Chapter 6: Advanced Application Development

This chapter deals largely with programming issues in the RAD environment. It assumes an understanding of general programming concepts.

SPL Programming

The Smart Programming Language (SPL) is a sophisticated, Pascal-like language. SPL is a feature of SmartWare; in fact, RAD itself is written in SPL.

SPL provides a structured, procedural means of controlling almost every detail of the SmartWare software package. Any action which can be performed directly from SmartWare can also be included in an SPL program. Standard programming language features such as function definitions, looping, branching, and input/output may be combined in SPL with any of the commands found in each of the SmartWare modules.

This powerful combination allows you to incorporate all of the high level database, spreadsheet, word processor and communication features into your programs. This is a significant departure from usual programming concepts; but once mastered, SPL becomes an effective systems development tool.

Functions may also be written in SPL that can be used in calculated cells or fields in the spreadsheet and database. This feature adds yet another level of flexibility to the system.

In general terms, the following list of features is delivered by the Programming Language:

- A structured programming language
- Remember mode for beginners
- Global, local and public variables and arrays
- Pointers

- Function libraries
- DOS/Unix compatible code
- Token based compile for fast interpretation
- Easy-to-use editor, help text and compiler
- Debugging tools (trace, quiet mode, single-step)
- Access to hundreds of formulas
- Additions to native SmartWare menus
- KEYS command passing keystrokes directly to SmartWare
- Sophisticated error handling
- Text file access, binary data and low level interaction
- Cross platform graphics commands
- SQL access
- DDE access (Windows)

For complete details on the how to use the programming language refer to the Project Processing manual. For complete details on formulas (expressions) and available functions, refer to the Formula Reference manual.

Note: while SPL is not an Object Oriented language, these programming techniques can be used through pointers and callbacks (see the Sample application).

RAD SPL Programming

Because so much of a RAD application is generated automatically by the Developer System, the role that programs play is quite unique. Programs are usually only used to create data processing routines, special commands, complicated reports, and customized user interfaces.

A RAD App automatically handles interaction with the end user, taking them through the menu structure as it responds to their key strokes. A large number of actions, such as adding and deleting information, queries, graphing, and reports, are also handled without the requirement of a program.

When non-standard functionality is required, a program may be executed. This can be easily done by creating a new command with a program executing action or modifying an existing command and assigning it a program executing action.

The standard end user environment can be modified. The source code for functions that draw menus and command lists, and the code that responds to user keystrokes is provided in the DEV directory. This will allow you to develop applications with your own look-and-feel.

Almost all of the functions that a RAD App is comprised of are available for programmers to use. Refer to Appendix A for a list of these functions. Public variables, and actions are also listed in the appendices.

If you already have developed applications in SmartWare, refer to Appendix E on how to convert a SmartWare application into a RAD App.

Programmer Overview

This section attempts to deliver a general impression of how a RAD application works.

The Developer and the programs required to run RAD applications are written with SPL. The Developer provides a frame work to house the views, programs etc., which make up the end application. A series of menu files define the hierarchical structure of the application. SPL programs delivered with SmartWare (installed in \ANGOSS\APSYS\SHELL) display the menus in various formats and respond to end user input.

Most activities done with the Developer menu will automatically create or modify menus or edit referenced objects such as reports, views, programs, etc.

Creating a New Application

The Make-Application program creates a directory structure containing a base set of data files, menus and programs. These files are copied from the \ANGOSS\APSYS\DEV\EMPTY subdirectory. Scripts to run the application are automatically generated. Under Windows, items are added to the desktop. (This is done by using the DDE SPL commands to talk to the program manager.)

The copied files primarily deliver the 'Management and Utilities' section of an application.

Briefly, the features already built into a RAD application are:

- User database, for setting up individuals.
- User mode maintenance for establishing who can do what.
- Activity log for auditing end user activity.
- File structure maintenance for presenting datafile relationships.
- Information Sets for organizing data access.
- Language Maintenance for translating applications.
- Archiving for removing old information from the system.
- Procedures and Job Streams for defining batch processing jobs or corporate procedures.
- Batch Update and Process Log for tracking the running of data processing routines.
- System defaults for defining and changing global defaults or preferences.
- Automatic end user documentation generation, which combines the menus, help text, screen shots, etc. into a SmartWare WP document.
- Menu Hierarchy Diagram, which delivers a road map spreadsheet of the application.

The actual compiled SPL programs relating to these features are installed in the \ANGOSS\APSYS\SHELL directory and are not copied. The menus and empty data files pertaining to the above features are copied.

Start Up

To begin loading a RAD application, the project file start.rf3 is executed. If the program is present in the home directory of the application, that version is executed. Otherwise, the version provided in the \ANGOSS\APSYS\SHELL directory is used. The source, start.pf3, is installed in the \ANGOSS\APSYS\DEV directory.

The startup program performs the following tasks:

- Sets various public variables.
- Loads project processing libraries.
- Loads the dv_info datafile.
- Executes the login program.
- Initiates the first menu of the application.

Running

Each menu of a RAD application is stored in a separate file in the SYSTEM\MENUS subdirectory. Internally, these menus are accessed and maintained by the mnu_ series of functions. To examine the contents of a menu file, use the developer command MODIFY MENU EDITOR.

A stack of menus is maintained while running an application. It can be examined by looking at the public array $$sn[], (menu Short Name) - ##lev indicates the current level.

To load a new menu the function push() is used. When push is run the read_header() function is in turn called which opens the menu, reads it into memory, and then closes the file. The menu is also processed. The processing includes deleting unused choices (controlled by usermode maintenance), adjusting colors, and adding extensions, such as key fields. Also, if running the application in character mode, the menu is flattened and sorted to obtain a command list.

The push function also:

- Ensures that the appropriate SmartWare module is active
- Calls module specific loading functions which load related, views, documents or spreadsheets.
- Activates SPL libraries bound to the menu.

Once the push has completed, the menu is displayed with the menu project files (menu1.pf3 to menu6.pf3) or with the get_com() function which assumes an active view, document or spreadsheet. Note that source code for displaying menus is provided so that developers may modify how a menu looks.

Once a user selects a choice from a menu it is acted upon by the do_choice() function. (do_choice() is reached by the menu("dochoice") call suggested in the documentation for previous versions).

The do_choice() function operates on three key public variables: $$act (action), $$obj (object), and $$fct (function or parameters). These three variables contain all the information and/or references required to execute a menu selection. The action indicates what to do and the object indicates who it is done to. For example, if the action is x (meaning eXecute) the program specified by the object is run. If the action is m (meaning Menu), the object contains the name of the menu file which is to be loaded. The $$fct variable is a general purpose information holder which can affect the execution. For example, the program executed by the x action, can use $$fct to decide what it has been called to do.

Summary

The intent of the Developer is to make the job of developing applications faster and easier.

It provides an excellent environment for building applications by:

- automatically delivering features that are useful in real life situations.
- providing an environment where programmers can easily integrate their own programs into the application.
- being flexible enough to accommodate all situations.
- reducing the amount of code that needs to be written.

Accessing SmartWare

Sometimes it may be necessary to move between a RAD application and SmartWare. This can be achieved by invoking the Developer System with Ctrl A and then selecting the Utilities Database (or other module) commands or by pressing the Ctrl Z key combination.

When using the Ctrl Z combination, a prompt will appear asking if you want to Cancel or Suspend execution. Pressing the escape key will ignore the Ctrl Z and program control will continue as though the Cancel/Suspend was not invoked.

Providing that you have remained in the same module, pressing F8 will return control to the RAD App place that you were prior to suspending. You should be aware of the consequences of certain actions while suspended before pressing the F8 key. For example, if a file is unloaded, it should be reloaded in the appropriate window.

Select Cancel if you want to access the SmartWare Command Line options including the Remember Start and Finish commands. Unlike suspend however, you must re-enter the application by executing one of two programs: MENU (located in the home directory) or RESTART (located in the shell directory). If the cancel occurred in the Database module, you can re-enter by executing the MENU SPL program which will return you to the Main Menu. Otherwise, you can re-enter by executing the RESTART SPL program which will return you to the menu you were on prior to the cancel.

If a Ctrl Z is issued during a utilities restructure, query, or sequential find, then the option to cancel or resume the operation will occur. Selecting Cancel will terminate the process and Resume will continue it. Neither option will pass to control to the SmartWare Command Line.

NOTE: In all cases, if you have moved to a different module while at the SmartWare Command Line, you will need to return to the database module and execute the START SPL program which is usually in the home directory of the system.

End User Access to SmartWare

To provide SmartWare access to the end user, insert the SUSPEND_APP function into your program. Refer to Appendix A for more information.

Custom Commands

To create a customized command, select the Developer's Insert Program Exit command sequence. This will create an access point to an non-audited program.

While not required, it may be a good idea to enter a value for the Function (sometimes called Parameter) in the Insert Execute dialog box. If this function/parameter uniquely identifies the menu item, then the program can be called from multiple places in the application. This works because the function/parameter value is placed in the $$fct variable which can then be inspected in the program with an IF or CASE statement to determine which task to invoke.

To modify the functionality of a standard command such as Enter or Update, an SPL program has to be written to handle it and the action of the menu item must be changed so that the program is run when the user selects the item. To do this in graphical mode, start the Developer, select Modify, select the item and, if required, select Item_Information. In character mode, first highlight the item, start the Developer, and select Item_Information. This presents the Menu Information dialog box:

In the Object text box, enter the name of the SPL program which is to handle the customized functionality.

In the Action text box, enter or select an **x** action (the execute action).

In the Function/Parameter text box, enter a unique identifier.

Sometimes it is desirable to modify the Return command so that a special task is carried out when the user exits the current menu or view. Do this in the same way as described above but be certain that the last line executed by the program reads POP(). The POP() function returns the user to the previous menu - in essence, it adds the RT (return) action to the end of your program.

The following table illustrates the Menu Information (description, action, object and function/parameter) of each item on a sample customized view menu. This view menu leaves the Browse, Find and Print options unaltered while the Delete, Enter, Update and Return commands have been customized as described above.

Sample Customized View Menu - Menu Information

Description	Action	Object	Function/Parameter
Browse	browse	-	-
Delete	x	custom	delete
Enter	x	custom	enter
Find	find	-	-
Print	m	oep	-
Update	x	custom	update
Return	x	custom	return

The following *custom.pf3* program handles the four customized commands.

```
'Program Name: custom.pf3

EXTERNAL $$fct
EXTERNAL pop()

MAIN
    IF $$fct == "delete"
            <commands>
    ELSEIF $$fct == "enter"
            <commands>
    ELSEIF $$fct == "update"
            <commands>
    ELSEIF $$fct == "return"
            <commands>
            pop()
    END IF
END MAIN
```

Processing Routines

Audited Programs

RAD applications have special support for processing routines and batch updates. These types of SPL programs are called audited programs. Any SPL program created in this fashion will automatically have the following characteristics:

- An input screen with the same name
- Job streaming capabilities
- Automatic recording in the Process Log
- Automatic filing of error/audit report and input screen

The input screen is provided to allow users to supply information to the program. These are usually parameters that specify how the task is to be carried out.

Each time a user invokes an audited program, the date, time and User ID is recorded in the Process Log. Any errors or other types of messages produced by the program are stored in a message file for later viewing. Data entered into the start up input screen is also retained, so that it may be reviewed at a later time.

To create an audited SPL program, invoke the Developer and select the command sequence Insert Program Audited. Fill in the Insert Audited dialog box text fields and create both the program and input screen.

Input Screen

The input-screen should contain a brief description of the process and should ask the end user any information that is required for the program to run correctly. This information can be gathered using the input screen's Table or Input-Item options. Store answers in variables.

For more information on Input Screens, refer to the SmartWare Project Processing manual.

Program

The Developer automatically generates all the lines of code required to handle the Auditing and Job Streaming features. Your code should be inserted into this generated program at the point indicated by the comment:

```
' ADD PROCESS HERE
```

Since this comment is in the middle of the MAIN function, it's a good idea to simply declare and define your own main process function and call it from there:

```
' ADD PROCESS HERE
my_function()
```

To access variables on the input-screen, they must be declared as PUBLIC in the program. To give these variables default values or to set them to null strings ("") before the input screen is loaded, add the assignments after the following line:

```
if $$task=="input" or $$task=="both"
```

For example, the variables $input1 and $input2 are assigned null values before the input screen is loaded:

```
if $$task=="input" or $$task=="both"
        $input1 = ""
        $input2 = ""
        #proc_num=start_aud($inputscr)
```

Notice the function named rec_message(1) at the end of the program. Call this function whenever a message should be entered into the error/audit report.

The following example contains a function that validates the information on an invoice before it is processed.

```
FUNCTION process_invoice()
    IF check_inv()
            ...process invoice
            rec_message( "PROCESSED: Invoice #" & STR(#inv) )
    ELSE
            rec_message( "- Error occurred on: Invoice #" & STR(#inv) )
    END IF
END FUNCTION

FUNCTION check_inv()
    DATA GOTO VIEW "customer.vws"
    DATA FIND [Customer #] EQUAL STR([inv.cust]) OPTIONS "g"
    IF [Customer #] <> [inv.cust]
            rec_message( "ERROR: Invalid customer #" & STR([inv.cust] )
            RETURN 0
    END IF
    ...
    RETURN 1
END FUNCTION
```

Messages generated by audited programs can be viewed or printed after the routine has finished. You may access the error report at a later date via the Process and Batch Update Log on the System Management Menu.

Changing Modules

If you want to jump outside of the database module during an audited project you must take certain steps to ensure proper execution. These steps also ensure that a job stream will continue to run.

In the initial audited project, you must add the following external reference:

```
EXTERNAL module_call()
```

Then, when you want to jump to the other module, insert the following line:

```
module_call(1,"ssbatch")
```

The above example calls the spreadsheet and executes the project file “ssbatch.rf1”. For more details on module_call() refer to Appendix A. The following sample project file is executed in the destination module.

```
EXTERNAL ##mhnd module_call()
FWRITE ##mhnd FROM "In destination module" ’ Write message to log
module_call(3,"batchret") ’ Return to DB
```

The module_call() function is executed when execution resumes in the database. The actions performed are the normal actions performed when terminating an audited project:

```
EXTERNAL $$advr $$clm $$dirsep
EXTERNAL ##cpstep          ’ Current procedure/job stream step #
EXTERNAL ##mhnd            ’ Message file handle
EXTERNAL ##process_curnum  ’ Currently running batch/process #
EXTERNAL complete_log(2) errorrep(1) get_path()
PUBLIC rec_message(1)

MAIN
    LOCAL #proc_num
    #proc_num = ##process_curnum
    rec_message("Now, I’ve made it back to the DB")
    rec_message("END OF REPORT")
    FCLOSE ##mhnd
    complete_log(#proc_num,"<any comments>")
    IF ##cpstep = 0 ’ Job Stream is NOT running
            errorrep(get_path("proclog")|"proclog"|\
            $$dirsep|str(#proc_num)|".err")
    END IF
    $$clm=str(#proc_num)
    EXECUTE $$advr|"restart" IN-MEMORY
END MAIN

FUNCTION rec_message($mess)
    FWRITE ##mhnd FROM $mess
END FUNCTION
```

Calling an Application from a Program

It is possible to execute an application menu selection from within a program, just as if a user had actually chosen that item from the menu system. To do this, you must

specify the action, object and function/parameter values, in the same way that it is specified in the menu system (these values can be found on the Menu Information dialog box).

To specify these values, set the public variables $$act, $$obj and $$fct to text strings containing the required information and invoke the RAD interpreter function, menu(). For example:

```
...
$$act = "sv"   'Action = spreadsheet view
$$obj = "disctab"  'Name of SS View = DISCTAB
$$fct = ""   'No Function needed for an SS View
menu( "dochoice" ) 'The "dochoice" parameter is needed
...
```

Notice the syntax of the function call to menu(). The parameter, dochoice, is required to instruct RAD to execute the command specified by $$act, $$obj and $$fct. Other parameter settings will cause the menu() function to have quite a different effect - refer to Appendix A for details.

Libraries

Application Libraries

The Developer allows you to create libraries of application functions which can then be accessed from any of your SPL programs. These functions can also be accessed from within calculated fields or cells providing that they do not have any module specific commands. Place all of your library functions for each SmartWare module in a separate SPL program named applib.rf? or applibhu.rf?, where the ? is the number for the resident module. Declare all functions as PUBLIC in these libraries.

While the reason for two application libraries is originally historical, they can be used to help organize functions.

Each time a module is initiated, the RAD application will check to see if the appropriate APPLIB program exists, loads it, and makes its public functions available. Whenever a library is loaded, the MAIN section is executed. You may leave this section blank by writing MAIN followed immediately by END MAIN. However, you

may choose to place a function here which will be called each time that module is initiated.

Use the Developer's Utilities Remember Application_Library command series to modify an application library.

Notes:

The copyright message can be changed by setting the $$dev_name variable in the application library (You must declare the variable as public and issue a LOCK-SYSTEM statement on it).

Any System Default variables used in application libraries must be declared as PUBLIC. This is due to the order of initialization process.

Menu Libraries

Programs can be bound to menus. A program of this type, known as a menu library, is loaded when the menu is accessed and remains loaded until the user backs up from the menu.

Libraries are bound to menus through the Menu_Characteristics dialog box.

Programs On Non Modular Specific Menus

Some menus do not have a specific module assigned to them. These menus show NO CHANGE as the module. Because any module can be active when the menu is displayed, special considerations exist when executing programs.

The x action executes the SPL program in the current module. For example, when the spreadsheet module is active, an .rf1 will be executed but, when the data base module is active, an .rf3 will be executed.

Module specific versions of the x action exist so that a RAD App will automatically switch modules when required. The following shows the module specific execution actions:

x1 - Spreadsheet

x2 - Word Processor

x3 - Database

x4 - Communications

Loading/Unloading Programs

Occasionally, there is a need to enhance the view loading procedures - some applications require a series of spreadsheets or documents, others require input from the end-user to determine the path or file name of the view.

For this to work, two areas are affected:

Program_(Loading/Unloading)

To create or modify the program, move to the navigation menu that calls the view. In character mode, highlight the item that leads to the view, invoke the Developer, and issue the command sequence Modify Program_(Loading/Unloading). In graphics mode, select the Developer's Modify command, highlight the item that leads to the view, and select Program_(Loading/Unloading).

Information Screen

The above step automatically places the loading program's name into the view's Information screen. However, to remove a loading program, go to the spreadsheet, document or data view whose loading process you wish to remove and modify the view's Information screen. One of the fields on this screen specifies the loading/unloading program. Delete the SPL program name of the custom loader from this field. Note: this does not delete the program from disk.

The loading/unloading program is executed when the view is invoked. Note that the standard loading sequence may also be executed after the custom loading program is complete. During the loading sequence, the RAD App will set $$fct to "loading" before executing the custom loading program. The program can then set the $$fct variable to:

"**abort**" - The standard load process is not performed.

"**standard**" - The usual loading process is performed when the custom program is done.

The standard loading procedure will not malfunction if a view loaded by the custom program has already been loaded. Thus the custom loading program may be used to override the default path settings, located on the dv_info record and in the .ssi and .wpi files.

With spreadsheet and document views, the custom loading program can completely override the default files which would normally be loaded. However, when a data view is invoked, the view whose name appears on the dv_info record MUST be loaded but the normal path may be altered.

Loading and Unloading Program - Database Example

The following example simply displays a message before performing the usual load procedure.

```
EXTERNAL $$fct

MAIN
    IF $$fct == "loading"
            MESSAGE "Display this message before loading"
            $$fct = "standard"
    ELSEIF $$fct == "unload"
            $$fct = "standard"
    END IF
END MAIN
```

Event Handling Functions

Advanced developers can completely control the application screen by attaching an event handling function to a menu.

An event handling function is called each time a user presses a key, uses the mouse or selects a menu item. The event can be ignored, causing the RAD App to behave normally or, the event can be handled, allowing the program to produce the desired response.

To attach an event handling function, invoke the developer system while running the application at the appropriate menu. Select the Modify Menu Characteristics command. Within the Menu Information dialogue box, enter a function name in the “Event handling function” field.

Next, create the specified function and ensure that it is active when the menu is loaded. The function can be placed in one of the application libraries to make it available at any time. The function can also be put in a separate library which can be bound to the menu.

Parameters

An Event handling function must have the following parameters:

#key	The event id as returned by INEVENT.
#mnu	The menu id of the selected choice when #key is {MenuSelect}.
#choice	The choice number of the selected choice when #key is {MenuSelect}.
#mode	Will be true. This indicates that the menu is a RAD application menu.
$data1, $data2	Contains general data about the menu. Application menus do not use these parameters except in the spreadsheet where $data1 is equal to the edit area string.

Return Value

An event handling function must return 0, 1 or 2.

0	The RAD App will deal with the event in the usual way. This is generally used for any event that is not handled in the function.
1	The RAD App will not take further action in response to the event.
2	This special return value tells the RAD App that the current menu is terminating. It is appropriate to use this return if you have called the push() or pop() functions.

Events

Events passed in the #key parameter are keystrokes, mouse actions, and menu activities as returned by the INEVENT function. All events can be represented with the {} constants.

Keystrokes take the form of {F10}, {Space}, {left} {A}, etc. Examples of mouse events are {LeftUp}, {MouseMoved}, etc.

Two menu related events are {MenuInit} and {MenuSelect}. {MenuInit} occurs when a menu is initialized and should trigger the initial screen paint. {MenuSelect} occurs when the user has selected a command. In most situations, you will not need to intercept {MenuSelect} events. When you do, the #mnu and #choice parameters can be used in the mnu_info() function to obtain information about the selected menu choice.

Example

The following example displays a hello message when F8 is pressed and displays "Bye Bye" when the user exits the menu.

```
EXTERNAL display_message() pop()
PUBLIC eg_event()
GLOBAL byebye()

FUNCTION eg_event( #key, #mnu, #choice, #mode, $data1, $data2 )
    IF #key = {F8}
             display_message( "Hello", "", 4 )
             return 1 'AN EVENT WE HAVE HANDLED
    ELSEIF #key={esc} OR #key={f10}
             byebye()
             return 2 'MENU TERMINATING
    ELSEIF #key = {MenuSelect}
             IF mnu_info( #mnu, #choice, 1, 1, MNU_ACT ) == "rt"
             'MENU ITEM WAS RT (RETURN)
                      byebye()
                      return 2 ' MENU TERMINATING
             END IF
    END IF
    RETURN 0 'AN EVENT WE DON'T CARE ABOUT
END FUNCTION

FUNCTION byebye()
    display_message( "Bye Bye", "", 4 )
    pop() 'RETURN TO PREVIOUS APPLICATION MENU
END FUNCTION
```

For a more advanced example refer to the sample application shipped with ANGOSS.

The Sales selection uses the GRAPHICS commands to paint the screen. The normal screen display and navigation commands are completely replaced.

Navigation Menu Programs

ANGOSS ships the navigation menu source code so that you can customize any of the menu styles (Plain Box, Look Ahead, Cascading, etc.) or develop your own.

To affect all your RAD Apps, install the new menu program(s) in the ANGOSS Shell directory. Alternatively, installing the programs in the home directory of the RAD App will affect only that application.

The source code is installed in the ANGOSS DEV directory. The file names are:

menu1.pf3 - Plain Box Style menus

menu2.pf3 - Look Ahead menus

menu3.pf3 - Cascading menus

menu6.pf3 - Graphical menus

Note that a custom program menu5.pf3 can be created. This will allow administrators to easily switch users back to the original menu style - the active menu style can be modified by usermode (or in the angoss.cfg file). Menu mode 5 is intended for developer customizations.

Running Other Software and Switching Modules

The ability to store the user's current position within the menu structure makes it possible to exit the RAD App altogether, run another application, and then return to the exact position within the RAD App.

SmartWare's BIGOS and TOOLS OS commands can be used to run a DOS or Unix command while keeping RAD Apps active.

Other Software Packages

If you need to run a third party application package from within your application, there are two fundamental approaches. The fastest and easiest to use is the OS window.

Or, you can use the RAD App's capability to exit completely, run the third party application and then reload the application.

OS Window

An OS window can be invoked through OS_Call or by the BIGOS command in SPL. Unfortunately, depending on your memory mode, you can not always guarantee that there will be enough memory. Under Unix, the OS window is the preferred choice.

In Windows, BIGOS opens a DOS box - to create a Windows process, use the SPL function, PROCESS_CREATE.

In DOS, the BIG OS command will free as much low memory as possible by writing it out to disk before shelling out. Under tight memory configuration, this command may be problematic.

Temporary Exit - DOS Only

As an alternative to the DOS window, you can temporarily leave the RAD App, run the desired application and then return. This approach is slow, but will guarantee that you have enough memory and called programs can use expanded and extended memory.

The temp_exit() function allows you to easily exit and re-enter the application. This function actually calls another function that saves the end-users position within the application, and then sets a reentry flag. Note that the ANGIIUSR environment variable must specify the user's ID.

The temp_exit() function executes a script file named osapl.bat, which must be located in the user's work directory. This batch file is responsible for starting the external application, as it would be started from the DOS prompt.

One method of invoking a batch file is to copy the file to osapl.bat prior to calling temp_exit(). The example presented below uses this method to invoke a CAD/CAM software package:

```
TOOLS FILE COPY "cadcam.bat" TO $$workdir|"osapl.bat"
temp_exit()
```

If the script needs to vary according to the current record or other variables, you can use the FWRITE statement to actually create each line of the script file before calling exit function. The following example creates a batch file containing the single line, *dispdiag -X*, where *X* is a parameter which specifies a hypothetical subassembly identification code:

```
FOPEN $$workdir|"osapl.bat" AS ##wrkhnd OPTIONS RW_MODE
FSEEK ##wrkhnd 0
FWRITE ##wrkhnd FROM "dispdiag -"|[Subassembly ID]
FCLOSE ##wrkhnd
temp_exit()
```

The exit function sets a flag, and then returns control to the operating system. Here, the start-up batch file, which is used to start the application, continues to execute where it left off when the RAD App took control. It is this batch file which handles the details of running the osapl.bat file, as well as, returning control to the RAD App when the external application terminates. RAD Apps automatically detect the presence of the files which store the location information and uses them to return the user to the location from which the external application was called.

Changing Modules

If it is necessary that a different SmartWare module be invoked from within a project file, use the module_call() function. The following example assumes that you are moving from the database to the spreadsheet.

```
EXTERNAL module_call()
module_call( 1, "readgraf" )
```

The first parameter in module_call is the module number (1=spreadsheet, 2=word processor, 3=data base, 4=communications). The second parameter is the project file to execute in the destination module.

The SEND command may also be used. For example:

```
EXTERNAL module_call()
EXTERNAL $$advr
module_call( -1, "readgraf" )
DATA SEND SPREADSHEET ROW-FORMAT "[f001;f002]" \
PROJECT-FILE $$advr|"ppmodch.rf3"
```

Note that the module number -1 tells the module_call function to suppress the module change. Also note that the SPL program ppmodch.rf3 is executed, which will in turn execute readgraf.

Function libraries will be automatically activated in the destination module. You may call an application menu if required. For example:

```
menu_call( "summary" )
```

The above command will load the “summary” menu, where the end user can carry on with application activity. This sequence may be used to allow an end user to view data that was just sent to the spreadsheet. For more information refer to the menu_call() function listed in Appendix A.

When you want to return to the original module, use the module_return() function. For example:

```
EXTERNAL module_return()

module_return()
```

This command will cause the original module to be invoked.

Note: If changing modules during a processing routine, special steps must be taken. Refer to the section titled “Changing Modules During a Processing Routine” for details.

Customizing Login and Start Programs

The source code files for the LOGIN and START programs are included in the Developer System directory. These may be altered in order to customize the start up or login process. If files by these names are found by the RAD App in the home directory of the application, they will be executed in place of the versions found in the Shell directory ($$advr). Copy one or both of these files into the home directory and then modify the copies, if you intend to customize these processes.

WARNING: Customizing the START and LOAD SPL programs could make upgrading to a new version of RAD a difficult process.

Passwords on Data Files

It is a common requirement to allow access to files only when running an application.

Under normal circumstances, a knowledgeable end user can start the database and manually load application data files, thus by-passing the security and protection of the application. In many situations, this is not a problem.

To stop uncontrolled access to files, you can attach passwords to them. The application can be programmed to invisibly supply the password when required.

Custom-views, data files, documents and worksheets can all have passwords attached to them. Usually, when a file has a password, the data is also encrypted.

To use a password protection scheme, attach passwords to the files and create a function in your application library which will perform all file loading and supply the required passwords.

Attaching Passwords

To attach a password, follow these steps:

Load the view

If you are running the application, access the SmartWare module with the Developer's Utilities command.

Use the File Password command and attach a new password.

Alternatively, for data files and views, use the File Structure Maintenance, Tools, Password command.

In the database, you can attach passwords to custom views, standard views and data files. For maximum security, protect each one. If you attach a password to a data file, you may decide to encrypt the data.

Programming the Application to Supply Passwords

Once passwords have been attached, the application must be modified. A function in the application library will be called each time a file is loaded. It is the responsibility of this function to load the file and, if appropriate, use the Keys command to supply the password.

To install this change, follow these steps.

Access the application library, through the Developer.

Read in the password code segment provided in the DEV directory (For example, in the program editor: Alt F3, \angoss\apsys\dev\password.pf3). If you already have functions in the library, rearrange the code that was read in.

Next, customize the new loading function to calculate the appropriate passwords and make any other customizations.

Chapter 7: Miscellaneous Information

RAD Colors

Menus

There are five defined colors that are consistent for both menu types. These are:

Text Foreground

Text/Menu Background

Highlighter Foreground

Highlighter Background

Border Foreground

Access to colors for each menu is accomplished through the Developer's Modify Menu Characteristics commands and, global menu changes, are made through the Developer's Utilities Preferences Menu_Colors commands. In both cases, the default value for each color is -1. This value tells the RAD application to derive the color from whatever the system (SmartWare) colors are at execution time, ensuring that colors will be appropriate for whatever OS you are running.

Menu colors can also be controlled by user modes. Access is provided through the Defaults Global command sequence in User Mode Maintenance.

Data Views - Screen Colors

Access to colors in data views is accomplished through the Developer's Modify Database Information commands. This produces an information screen for a large variety of data view controls. Paging down to the final screen displays the Screen Default Colors.

When a value in a Screen Default field is blank, the color for that screen element will be the one that was set and stored in the custom view. If a value is -1, the screen color is derived from the menu. Any other value (between 0 and 15) sets the color to that value.

In the last two cases, this works because the custom view is loaded with repaint off and then painted with SmartWare's Window Paint commands.

The view editor's Paint commands, override any other settings.

Icons

Adding Icons

When running in graphical mode, icons are used on navigation menus and on the toolbar in conjunction with pulldowns.

The graphical navigation menus (menu6) will always draw an icon beside the textual description. If an icon is not specifically identified, an icon based on the choice's action is used. For example, if a choice leads to a menu, the icon is a picture of a menu.

Icons can also be found on a toolbar, usually just below the menu bar. At views, these toolbar icons can be used to drive the application (e.g. Update Record) otherwise, they are used to select system tools. By default, some of the most common actions on a view will produce a toolbar icon. Bitmaps, based on icon names (e.g., print.bmp), stored in the shell/icons directory can be overridden by placing them in the RAD App's icon directory.

Toolbar icons can be one of two sizes: 32x32 pixels or 16x16 pixels. The 32 pixel size is the same as that used on the graphical navigation menus. RAD Apps will automatically switch the toolbar icons to the smaller size if the larger icons do not fit within the height of two text lines. Note that you can force the toolbar size by adding a ##icon_size line to the angoss.cfg file, specifying the size as either 32 or 16.

A developer can customize the icons. First, you must bind an icon to a menu choice. To do this, access the Developer System and select Modify. The Modify menu lists every choice on the current application menu. Select the appropriate choice. If there is a sub menu, select Item Information.

The “Menu Information - Current Choice” dialog box is displayed. Enter a file name for the bitmap and use the Icon Editor button to edit the specified bitmap.

Icon Editor

The Icon Editor is a fat bitmap editor. This means that you can select a color from the color bar and paint individual bits by clicking or dragging the mouse in the edit window.

As described in the Adding Icons section, icons can be either 32x32 or 16x16 pixels in size. When editing from navigation menus, icons in the editor will use the larger size however, when edits are performed in a view, the size of the icons depends on a number of factors.

Edited icons are stored in one of three directories below the system directory:

..\system\icons General icons

..\system\icons\32 32x32 icons

..\system\icons\16 16x16 icons

The Icon Editor's commands are:

Import

Imports another bitmap into the edit window.

Clear

Sets all bits to the currently selected color effectively clearing the icon in the edit window.

Save

Files are saved as bitmaps.

Change-Color

Changes the color of certain bits to the currently selected color. To do this: set the current color, select Change-Color, and click on one of the bits in the edit window.

Delete-Icon

Removes the bitmap file.

Toggle-Size

Toggles between 32x32 and 16x16 bitmap sizes. If the current bitmap does not exist in the appropriate directory, it is converted otherwise the new bitmap size is simply loaded.

How Data Paths are Handled

A properly designed RAD application, should not use hard coded paths. All paths should use one of the public path variables. These variables are either set by the start up script, the user's infoset, or are calculated. The following table lists the path variables, their use and how they are set. Note that, with the information set tools available, you may create new path variables.

Variable	Source	Description
$$adev *	calculated	The Developer location. Can be overridden in angoss.cfg.
$$advr	calculated or ANGIIDVR environment variable	Shell location. Note that if an alternative language is used, this can be overridden after the application starts. Calculation overridden by ANGIIDVR.
$$homedir	calculated	Home Directory of the application. For compatibility, can be specified in ANGIIAPL environment variable.
$$macpath	infoset	Location of end user defined macro definitions. Default: $$homedir|"macro".
$$merpath	infoset	Location of end user defined merge definitions. Default: $$homedir|"merge".
$$persdir	user file	User's personal directory. Used for send to spreadsheet and print to disk. For personal data files.

Variable	Source	Description
$$quepath	infoset	Location of end user defined query definitions. Default: $$homedir|"query".
$$reppath	infoset	Location of end user defined report definitions. Default: $$homedir|"report".
$$senpath	infoset	Location of end user defined sort definitions. Default: $$homedir|"sort".
$$sysfiles	infoset	Main application data files directory. This defaults to $$homedir|"files".
$$sysfiles2 $$sysfiles3 $$sysfiles4 $$sysfiles5	infoset	Other application data files.See $$sysfiles above. These also default to $$homedir|"files".
$$syspath *	calculated	Location of RAD application system files. Can be overridden by angoss.cfg.
$$tpltpath	infoset	Location of DDL, FPR templates. It should be set to $$homedir|"template".
$$workdir	(user id)	System work space. One directory for each user to avoid conflict. Calculated by $$sys-path|"users"|$$dirsep| <user id>.

* The angoss.cfg file in RAD Apps created prior to version 2.65 may contain path which should be removed.

Moving Systems

To move a RAD App, first copy the entire directory tree to the new location. If path names have not changed then the application should run without a problem. If path names have changed, you may need to modify the following configuration files.

DOS Startup Batch File

This batch file contains two hard-coded application paths that may need to be changed. For example, the radhr startup batch file created in the tutorial contains the lines:

```
if not exist C:\RADHR\system\users\%ANGIIUSR%.0\osapl.bat goto end
angossx -RADC:\RADHR  -omC:\ANGOSS\apsys\shell\angoss.rcm
```

The C:\RADHR reference in both lines should be changed.

Information Set

If you need to change variables derived from the information set by the RAD App, but you cannot load the system because of this, you can use a special login utility.

When you start the application select a user ID of PATHSET. You will then need to enter the administrator password. You will then be taken to the standard (default) information set where you can edit the paths. Any changes will be recorded and the application will terminate. You can then restart the application.

You may also delete the STANDARD.FSI in the <homedir>|"system" directory. If you do this, then when the application restarts, it will be rebuilt with default values. If you have changed the default path values, you must go into the File Structure Maintenance selection and select the Info_set command. From there you edit the path names.

To move the ANGOSS directory (including the Developer and Shell), you may need to modify the following configuration files.

DOS Startup Batch File

This batch file contains one hard-coded ANGOSS path that may need to be changed. For example, the radhr startup batch file contains the line:

```
angossx -RADC:\RADHR  -omC:\ANGOSS\apsys\shell\angoss.rcm
```

The C:\ANGOSS reference should be changed.

ANGOSS Configuration file

Though not a default condition, the location of RAD and its Shell may be set in the angoss.cfg file. If there are any lines that begin with $$adev or $$syspath these may need to be changed.

NOTE: If you delete an angoss.cfr file, it will be rebuilt with default paths and values.

System Database Paths

It is recommended that you do not move any of the system data files relative to the home directory. However, for the adventurous, the following table indicates the default data paths, and various notes about moving them.

View	Path	Notes
appdoc	.\files	May be moved
archjobs	.\files	May be moved
archlog	.\files	May be moved
dsl	.\system	DO NOT MOVE
dwl	.\system	DO NOT MOVE
dv_info	.\system	DO NOT MOVE
db_info	.\system	DO NOT MOVE
info_set	.\system	DO NOT MOVE
language	.\system	May be moved, must be same as user
loc_list	.\system	DO NOT MOVE
menu	.\files	May be moved
proced	.\files	May be moved (linked to user)
proclog	.\files	May be moved (linked to user)
rel_info	.\system	DO NOT MOVE

View	Path	Notes
steps	.\template	DO NOT MOVE
sysactlg	.\system	May be moved (linked to user)
sysdef	.\files	May be moved
user	.\system	May be moved, but you must customize LOGIN.PF3
usermode	.\system	May be moved, must be same as user

You must physically move the file and change references in the dv_info data file.

Upgrading a User's System

This section will discuss the procedures necessary for upgrading a client's RAD App with your modifications. This assumes that you have made modifications to a production copy of an application and wish to update a live, working copy (a user's application). This procedure should be followed step by step to ensure that there is no damage done to the existing user's Application.

Backup the System

The **first and very important** step when upgrading an application is to make a complete backup of the current client system. By maintaining the backup, any errors can be corrected without data loss.

Restructuring Modified Data Files

Installing modified data files involves removing test data from the new data file and transferring the real data into the new data files. The steps for a single modified file are outlined below, however if modifications are minor, it may be easier to alter the file directly on the client system.

- Check the modified file for test data. If data is found, remove it by performing a REPLACE DELETE query and a purge.
- Move to the client's files directory (e.g., \CLIENT\FILES).

- Rename the client's data files to a new file name (e.g., rename OLD_DATA.* NEW_NAME.*).
- Copy the modified data files into the client's files directory.
- Start the SmartWare database module.
- Renaming a data file will prevent it from being loading, so we must File-Fix the view (standard) to be able to load it under the new name. SmartWare will inform you that the view already exists. Reply Yes to the rebuild option.
- Load the standard view of the client's file and then load the standard view of the modified file.
- With the Data Utilities Append command, restructure the records in the client's file to the modified file. A screen appears with source fields and matching destination fields indicated. Any new fields that should contain data from the client's data file, should be added to the destination list. Refer to the Database manual for instruction on the Append command.

When completed and tested, the renamed files can be deleted to free up disk space.

Copying Program and Definition Files

The next step is to copy the system data files if needed. If a data view has been created or altered (files loaded in the background, new keys etc...) then the following files should be copied into the system directory of the application:

dv_info.*

db_info.*

rel_info.*

If Database links to the Spreadsheet or Wordprocessor (DSL's and DWL's) were created or modified then the following should also be copied into the system directory:

dwl*.*

dsl*.*

Individually changed definitions should be copied over as needed. To determine the required definitions to copy, refer to the changes.txt log, or you can use SmartWare's

Tools Directory Display command and sort the files by date (F3) to indicate the most recent files. Check both the home and system directories.

After the above files have been copied, any altered menu files (*.MNU) should then be copied into the system/menus directory. When menus are changed (i.e.: Inserted, deleted, repositioned, etc.) it is recommended that you check the date of the menu files (*.mnu) to confirm that all new files are copied to the client's system.

Before copying menu files to the client system, it is important that the menus contain the same number of user modes as the client application. Menus that are copied into the client system may not have the user mode restrictions that were assigned by the client. This will require that User Mode Maintenance be performed on the menu once again. If the menu changes are minimal and the client has a valid copy of the Developer, then modifications to the menus can be performed directly.

Copy upgraded program files, definitions etc... into the home directory of the client's application. Any other upgrades should also be copied into their appropriate directories.

Test the System

When all the upgrades have been installed, the client's application should be started and tested. This testing should not only include the upgrades, but also the standard system to ensure that one upgrade does not affect another part of the system.

Appendix A

Function Listing

A wide variety of functions are available to programmers. Most of these functions form an integral part of the RAD App. To use these functions in your own programs, simply list them as external functions before the main section of the program and then follow the syntactical instructions laid out in this section.

The arguments in the syntactical example (listed beside the word Format:) are preceded with either a dollar sign ($) or a pound sign (#). The dollar sign indicates that the argument is a text string, while the pound sign indicates a numeric value is expected.

Arguments preceded by both ($/#) can be either numeric or text values.

ACTIVE_LEV

Format: active_lev(#menu_level)

Module: all

Return: numeric - level of current view

Returns the level (or position in the menu stack) of the currently active data view, spreadsheet or document view. Hence, if a number of menus are called from a view, you could call this function with the current level as the argument, and the returned value would be the level of the view from which all of these menus were ultimately called. This function is used for determining which $$stat[] array element should be used for reading and writing to the database, spreadsheet or document. The following example determines the status at the level of the currently-active view with respect to the current level (which is held in ##lev).

```
$status=$$stat[active_lev(##lev)]
```

Refer to Appendix B for a discussion of the menu stack and usage of the $$stat[] array.

ADD_LOG

Format: add_log()
Module: database
Return: numeric - Process ID

Records an entry in the *Batch Update/Process Log* view. It obtains its information for the entry from the public variables $$desc, $$obj, $$sysid and $$usrid. Use the returned Process ID as a parameter in COMPLETE_LOG to close the log entry.

Related function: COMPLETE_LOG()

ASK_FILENAME

Format: ask_filename($extension, $title, #create)
Module: all
Return: text - name of selected file or 0 if Esc is pressed

Specifies a group of files from which the user selects one. The extension parameter is a space separated list of extensions that determines which files are to be displayed to the end user. The title is displayed as a prompt at the bottom of the selection box. The path of the files is assumed to be the value of the function expression PATH(datapath) which is the current data path. Note that a developer can issue a Tools Directory New-Directory before calling this function in order to change the path. The end-user may override this path if desired. If create is set to zero, then the file the user selects must exist. If set to one, the user may create a file by typing a name which does not exist. If the user presses the escape key then this function will return a 0. Note that the path filename and extension (e.g., c:\oe\test.dfr) is returned.

```
$dfr=ask_filename( "dfr", "Reports", 0 )
```

ASK_FILENAME_DIR

Format: ask_filename_dir ($file_spec, $title, #create)
Module: all
Return: text - string of path and file name or 0 if Esc is pressed

Calls the ask_filename function however, its format is sometimes more useful. The $file_spec parameter must be a text string with a path, a * for the filename and the

extension. The $title is used as prompt and #create indicates if the user may specify files that do not exist.

```
$file = ask_filename_dir ( "c:\myfiles\*.doc", "Enter:", 0 )
```

CHAP

Format: chap($directory)
Module: all
Return: 0 or 1

Changes execution to another RAD App. The argument specifies the home directory of the target application. When an end user exits the target application, the original system will be reloaded, and the user will be returned to the point in the application from where this function was called.

Warning: Be sure to specify a directory with a working RAD application otherwise unpredictable results can occur.

CHECK_FILE_NAME

Format: check_file_name($filename)
Module: all
Return: numeric - status (0 = invalid name, 1 = valid name)

Checks whether the argument text string is a valid file name (path and extension cannot be included). If it is valid, a value of one (1) will be returned. If invalid, **an error message will be displayed** and a zero value will be returned.

This example will continue to ask the user for a file name until a valid one is entered.

```
WHILE TRUE
    SCREEN SHORTINPUT $fn "Enter file name:"
    IF check_file_name($fn) = 1
            EXIT WHILE
    END IF
END WHILE
```

CHECK_PATH

Format: check_path($path)
Module: all
Return: text - corrected path

Prepares the argument text string to be concatenated with a file name by appending a slash or back-slash as required. Note that valid filespec characters are not checked.

```
if file( check_path($path)|"myfile.txt" )
    ...Process MYFILE.TXT
end if
```

CHOICE_TITLE

Format: choice_title($menu, #choice)
Module: all
Return: text - choice description or error

Returns the description of the specified choice as it appears on the application's menu. The string "MENU DOES NOT EXIST" is returned if $menu is invalid. However, a **run-time error** will occur if the #choice value is out of bounds for the specified menu.

The first parameter is the short name of the menu on which the choice in question appears. The second parameter is that choice's number on the menu. The description is always in the default language, even if another language is currently active. The choice number searches for the *physical* location of the choice, making no provisions for choices which have been removed from the menus of certain user modes.

This example returns the name of the first item on the Main Menu.

```
$desc = choice_title( "mm", 1 )
```

CLEAR_KEYBOARD

Format: clear_keyboard()
Module: all
Return: 0

Clears any key strokes from the keyboard buffer.

CLOSE_CUR_MENU

Format: close_cur_menu(#write)
Module: all
Return: numeric - status (1 = successful, 0 = failed)

Closes and optionally writes the currently opened menu.

CLOSE_ALLWIN

Format: close_allwin($text_file)
Module: database - file handle ##wrkhnd
Return: numeric - status (1 = successful, 0 = failed)

Unloads all views (except for DV_INFO). If a text file is specified, it saves the list of open files including (DV_INFO) so that a reload may be performed. A failure occurs if the text file cannot be created.

```
close_allwin( $$workdir|"save.win" )
```

Related function: RESTORE_ALLWIN

CMENU_ADD
CMENU_FREE
CMENU_INIT
CMENU_MAKEMNU
CMENU_RUN

Command Menu Interface - COMMANDLIST replacement.

This series of SPL functions simplify the interface to the pulldown menus and is an improved alternative to the old commandlist() function. These functions allow developers to quickly specify and run pulldown menus or command lists.

Note: If you are not using these commands from within RAD, you must load the RAD libraries by executing the following line before using these functions:

```
execute path(systpath)|"loadalib.rf0" in-memory
```

Constructing a menu

The functions discussed here allow you to quickly build a menu. In essence, they are a simplified alternative to the mnu_open() and mnu_insch() functions. Note that these functions are written in SPL and, in fact, use the mnu_ functions.

cmenu_init

Format: cmenu_init()
Module: all
Return: numeric

This function starts a new menu. It returns 0 for fail, 1 for success.

cmenu_add

Format: cmenu_add($desc, $act, $linehelp, $helpfile)
Module: all
Return: none

Adds an item to menu being built. The following parameters are used.

$desc - Command description. Use -> to specify levels. For example “File->Exit” creates a command named exit under a submenu named file.

$act - Command action. A string that can be used to identify the choice. If left blank then the description of the choice is returned.

$linehelp - The autohelp line for the current choice.

$helpfile - The base file name of the file containing help text for this command.

cmenu_makemnu

Format: cmenu_makemnu()
Module: all
Return: numeric

This function builds a menu, using mnu_info() related functions, based on the information previously provided with cmenu_add().

It returns a menu handle or 0 if a failure occurred.

cmenu_free

```
Format: cmenu_free( #menu )
Module: all
Return: none
```

Frees the memory used by a menu file. It is the same as mnu_free().

cmenu_run

Format: cmenu_run(#menu,#nav,$key_cb,$com_cb,$h_pth,$h_ext,$#datal)
Module: all
Return: none

The cmenu_run() function displays a menu and responds to the end user. The menu can be built with the above listed cmenu functions or with the mnu_info() related functions. For a “quick-and-dirty” menu, the menu choices can also be passed as a text string. Here is the format of cmenu_run()

#menu - The menu handle or a space separated text string listing commands.

#nav - Navigation keys. If it is set to 1, then the end user may use the mouse or keystrokes to move around the database, spreadsheet or document. Otherwise, use 0.

$key_cb - Keystroke call back function. If you want your own function to respond to mouse events, keystrokes, etc. then list the name of the public function to deal with these events. See below for more details. If you do not want to use it, simply enter "".

$com_cb - Command call back function. It is best if you specify a function that will be called when a command is selected. This way, the processing overhead to set up the menu only need be run when the menu is entered. See below for details. If you do not specify this parameter (i.e., enter ""), then cmenu_run() will terminate when the user selects a choice, returning the action (or description) of the choice selected.

$h_pth - Help path. This must be a directory name which contains help files for the menu.

$h_ext - Help file extension. This is the three character file extension that the help files for this menu will have. When a user requests a help file the path ($h_pth), file name (from cmenu_add()) and the extension ($h_ext) are combined to calculate the file to display.

$#data - This parameter is passed to the call back functions. You may use it however you want. If more than one data element is required, consider using an array pointer.

If you are running in character mode, (##pulldowns=0) then a command list will be used. A command list is a ring style menu occupying the bottom portion of the text window. If you are running in graphics mode, (##pulldowns=1), then pulldown menus, which sit above the text window, are used.

Command Call Back Function

The command call back function, which is passed as an argument to cmenu_run(), is called when the end user selects a command. You must write this function yourself. It must be a public function and it must have the following arguments:

```
function my_callback($act,$#data,#menu,#choice)
```

$act - This is the "action" of the menu choice selected as specified by the cmenu_add() or MNU_ACT. Typically, your function will branch based on this value.

$#data - This is whatever was passed to cmenu_run().

#menu and #choice - The #menu and #choice parameters are usually not used. They can be used to further enquire about the menu choice selected. For instance the password, help file, object, etc. can be obtained by passing these choices to mnu_info().

The command call back function should usually return 0. If it returns a value other than 0, cmenu_run() will terminate, returning the same value that you returned. So, when the user selects a choice that should terminate the menu, return a value other than 0.

Key Call Back Function

When a key call back function is specified in cmenu_run(), each time the user presses a key or uses the mouse this function is called. As usual, the call back function must be public. The key call back function requires this format:

```
function my_callback(#key,$#data,#menu)
```

#key - Is the event id as returned by the EVENTINFO function.

$#data - This is whatever was passed to cmenu_run().

#menu - The #menu and #choice parameters are usually not used but can be

used by mnu_info() to obtain more information about the current menu.

The key call back function must return 0 if the event was not handled, 1 if the event was handled and 2 if you want the menu to terminate.

Examples

Each program using the cmenu functions must include the following declarations:

```
' Menu Building
external cmenu_init()
external cmenu_add()
external cmenu_makemnu()
external cmenu_free()
' Menu Run
external cmenu_run()
global simple()
global very_simple()
global still_simple()
global complex()

' If you are not using RAD, the functions must be loaded by:
execute path(syspath)|"loadalib.rf0" in-memory

' VERY SIMPLE EXAMPLE
'*******************************************
function very_simple()
local $ret
    $ret =  cmenu_run("Run Exit Copy Delete", 0, "","","","","")
    ' Call a command menu with above commands, no data base navigation
    case $ret
            when "Run"
                    message $ret
            when "Exit"
                    message $ret
            when "Copy"
                    message $ret
            when "Delete"
                    message $ret
            otherwise
                    message "huh:"|str($ret)
    end case
end function
```

```
' SIMPLE EXAMPLE
'*********************************************
function simple()
local $ret #menu
    ' Build a Menu
    if cmenu_init()
            cmenu_add("Run","r","","")
            cmenu_add("Exit","x","","")
            cmenu_add("Copy","c","","")
            cmenu_add("Delete","d","","")
            #menu = cmenu_makemnu()
            $ret = cmenu_run(#menu, 1, "","","","","")
            ' Call command menu, with buildmenu with navigation
            cmenu_free(#menu)
            case lower($ret)
                    when "r"
                            message $ret
                    when "x"
                            message $ret
                    when "c"
                            message $ret
                    when "d"
                            message $ret
                    otherwise
                            message "huh:"|str($ret)
            end case
    end if
end function

'AN ADVANCED EXAMPLE
'*********************************************
public my_commands()
public $mydir
function still_simple()
local #menu $ret
    $mydir = ""
    if cmenu_init()
            cmenu_add("File->Run","run","run a program","run")
            cmenu_add("File->Exit","x","quit this program","quit")
            cmenu_add("File->Whatever","what","quit","quit")
            cmenu_add("Edit->Copy","cpy","copy something","cpy")
            cmenu_add("Edit->Delete","del","delete something","del")
            #menu = cmenu_makemnu()
```

```
            $ret = cmenu_run(#menu,1,"","my_commands",$mydir,"txt","")
            cmenu_free(#menu)
    end if
end function

' 0=continue, anything else exit
function my_commands($action,$data,#menu,#choice)
    case $action
            when "run"
                message $action
            when "cpy"
                message $action
            when "x"
                message $action
                return 1 ' This terminates menu
            when "del"
                message $action
            when "what"
                message $action
            otherwise
                message "action:"|$action
    end case
    return 0
end function
```

'COMPLEX EXAMPLE

```
'*********************************************

public my_keys()
' Also uses my_commands from previous example
function complex()
local #menu $ret
    $mydir = ""
    if cmenu_init()
            cmenu_add("File->Run","run","run a program","run")
            cmenu_add("File->Exit","x","quit this program","quit")
            cmenu_add("Edit->Copy","cpy","copy something","cpy")
            cmenu_add("Edit->Delete","del","delete something","del")
            cmenu_add("File->Whatever","what","quit","quit")
            cmenu_add("Data->Find->Name","f name","find name","fn")
            cmenu_add("Data->Find->Phone","f phone","find phone","fp")
            cmenu_add("Data->Order->Name","o phone","order name","on")
            cmenu_add("Data->Order->Phone","o phone","order phone","op")
```

```
            #menu = cmenu_makemnu()
            $ret=cmenu_run(#menu,1,"my_keys","my_commands", \
                    $mydir,"txt","Hi There")
            cmenu_free(#menu)
    end if
end function

function my_keys(#key,$data,#mnu)
    case #key
            when {Alt-H}
                    message $data
                    return 1 ’ Thank you, I have dealt with event
            when {Alt-Q}
                    message "bye alt-q"
                    return 2 ’ Let’s get out of menu
            when {f10}
                    message "bye f10"
                    return 2 ’ Let’s get out of menu
            when {esc}
                    message "bye esc"
                    return 2 ’ Let’s get out of menu
    end case
    return 0 ’ I don’t know about this event
end function
```

COMPLETE_LOG

Format: complete_log(#process number, $comments)
Module: database
Return: 0

Adds a comment and closes a *Batch Update/Process Log* view entry. The process number must be the value returned by the ADD_LOG() function otherwise the comment will be written to an unintended log entry (most likely the first entry in the view).

Related function: ADD_LOG()

COUNT_WORDS

Format: count_words($text)
Module: all
Returns: numeric - number of words

Counts the number of words in a text string. A word is considered to be any sequence of characters not containing a space. If $text is a null string or a number, 0 is returned.

```
#words = count_words( $str )
```

DECODE_EDA

Format: decode_eda($edit_area)
Module: spreadsheet
Return: 0

Parses the $edit_area string and populates the global $area[] array with those edit area specifications. The $area[] array is used by the EDIT_AREAS() functions. The format of the edit area string is:

```
block<outside rows,outside cols>inside rows,inside cols;...
```

Block specifies a block of spreadsheet cells that define the edit area. The specification must be in standard SmartWare expression format: either an explicit reference (e.g., r1:3c2:19) or the name of a predefined block. Named blocks are defined with the SmartWare command sequence Sheet Name Define. The outside and inside area locks are optional; these specify the number of rows and columns to lock, outside and inside the defined block respectively. The outside lock, if specified, must be preceded with a less-than symbol (<). The inside lock specification, if included, must be preceded with a greater-than symbol (>).

DEF_EDIT_AREAS

Format: def_edit_area($default_areas)
Module: spreadsheet
Return: text - modified areas string

Interface that allows an end user to define edit areas. The edit areas screen is displayed where the user may type in edit area specifications or choose them from the spreadsheet with the block-marking technique. Edit areas are initially set to those specified by the default areas string. A string specifying the user-selected area specifications is returned.

Note that, because this function presents an interface to the user, your program will be halted.

DISPLAY_MESSAGE

Format: display_message($string1, $string2, #mode)
Module: all
Return: 0

Displays a one or two-line message. The mode parameter is used as shown below:

0 - Screen only - no key press needed (will not halt program execution)

1 - Key press and restore screen

2 - Key press and clear

3 - Message dialog box with OK

4 - Mode 1 or 3 depending on ##pulldowns

DO_CALC

Format: do_calc($definition)
Module: database
Return: 0

Executes a totals definition and displays the results on the screen. It operates on the current data file in its current state - if a standard Query command had been issued to

select a subset of records, only these records are totalled. If the totals definition is not found, and error message to that effect is displayed. However, a SmartWare error message can occur when the totals definition is invalid.

```
do_calc( "totsales" )
```

EDIT_AREAS

Format: edit_areas($areas, #start_block, #row, #col, $fkey, #return_fkeys)

Module: spreadsheet

Return: numeric - value of key stroke used to exit

Interface that allows an end-user to modify the contents of the specified edit areas on the current worksheet. The areas string specifies the edit areas using the format described in the DECODE_EDA function. The row, column and block number in which the cursor is initially placed are specified by the #row, #col, and #start_block parameters. A description of key stroke functionality can be supplied with the $fkey parameter. If the #return_fkeys parameter is set to 1, then the user key stroke is returned if F3, F4, F6 or F9 is pressed. Otherwise, these keys are ignored.

Note that, because this function presents an interface to the user, your program will be halted.

The returned key stroke is the value of the key so use the brace codes for testing purposes. For example, if the user exits the interface with the F3 key, you can trap this value with the {F3} brace code.

```
#exitkey=edit_areas("blk1<1,1;r1:10c2:20>2,2",1,1,1,"F3=Special",1)
IF #exitkey = {F3}
    ...
ELSEIF
    ...
END IF
```

ERRORREP

Format: errorrep($path_file)
Module: all
Return: 0

Asks the end-user to indicate whether an error report should be printed or displayed and then performs the appropriate action. Typically, this is used in an audited program and is, in fact, placed in audited programs by default.

Note that, because this function presents an interface to the user, your program will be halted.

```
errorrep( "c:\test\system\errors.txt" )
```

FIND_COMMAND

Format: find_command($task)
Module: database
Return: 0

Presents the Find command interface based on the current view. This always returns 0.

Note that, because this function presents an interface to the user, your program will be halted.

FINDRECORD

Format: findrecord($key_menu, $keys, #view)
Module: database
Return: 0

Displays a list of the current view's key fields, allowing an end user to select one on which to search based on the user's comparison data. The first record meeting that criteria is made current.

The $key_menu parameter is a string listing each key field name, separated by spaces. Any spaces in the actual key field names must be replaced with an underscore character. The $keys parameter lists the key field names again, however, spaces

within field names are NOT replaced with underscores. The #view parameter indicates whether the appropriate $$stat[] array entry should be updated to reflect any changes in the order of the data file. It also determines whether the variable $$kytp should be used to indicate if and when the user's input should be automatically converted to upper case, lower case, proper case or masked. A value of 1 indicates that the $$stat[] array should be updated and the $$kytp variable dictates any input case conversions or masks.

Warning: this function requires valid key field names otherwise a SmartWare runtime error or series of errors will occur.

```
findrecord( "Part_# Part_Type", "Part # Part Type", 0 )
```

FREE_CELL

Format: free_cell()

Module: spreadsheet

Return: numeric - status (0=locked, 1=free)

Checks the current cell and returns a value of 0 (False) if it is locked and 1 (True) if it is unlocked.

GET_ANGII_VER

Format: get_angii_ver(#product, #info)

Module: all

Returns: text - version and/or build information

Obtains the version and build information. Set the #product parameter to 0 for application environment information and 1 for Developer information. Set the #info parameter to 0 for the version number, 1 for the build information or 2 for both. If an error occurs, a question mark (?) will be returned.

GET_CHOICE

Format: get_choice($str1, $str2, $st3, $str4, $choices, #mode)

Module: all

Return: numeric - selected choice

Displays message and prompts for selection. Four message strings are allowed - these are the $str*n* arguments. The $choices argument is a space delimited list that contains the items in the menu. The mode parameter is used as shown below.

1 - Key press and restore screen

2 - Key press and clear

3 - Message dialog box with choices as buttons

4 - Mode 1 or 3 depending on ##pulldowns

The return value is the number of the user selected choice or 0 if escape is pressed. In mode 3, the F10 key also returns 0.

```
#output = get_choice("Output type","","","","Printer File Screen",3)
if #output = 1
    'printer code ...
elseif #output = 2
    'file code ...
elseif #output = 3
    'screen code ...
else
    'esc (or F10 in mode 3) pressed
end if
```

When GET_CHOICE is run in mode 1, the menu choices appear as a first letter list delimited with slashes, such as (p / f / s). Because the user will make selections with the keyboard, it is important to make each choice begin with a different letter and to describe the choices in the $str*n* arguments. This is not required in mode 3.

Note that, because this function presents an interface to the user, your program will be halted.

GET_FILE

Format: get_file($pth,$vw,$typ,#cur,#excl,#retry,#hnd,#ind)
Module: database
Return: result string

This RAD function loads the specified view or makes it current if the view is already loaded. This last action assumes that the #cur parameter is set - see below.

Parameters:

$pth - The path of the view. If its value is a null string, the path is determined by the RAD application's internal data (i.e., dv_info). NOTE: If the view is already loaded, the path parameter is ignored. In cases where the application uses multiple views with the same name, it is safer to unload the view and call GET_FILE again - see the Return Strings section below. If the path parameter is a view name prefixed with a question mark (e.g., "?customer"), GET_FILE will find that view's path (again in dv_info) and use it. Typically, this is done when loading a standard-view that is normally attached to a custom-view but does not share the same name.

$vw - The view name. Do not include the path or extension.

$typ - Indicates the type as custom-view or standard-view. Valid options for this parameter are "vw" for custom-view and "vws" for standard-view.

#cur - Makes the view current. Valid options for this parameter are 1 (view is current) and 0 (view is active).

#excl - Loads the view as exclusive - this prevents other users from loading the file. Note that the #cur parameter must be true (1). NOTE: If the view is already loaded but is not exclusive, unload it before calling GET_FILE again, otherwise the view will not be reloaded - see the Return Strings section below.

#retry - Number of load attempts. If this parameter is set to 0, the program is interrupted and the user is prompted to retry or abort the load attempt. A value greater than 0 indicates the number of uninterrupted retries. In this case, after each failed load attempt, the function pauses before retrying. This pause is increased by one minute per attempt (e.g., after the ninth attempt, the function waits nine minutes before retrying).

#hnd - Message file handle. When this parameter is greater than 1, mes-

sages are written to the opened file represented by the handle. The program will not be interrupted when errors occur. Typically, this is used in an AUDITED PROGRAM where the standard handle passed to GET_FILE is ##mhnd. Messages are then written to the Batch Update/ Process Log. When the handle is set to 0, error messages are displayed on the screen while the program waits for user input.

#ind - Indentation level for messages.

Return Strings:

"error" - Unable to load the view.

"loaded" - GET_FILE loaded the view.

"already loaded" - View was previously loaded.

Example 1:

The code fragment below loads the view, if it is not already loaded, and makes it current. Exclusive mode is not invoked, load retries are prompted, and error messages are displayed on the screen. For more advanced usage, refer to the second example.

```
FUNCTION Init()
LOCAL $view
    $view = "customer"
    IF get_file("",$view,"vw",1,0,0,0,0) == "error"
            RETURN 0
    END IF
    '... other initiation code
    RETURN 1
END FUNCTION
```

Example 2:

The code fragment below was extracted from an AUDITED PROGRAM. It loads the inventory custom-view exclusively. In case of loading errors, six attempts will be made to retry the load.

Note that, because the view must be loaded exclusively, an unload is performed if GET_FILE returns "already loaded". The WHILE loop then causes a reload.

```
FUNCTION UpdateInventory()
LOCAL $view $result
    $view = "inventry"
    WHILE TRUE
```

```
            $result = get_file("",$view,"vw",1,1,6,##mhnd,2)
            IF $result == "already loaded"
                    FILE UNLOAD VIEW $view|".vw"
            ELSE
                    EXIT WHILE
            END IF
    END WHILE
    IF $result == "error"
            rec_message( "Could not load" & $view )
            rec_message( "* Critical view - Batch aborted" )
            RETURN 0
    END IF
    '... code to update view
    RETURN 1
END FUNCTION
```

Notes:

While GET_FILE can be used to load views in any program, it has a number of features that make it valuable in Batch Processing programs where the program runs unattended. The message and load retry facilities are ideal in that, no program interruption is required. Further, the exclusive loading mode eliminates potential LOCK RECORD problems and other update record issues.

GET_LINE

Format: get_line($str1, $str2, $st3, VP_$str, $default, #len, #mode)
Module: all
Return: numeric - exit status (0 = esc/cancel, 1 = value accepted)

Displays message and prompts for input. Three message strings are allowed - these are the $str*n* arguments. The VP_$str argument is a pointer to a string variable which will contain the user's input text. The $default parameter specifies a default value for the text and the #len argument specifies the text's maximum length. The mode parameter is used as shown below.

1 - Key press and restore screen

2 - Key press and clear

3 - Message dialog box with choices as buttons

4 - Mode 1 or 3 depending on ##pulldowns

The return value is the key or button selected by the user to complete the prompt. If escape is pressed or cancel selected, GET_LINE returns 0. If the value is accepted, 1 is returned.

```
VP_$str = VARPTR($str)
#keypress = get_line( "Enter Code", "", "", VP_$str, "TNAILS" , 8, 3 )
if #keypress = 0
    return 0
end if
message "The code you entered was:" & $str
```

Note that, because this function presents an interface to the user, your program will be halted.

GET_PATH

Format: get_path($short_name)
Module: spreadsheet, database - Uses: file handle ##wrkhnd
Return: text - path name

Returns the path of a spreadsheet or data view whose short name is specified by the parameter. If more than one data view or spreadsheet is associated with the menu, then it is the path of the primary view or worksheet which is returned.

```
$path = get_path( "disctab" )
```

If the shortname does not exist, a SmartWare run-time error or series of errors will occur.

INT_PATH

Format: int_path($pathvar)
Module: all
Return: text - decoded path

Interprets coded paths. Such paths utilize public variables, which contain system-wide path names, so as to avoid hard-coding these paths. These variables are converted into explicit paths. To mix explicit path specifications and coded specs, separate the variable name and the explicit portion by a single space.

KEY_FIELD_PROMPT

Format: key_field_prompt($type)

Module: database

Return: Key field number, 0 for Esc, see below

Presents a menu that prompts a user for a key field based on the current view. The $type parameter specifies whether Special_sort or Physical are added to the list:

k - Only key fields are presented.

f - Only Physical is added.

o - Both Special_sort and Physical are added.

Possible return values:

0 if Esc was pressed by the user.

The number of the key field, whose value can be used as an array index in $$keys[], $$key_types[], and $$key_alias[]. The text “Special_sort”. The text “Physical”.

Note that, because this function presents an interface to the user, your program will be halted.

KRUNCH_SCREEN

Format: krunch_screen($screen, $path)

Module: database

Return: 0

Allows an end user to krunch one or all data files attached to the specified view. This is the programming interface to the Krunch command. When run, the user will be given suitable warnings.

```
krunch_screen( "depart", $$sysfiles )
```

Note that, because this function presents an interface to the user, your program will be halted.

LOADDATAVIEW

Format: loaddataview($menu, $load/unload, $mode)
Module: database
Return: numeric - 0=unsuccessful, 1=successful

Loads or unloads all data files and views associated with the specified menu name. The load/unload parameter can be set to “load” or “unload”.

The mode parameter is left blank under most circumstances. When it is used, the first three characters specify the type of special action to perform. Additional characters may be required, as outlined below.

If the left three characters are “DDL”, the system will read the *Database to Database* link definition specified by the remaining characters of the third parameter (e.g., “DDLmyddldef”).

If the left three characters are “PTH”, the main data view will be loaded from the path specified by the remaining characters (e.g., “PTHC:\arch”).

If the left three characters are “ALL”, the main view and all associated views will be loaded from the specified path.

```
IF loaddataview( "oe", "load", "")
    process data view...
ELSE
    error handling...
END IF
```

LOCK_REC

Format: lock_rec($message, #must_lock)
Module: database
Return: number - 0=unsuccessful, 1=successful

Locks a record and retries the lock if not successful. The first parameter is a text string that is displayed when the lock fails. Typically this message should communicate the name and path of the data file that is being locked. The RAD App will also print the physical record number and allow the user to view the data file being locked.

The second parameter must be 1 or 0 (true or false) and indicates whether the record must be locked. If it is 1, the function will not terminate until the record is locked. If the second parameter is false and the maximum number of retries is reached or the user presses escape, the function will abort and return 0.

NOTE: The LOCK_REC functionality is modified by two public variables which are set in the angoss.cfg file. The ##lck_rt variable indicates the number of seconds between lock retries and the ##lck_nr indicates the number of retries that will occur before giving up - note that this only has an effect when the #must_lock second parameter is 0. The product of ##lck_rt and ##lck_nr is the approximate number of seconds LOCK_REC will wait before giving up.

```
#lock = lock_rec("Locking "|get_path("inven")|"inven.db.",0)
IF #lock
    [on hand] = [on hand] + #qty
    WRITE-RECORD
ELSE
    rec_message( "ERROR: lock failed" )
    rec_message( [part #]&"not updated by"&str(#qty) )
END IF
```

MENU

Format: menu($mode)
Module: all
Return: 0 (or causes program to terminate)

This function accesses the program which drives every RAD App. It is responsible for managing the menu structure, and interpreting user input. It may be called in either of two modes, specified by the $mode parameter. If called with mode set to “normal”, the calling program will terminate, and the MENU() function will take over indefinitely. If mode is set to “dochoice”, a single action will be processed and control is then passed back to the calling program *providing a module change was not performed or an OSA script is not run*.

To invoke in “normal” mode, the READ_HEADER() function must first be called in order to set the public variables appropriately and open the menu file. All stack variables must be set properly (see Appendix B). When called, the MENU() function will cause the calling program to terminate and the RAD App menu system will be made operational.

To invoke in “dochoice” mode, the public variables $$act, $$obj, $$fct (action, object and function) must be set to specify the desired task. When invoked in this mode, only this one task will be performed.

```
$$act = "pcr"  'Print Current Record
$$obj = ""
$$fct = ""
menu( "dochoice" )
```

MENU_CALL

Format: menu_call($menu)
Module: all
Return: 0

Invokes the application menu specified by the $menu parameter. The menu will operate as usual. Any choice which appears on the menu may be selected, including those which lead to other menus. This function does not terminate until the user exits the called menu.

In order to guarantee that control will be returned correctly to the calling program, you must disable module changes by setting the public variable ##dsmch to 1 first. If ##dsmch is 0 and a module change takes place, the calling program will be terminated. Be certain to reset ##dsmch to 0.

```
##dsmch = 1
menu_call( "oe" )
##dsmch=0
```

MENU_TITLE

Format: menu_title($menu)
Module: all
Return: text - name of menu or error text

Returns the title of the menu specified by the $menu parameter. The title is returned in the default language. If the menu shortname does not exist, the message “MENU DOES NOT EXIST” is returned.

The following example uses the shortname array to specify the menu. This will return the title of the current menu.

```
menu_title( $$sn[##lev] )
```

MODULE_CALL

Format: module_call(#module, $project_file)
Module: all
Return: numeric

This function, along with the MODULE_RETURN function, is used to call a project in another SmartWare module when running a RAD application. They do all the work necessary to save and restore the current application position.

The parameters are:

#module - 1=SS, 2=WP, 3=DB, 4=CM, (-1=special)

$project - text string which is path and filename to execute when at the destination module (required).

If #module = -1 then the caller is to do the actual module change. (e.g., using the send command.) If -1 is used, the caller must execute the project $$advr|"ppmodch.rf3", which in turn executes the project specified by $project_file.

Return values are:

0 = Fail

1 = If #module=-1 and successful.

Doesn't return if 1<=#module<=4 and successful

Related Function: MODULE_RETURN

MODULE_RETURN

Format: module_return()

Module: all

Return: nothing

The MODULE_RETURN function is only used after MODULE_CALL. It requires no parameters. When called, it returns to running the application at the point that MODULE_CALL was issued.

Related Function: MODULE_CALL

ONE_RECORD

Format: one_record($index, $view, #record)

Module: database - Uses: file handle ##wrkhnd

Return: 1

Creates an index containing a single record and orders the file to that index. Subsequent print-outs, mail merges or send to spreadsheet actions will involve only this record.

The $index parameter contains the filename of the index to create. The $view parameter specifies which view (of those currently loaded) to act upon. If you specify an invalid view, the currently active view is used. The #record parameter specifies the physical record number of the record to select.

The function always returns 1 however, if the index file cannot be created or the record number exceeds the actual number of physical records, a SmartWare error will occur.

```
one_record( "temp", "invoice", PRECORD )
PRINT REPORT EXECUTE "inv" PRINTER DETAIL START * END * COPIES 1
reorder()
TOOLS FILE ERASE $$workdir|"temp.idx"
```

OPEN_FILE

Format: open_file($path, $file, #mode)
Module: All
Return: numeric - file handle, 0 or negative value if error occurs

Opens the specified file and returns a file handle. Used as a replacement for SmartWare's FOPEN command. The chief advantage here is that you don't have to know the next free file handle. When you're finished with the file, you must close the file with FCLOSE just as you normally would.

The file is specified by the $path and $file parameters. Note that the path parameter can be set to null ("") and added to the file name instead, however, if the path parameter is used, it must contain a directory separator at the end. The #mode parameter specifies whether the file should be opened in read only mode, share mode, etc. You can pass the mode parameter as a named constant (rw_mode, ro_mode, ro_share, rw_share) or as a string ("rw_mode", "ro_mode", "ro_share", "rw_share").

Briefly, the differences in the modes are:

RW_MODE - Read and Write abilities, locks file. Will create file if it does not exist.

RO_MODE - Read Only, locks the file. Error if file does not exist.

RO_SHARE - Read Only, allows simultaneous access. Error if file does not exist.

RW_SHARE - Read and Write abilities, allows simultaneous access. Creates file if it does not exist.

The returned file handle allows you to write programs that will not conflict with the RAD App or any other currently running programs.

If the function runs out of user file handles (maximum of 20 allowed) it returns 0. If an error occurs when opening the file, it returns the error number as a negative value, so that the error code can be determined by an absolute value (ABS function) of the result. No error message is shown to the end user.

```
LOCAL #hnd
#hnd = open_file( $$workdir, "temp.txt", RW_MODE )
IF #hnd=0
    MESSAGE "No more file handles. Press a key..."
    RETURN 0
ELSEIF #hnd<=0
```

```
    MESSAGE ERRORTEXT( abs(#hnd) )|". Press a key..."
    RETURN 0
END IF

FSEEK #hnd 0
FWRITE #hnd FROM "Hello There!"
FCLOSE #hnd
RETURN 1
```

OPEN_LOG

Format: open_log()
Module: all
Return: numeric - 0=unsuccessful, 1=successful

Opens the user activity log for the current user ($$usrid) and seeks to the end of the file in preparation for more entries. The first time called, $$workdir is set.

ORDER_COMMAND

Format: order_command($task, $wor)
Module: database
Return: 0

This presents the ORDER COMMAND interface on the current view. The $task argument specifies the number of the key field listed in the screen information - if it is passed as a null string, the user will be asked for the key or sort/order type. The $wor argument specifies the range (assuming the user selected a key or you passed a key number for $task). Valid $wor arguments:

“wf” - whole file

“oo” - one only

“r” - range

If the $wor is passed as a null string, the user will be asked for the range. If “wf” is specified, no further interaction is required. Thus, ORDER_COMMAND will not be interactive if a key number and whole file are passed. For example:

```
order_command( 1, "wf" )
```

The return value is always 0.

Note that, because this function may present an interface to the user, your program can be halted.

OS_APL

Format: os_apl($script)

Module: all

Return: 0 (or causes program to terminate)

Runs an operating system application such as a third-party software package. The $script parameter specifies the batch file. Its location must be in the system directory ($$syspath) with an extension of .OSA. When the application is terminated, the user is returned to the menu last used prior to execution of this function.

```
os_apl( "cadcam" )
```

This is only recommended for use in DOS.

PASSWORD

Format: password($password)

Module: all

Return: numeric - correct password (0=incorrect, 1=correct)

Displays the “Enter the password:” prompt. Any characters typed will be displayed as asterisks (*) on the screen. The user will have three chances to enter the correct password (specified by the parameter). If the correct password is entered, a value of 1 is returned, otherwise 0 is returned.

```
IF password( "ABC" )
    'allow access...
ELSE
    MESSAGE "Access denied."
END IF
```

POP

Format: pop()
Module: all
Return: 0

Unloads all files related to the current menu and sets up the files and variables required for the previous menu in the hierarchy. The ##lev level indicator is decremented by one. This function will not terminate the current program unless it causes a module change. In that case, the program terminates, changes modules, and returns control to the RAD App (i.e., the MENU function in “normal” mode). However, if the variable ##dsmch is set to 1 (this disallows a module change), the POP function and module change will not occur and the current program will continue from the following line.

Related function: PUSH

PRINT_DOC

Format: print_doc($file, $en_dr, $copies, $startpage, $endpage, $dest)
Module: word processor
Return: 0 if escaped, 1 if print command executed

Prints a word processor document. The following table shows the acceptable values for the parameters.

$file	Path and file name of document to print - "[]" indicates the current document.
#en_dr	Print mode: "E" prints enhanced mode, "D" prints draft mode, "A" prompts user for mode.
$copies	Number of copies to print: "A" prompts user for number of copies, null for 1 copy.
$startpage	Start print at page number: "A" prompts user for starting page, null for first page.
$endpage	End print at page number: "A" prompt user for end page, null for end of document.
$dest	Path and file name to print to, "A" prompts user for destination, "P" prints to the printer.

The following call prints the current document after confirming the print mode with the user. It will print 1 copy from the first page through to the last page.

```
print_doc( "[]", "a", "", "", "", "p" )
```

PRINT_T_FLD

Format: print_t_fld(#row, #col, #fg, #bg, $text)
Module: database
Return: text - next row and column

Print the contents of a scrollable text field on the screen. The upper left-hand corner is defined by the row and column parameters. The foreground and background parameters specify the colors that are to be used. The last parameter is the text to print. Database field references may be used. Tilde characters (~) are used as line feeds.

The row and column of the last screen position used by the print are returned in the form of a string: row followed by column and separated by a space (e.g., "10 42").

The following code fragment prints the contents of the [Question] field at position (5,10) on the screen, with a black (0) on white (15) color scheme. The returned position is then used as the starting position for the user's answer.

```
$rc = print_t_fld( 5, 10, 0, 15, [Question] )
#row = VALUE( GROUP($rc,1) )
#col = VALUE( GROUP($rc,2) )
SCREEN INPUT #row #col 0 15 8 $answer
```

PUSH

Format: push($menu)
Module: all
Return: numeric - success? (0=unsuccessful, 1=successful)

Sets the variables and loads the files required for the menu specified by the $menu (short name) parameter. It increments the ##lev stack level variable by one. If a module change is required by the new menu, the calling program will be terminated and control returned to the RAD App.

Related function: POP()

READ_CHOICE

Format: read_choice(#choice)
Module: all
Return: 0

Sets public variables according to the information for the specified choice on the current menu. The menu file must have been read with READ_HEADER(). The #choice number represents the sequential node number in the menu file. Note that, is READ_HEADER was called in RO mode, various transformations take place, such as deleting choices not required for the current user.

The following table lists the variables that are set by this function:

$$desc	Description which appears on the menu
$$act	Action
$$obj	Object
$$fct	Function/parameter
$$ah	Autohelp line
$$hppt	Help file pointer (".AHP")
$$pw	Password, if any

The $$desc and $$ah variables will be appropriately translated if an alternative language is active. The $$pw variable is determined by the current user mode.

```
read_header( "oe" )
FOR #ct=1 to ##numch
    read_choice( #ct )
    SCREEN PRINT #ct 1 15 0 $$desc
END FOR
```

READ_HEADER

Format: read_header($menu)
Module: all
Returns: numeric - success? (0=unsuccessful, 1=successful)

Opens the specified menu file in read only mode and sets public variables for the menu.

The following public variables are set:

$$sn[##lev]	Current menu short name indicator
$$typ[##lev]	Current menu type indicator
$$mttl	Menu title
$$clttl	Command list title
$$hphp	Overview help file name
##numch	Number of choices on current menu
##top	Top of menu (row)
##left	Left position of menu color
##width	Width of menu
##height	Height of menu
##comfg	Text foreground color
##combg	Text background color
##grafg	Graphics color
##audon	Record selections in Audit File
##mod[##lev]	Current resident module locator

Menu color variables are also set - refer to the RAD Colors section in Chapter 7.

RELOAD

Format: reload($files)
Module: spreadsheet, word processor
Return: 0

Loads all files passed in the $files parameter. This string accepts full file names with each file separated by a space (e.g., "\test\files\file1.ws \test\files\file2.ws"). This string can be obtained through SmartWare's CURRFILES() function prior to unloading files. Note that reload function detects files which are already loaded and skips all such files.

```
$files = CURRFILES(2)
FILE UNLOAD ALL
...other operations
reload( $files )
```

REORDER

Format: reorder()
Module: database
Return: numeric - success? (0=unsuccessful, 1=successful)

Reorders records in the current data view, placing the records in the order specified by the appropriate $$stat[] array entry. Each entry in this array contains the order (i.e., the key name, index name or "physical") and the record number. The stat array should be modified by the UP_DB_STAT function.

```
'SAVE RECORD #
#lev = active_lev( ##lev )
up_db_stat( #lev, record, "", "", "") 'update record #

'ORDER FILE
ORDER CHANGE KEY [name]
PRINT REPORT EXECUTE ...

'RETURN TO ORIGINAL STATUS
reorder()
```

RESTORE_ALLWIN

Format: restore_allwin($file)
Module: database
Return: numeric - success? (0=unsuccessful, 1=successful)

Reload all files that were closed and saved by the CLOSE_ALLWIN() function. The $file parameter specifies the text file that stored open file names at the time of the close call.

```
close_allwin( $$workdir|"save.win" )
...other operations
restore_allwin( $$workdir|"save.win")
```

Related function: CLOSE_ALLWIN()

RP_EXEC

Format: rp_exec($vw, $rep, $in_scrn, $qry, $prog, $order, $params, #no_interrupt)
Module: database
Return: 0

Executes a previously defined Developer report. Each of the objects (or null strings) must be passed. Execution parameters ($params) are passed as a set of seven numbers (the order is based on the report's execution dialog box).

1	Destination	(1) ask user, (2) disk, (3) printer, (4) screen, (5) text screen
2	Type	(1) ask user, (2) details, (3) totals only
3	Pages	(1) ask user, (2) complete report
4	Copies	(1) ask user, (2) one
5	Allow One	(1) ask user, (2) yes, (3) no
6	Records	(1) whole file, (2) user index
7	Audit	(1) yes, (2) no

If $param is a null string, the user is prompted for all options with the report operating on the user index and no audit. Thus, the value of this parameter defaults to "1111122".

Warning: If a program that executes RP_EXEC is called in a job stream, its input screen can halt the stream. In this case, call the report directly from the job stream or do not use an input screen.

RUN_CRYSTAL

Format: run_crystal($wintitle, $report, $destination)

Module: all

Return: numeric - 1 (report launched), 0 (error or aborted)

Executes a Crystal report - available in Windows only. The $wintitle argument indicates the name of the window when the report is in Print Preview mode. The $report argument specifies the Crystal report. The $destination argument determines if the report is printed, previewed, or left to the user to decide its destination. - values can be "P", "S", and "A" for Printer, Screen, and Ask User.

You should set the $$crw_path variable in the angoss.cfg file to point to the directory where Crystal Reports is installed. If it is not set, RAD looks in the C:\CRW directory and the D:\CRW directory.

Note: Crystal Reports must be installed in order to create or modify these reports. However, the runtime files installed with SmartWare will allow users to print Crystal reports without requiring the actual program.

Reports (.rpt files) should be saved in the home directory of the application. This is important when you first launch Crystal Reports, because it may not be in the correct directory.

Note that the order or index of the current view does not affect the printing of a Crystal report nor does Crystal Reports have access to custom views. This is because the connection between SmartWare and Crystal Reports is through an ODBC driver. One way of working around these limitations is to create a temporary database (using the DATA QUERY command for example) which reflects the current order and index and includes any calculations you want to use. Then make your Crystal report access the temporary table.

RUN_QUERY_STR

Format: run_query_str($name, $query, #mess_hnd)
Module: database
Return: numeric - number of records selected or error, see below

This function runs a query specified by the $query string variable. It avoids the need for separate query files and makes code more readable by embedding query statements in project file.

The parameters are:

$name - File name of index that is generated (do not supply path or extension) path will be $$workdir, ext=.idx.

$query - String containing the logical expression of query. Use \n for new line. Use \' for double quotes (see example).

#mess_hnd - File handle of an already open text file for messages. Use 0 for no message file. Use number to write the query name and expression to a file when executed for tracing purposes.

Return values:

Successful - the number of records selected

Error - 0 means error was caused internally by the RUN_QUERY_STR function, otherwise a negative value indicates a SmartWare error number (multiply this by -1 to get real number).

Examples:

```
run_query("dept","[Department]=\'A\'",0)
$dept = "A"
run_query("dept","[Department=\'"|$dept|"\'",0)
```

SCROLL

Format: scroll(#speed)
Module: all
Return: 0

Causes the records, cells or lines of the current data view, spreadsheet or document to scroll up or down on the screen. The #speed parameter must be an integer from -10 to 10, 0 not included. The absolute value of this integer indicates the speed of the scroll; its sign indicates the direction - negative values cause upward scrolling, and vice-versa.

While scrolling, the user may change the scroll direction by pressing the up/down arrow keys and the speed by pressing the appropriate number key.

```
scroll( 8 )
```

SORT_STRING

Format: SORT_STRING($list)
Module: all
Return: text - sorted list of words

Alphabetically sorts the list of words contained in the argument. Words are delimited by spaces.

SPLIT_PFE

Format: SPLIT_PFE($fullpath)
Module: all
Return: text - argument string broken into words

Separates the path, file name and extension in the argument string from one another. Each element is stored in the returned string separated by spaces. SmartWare's GROUP function may be used to separate these elements from the returned string.

```
$pfe = split_pfe( "c:\gl\files\coa.db" )
IF GROUP( $pfe, 1 ) = "c:\gl\files\"
    'Path exists
END IF
```

```
IF GROUP( $pfe, 2 ) = "coa"
    'File exists
END IF
IF GROUP( $pfe, 3 ) = "db"
    'extension exists
END IF
```

SUSPEND_APP

Format: SUSPEND_APP($message, #pulldowns)

Module: all

Return: nothing

Suspends the RAD App and provides end user access to SmartWare. When the user exits SmartWare, the program is resumed. The message is a text string that appears on the top right of the SmartWare window if the #pulldowns argument is set to 0 for standard ring menus. In this case, the user should press F8 to return to the RAD App (typically the message informs the user to press F8). If #pulldowns is set to 1, the message does not appear and the user must select File > Exit to return to the RAD App.

Note: If the $message is too long (depending on window size) the error "*Invalid line/ column parameters*" will occur.

TEMP_EXIT

Format: TEMP_EXIT()

Module: all

Return: none - causes termination of program

Saves the current status of the application, sets a return flag, and exits the RAD App. The start up batch file will search the user's work directory ($$workdir) for a batch file named osapl.bat. If found, the file is executed. When execution of the script is complete, the RAD App is restarted and the user is returned to the original location.

This function is intended for DOS users only.

TOOLS_FILE

Format: tools_file($command, $file1, $file2, #interrupt)
Module: All
Return: 1

A convenient version of SmartWare's TOOLS FILE... commands.

The $command parameter must be "copy", "delete", "move", "rename", or "print". The $file1 parameter specifies a full file name. Wild card characters are allowed. The $file2 parameter is required for the copy, rename and move variants, specifying the destination file name.

If the #interupt parameter is set to 0, an end user will NOT be able to terminate the operation with an Esc key. Normally the Esc key will interrupt a TOOLS FILE operation and present the end user with the prompt: Abort (Y/N). If you have already used the REPLY command to affect prompt 12, or want to allow the Abort (Y/N) prompt then set this parameter to 1.

TOOLS_FILE turns error messages to the end user off so there's no need to check for the existence of a file before erasing it. Use CLEARERROR and LERROR to check for any errors.

UP_DB_STAT

Format: up_db_stat(#lev, #record, $ord_type, $ord_name, $ddl)
Module: database
Return: 0

Modifies the $$stat[] variable (status stack variable). The $$stat[] array contains information about data views or spreadsheets that are active. In the database it contains: the current record number, the order of the file, and any active DDL's (database to database link definitions). This information is used when changing menus. The REORDER() function, which resets a file's order and record number, inspects the $$stat[] array. You will need to use this function if you change the order of a data file and want the RAD App to recognize the new order or record number.

The #lev parameter is used to identify which active data view you are specifying new information for. If you are changing the current data view, pass the ##lev public variable.

The #record parameter indicates the new record number. The $ord_type parameter must be set to “key”, “index” or “physical”. The $ord_name parameter is specifies the index name or the key field name if appropriate. Finally, the $ddl parameter specifies the database to database link definition that was used to reach the current data view. To skip the last three parameters, pass null strings.

```
ORDER CHANGE KEY [name]
up_db_stat( ##lev, record, "key", "name", "" )
```

WRITE_NAME

Format: write_name($str, #pos, #handle)
Module: database
Return: 1

Writes a view name to an index file. This allows you to create indices manually with SPL. It assumes the index is already opened in write mode with the specified file handle. To write a view name on an index, use a position of 4.

```
write_name( "inven", 4, ##wrkhnd )
```

See function WRITE_NUM for more information.

WRITE_NUM

Format: write_num(#number, #pos, #handle)
Module: database
Return: 1

Writes record numbers or record counts on an index file. This allows you to create indices manually with SPL. It assumes the index is open with the specified file handle. The following example creates an index with one record, the current record, in it.

```
FUNCTION one_record( $index, $file, #record )
    WHILE NEXTKEY <> 0
            INCHAR
    END WHILE

    IF FILE( $$workdir|$index|".idx" )
            TOOLS FILE ERASE $$workdir|$index|".idx"
    END IF
```

```
    'CREATE INDEX
    TOOLS FILE COPY $$advr|"blank.idx" TO $$workdir|$index|".idx"

    FOPEN $$workdir|$index|".idx" AS ##wrkhnd OPTIONS RW_MODE

    'WRITE RECORD COUNT
    write_num( 1, 0, ##wrkhnd )

    'WRITE NAME
    write_name( $file, 4, ##wrkhnd )

    'WRITE PHYS RECORD #
    write_num( #record, 64, ##wrkhnd )

    FCLOSE ##wrkhnd
    ORDER CHANGE INDEX $$workdir|$index
    RETURN 1

END FUNCTION
```

See also WRITE_NAME.

WRITE_ULOG

Format: write_ulog()

Modules: all

Return: 1

Records an entry in the user usage log. It uses the public variables $$usrid, $$sn[##lev], $$act, $$obj, and $$fct to determine the details of the entry. It assumes that the user's usage log is open with file handle ##lgfh.

```
write_ulog()
```

ZAP_DB

Format: zap_db($file, $path)
Module: database
Return: 0 if an error occurs, or 1 if no error occurs.

Completely erases all records in a data file. If the standard view is loaded, it is unloaded, purged and then reloaded. No other views containing the specified data file must be loaded.

Note: This gives no warnings to the end user.

Appendix B

Public Variables

This section describes the public variables used by RAD applications.

All RAD public variables conform to a simple naming convention. Those variables holding text data are preceded by two dollar signs ($$), while those containing numeric information are indicated by two pound signs (##). Numeric variables may double as boolean variables; a value of one (1) may correspond to True, and zero (0) to False. Such variables may be set in an SPL program by using the specifiers “True” and “False” (e.g., ##flag = True). Note, however, that actual variables in SPL make no distinction between numeric and text variables; these are merely arbitrary naming conventions to avoid conflicts with an application's public variables.

General-Purpose Variables

Many of the following variables are set in the angoss.cfg configuration file. Others are set by the Information Set values.

Variable	Description
$$adev	Path containing the Developer location.
$$advr	Path containing RAD Shell location.
$$archpath	Archive data path. Set by the information set.
$$botp	Prompt which appears at the bottom of box-style menus (varies by language). Set by the ANGOSS.CFG in default language or by the Language data file when using an alternative language.
$$crw_path	Points to the directory where Crystal Reports is installed. If it is not set, RAD looks in the C:\CRW directory and the D:\CRW directory.

Variable	Description
$$dev_name	This variable is shown on the copyright message that is displayed just before the first menu of the application is displayed. It can be used to indicate who created the application. To use it, declare it as a public variable in one of the database application libraries. Issue a LOCK-SYSTEM in the main portion of the program, and set it to the string you want to display.
$$dirsep	Specifies the character used to separate directories in path specifications ("\" or "/").
$$homedir	Main directory of the RAD App.
$$macpath	Macros data file and definitions path. Set by the Infoset.
$$merpath	Mail Merge data file and documents path. Set by the Infoset.
$$opsys	Indicates the current operating system. See also SPL's SYSVAR function.
$$procpath	Path of Procedures directory. Set by infoset.
$$quepath	Path of Query data file, definitions and indices. Set by the infoset.
$$reppath	Path of Custom Reports data file and definitions. Set by the infoset.
$$senpath	Path of Send data file and definitions. Set by the infoset.
$$sorpath	Path of Sort data file and definitions. Set by the infoset.
$$sysfiles $$sysfiles2 $$sysfiles3 $$sysfiles4 $$sysfiles5	These five variables contain path names of the default application data path(s). Set by infoset. New ones can be created.
$$sysid	Short text-string which serves as identification of the system. It usually correlates with the name of the main directory of the application and is used when auditing batch update and process logs. It should not be longer than eight characters. Set by the ANGOSS.CFG.
$$syspath	Path of files generated by the system.

Variable	Description
$$topp	Contains the prompt which appears at the top of box-style menus (varies by language). Set by the ANGOSS.CFG file when in the default language, otherwise set by the Language data file.
$$tpltpath	Path of Template files. Set by the infoset.
$_ttl	Application title. Set by the System Defaults view when running in default language, otherwise set by the Language data file.
##activeimp	Flag indicating whether or not import is active (0=no, 1=yes). If it is true, then the applications menus have been imported into the menu data files. The danger of updating menu information in two places and subsequently overwriting exists.
##audlg	Flag indicating whether or not to audit logins (0=no, 1=yes).
##chlog	Flag indicating whether or not to ask the developer to enter Changes Log entries each time the Developer System is invoked. (1=always, 0=never, -1=ask user). Set in the ANGOSS.CFG file.
##cmplt	Flag indicating whether the system is complete, or whether it is still under development. If it is true, the Developer is disabled. Set by the ANGOSS.CFG file.
##curblock	Current spreadsheet edit area block.
##dks	Key stroke to activate the Developer (numeric key code). Set by the ANGOSS.CFG file.
##lgfh	Activity log file handle. Used also to determine whether the user has logged on more than once concurrently and to write activity log. Set by the ANGOSS.CFG file.
##lmks	Key stroke to invoke language maintenance. Set by the ANGOSS.CFG file.
##mmod	Mode in which menus are displayed. (1=box style, 2=look ahead, 3=cascading, 4=voice, 5=custom, 6=graphic). Set in ANGOSS.CFG.
##numlang	Number of languages defined in the system.
##numum	Number of user modes defined in the system.

Variable	Description
##pidx	Indicates in which directory indices created via Query and Sort processes are stored (1=personal directory: $$workdir, 0=central directory: $$quepath or $$sorpath). Set in the ANGOSS.CFG file.
##prhnd	Procedure file handle. Used to open procedural instructions. Set in the ANGOSS.CFG file.
##umks	Key stroke to initiate User Mode Maintenance system. Set in the ANGOSS.CFG file.
##wrkhnd ##wrkhnd2 ##wrkhnd3 ##wrkhnd4	These four variables contain file handle numbers. Set in the ANGOSS.CFG file, for general use by SPL programs.

Login Choice Variables

All variables listed here are determined by the user ID. Most values are set directly from fields on the user profile.

Variable	Description
$$infoset	Indicates user's information set.
$$lang	Short name of the alternate language for the current user mode.
$$lpath	Language indicator: ""=default language, otherwise indicates path of language files.
$$persdir	Path of current user's Personal Directory. More than one user may be assigned the same Personal Directory. Used during send to spreadsheet or printing to disk.
$$smen	Short name of the start up menu.
$$um	Description of current user mode. A Null string indicates Developer status.
$$usrid	Current User ID code.

Variable	Description
$$workdir	Path of general-purpose directory which is reserved for use by the current user only. Generally used to create temporary files etc.
##admin	Flag indicating whether or not the current user can access the User Mode Maintenance system.
##ah	Flag indicating whether or not autohelp is enabled.
##aphlp	Status flag - user access to application help: 0=Read Only, 1=Author, 2=Disabled.
##defin	Flag indicating the ability of the current user to modify definitions (e.g., Query, Sort). 0=do not allow modification, 1=allow modification.
##doslog	Flag indicating whether the User ID was specified from the OS command line (##doslog=1) or from the prompt supplied by the RAD App (##doslog=0). In the latter scenario, OS Application calls will not function properly.
##lang	Flag indicating whether or not the Language Maintenance system is available to the current user.
##prhlp	Flag indicating whether or not Procedural Help is available to the current user. (0=read only, 1=author).
##tchlp	Flag indicating whether or not Technical Help is available to the current user. (0=read only, 1=author).

Current Choice Variables

In the following descriptions, the term “current choice” refers to the menu item whose information has been most-recently read. Interchangeably, it refers to the

choice which is highlighted on the current menu; in this case, these variables are being set and accessed by the RAD App program itself.

Variable	Description
$$act	Specifies the Action of the current choice. Possible Actions are described in Appendix D of this manual.
$$ah	Contains the autohelp text associated with the current choice.
$$desc	Contains the description of the current choice; this is the text which appears highlighted on the menu.
$$fct	Contains text which can be used by SPL programs to determine the function of the current choice.
$$hphp	File name of Overview help text file for the current menu.
$$hppt	File name of help text file for the current choice.
$$obj	Contains data on which the Action of the current choice operates. For example, if the Action calls a menu, the Object specifies which menu to invoke.

Current Menu Variables

The variables listed here pertain to the menu (navigation or view) that is currently active.

Variable	Description
$$clttl	Current view menu title in character mode (command list).
$$lprog	Name of load/unload override program (optional).
$$mttl	Contains the title which is displayed at the top of the current menu.
##audon	Flag specifying whether or not to audit the usage of menu items by recording each occurrence in the audit log. If ##audon=1, menu selections will be audited.
##barbg	Highlighted bar background color.

Variable	Description
##barfg	Highlighted text foreground color.
##combg	Background color of items which are not highlighted. For writing at scrheight-3 or scrheight-2.
##comfg	Foreground color of unhighlighted items.
##curch	Number of the current choice as it appears on the menu.
##grafg	Graphics border color.

Menu Stack Variables

All of the following arrays are dimensioned to the size indicated by ##maxlv which determines the maximum depth of menus. These arrays act as a stack when: an item on one menu leads to another menu, one more menu are pushed onto the stack, and when a user returns from a menu, that item is popped from the stack.

Variable	Description
$$actm[1]	Name of the menu which caused the current data view, SS view or document view to be loaded.
$$mac[1]	The macro file that is loaded at that level.
$$sn[1]	Menu short names.
$$stat[1]	Status: contains various data regarding the current data view, SS view or document view. See below for more details
$$typ[1]	Menu type: navigation menu (menu), command list (comlist), data view (dataview), Spreadsheet view (ssview), document view (doc-view).
##cur[1]	Item number selected by user.
##lev	Current level in stack (serves as index into the stack arrays).
##maxlv	Maximum number of levels in the menu hierarchy and therefore the maximum number of elements in the stack arrays.
##mod[1]	Module in which menu resides. (1=SS, 2=WP, 3=DM, 4=COM)

Status Array Layout

$$stat[n]

This section describes the layout of the $$stat stack array. Note that the $$stat array is only filled for menus that load a data view, document or spreadsheet. For example, if a main menu choice loads the customer file and a choice from that customer view displays the print menu, the following stack will be built:

1. Main Menu

2. Customer File

3. Print Menu

Only the customer file level fills the $$stat array. There is no active data file for level one and level three contains the same file status information as level two. The ACTIVE_LEV() function returns the level that stores the status information. In the above example ACTIVE_LEV(3) would return 2.

Status Information for Each Module

Database

Position	Description
1 - 2	Not used
3 - 14	Record number
15 - 22	Order Type: “Physical”, “Index” or “Key”
23 - 102	Order Name (Key name or Index name)
103 - 182	DDL Definition Path and File Name if DDL is active

Spreadsheet

Position	Description
1 - 8	DSL Name if linked to the database
9 - 16	File name

Position	Description
17 - 66	Path
67 - 72	Block number
73 - 78	Full DDL File Name if a DDL is active
79 - 84	Column number

Word Processor

Position	Description
1 - 8	DSL Name if linked to the database
9 - 16	File name
17 - 66	Path

Current View Variables

Current active data view, spreadsheet or word processor variables.

Variable	Description
$$areas	String specification of all edit areas for the current spreadsheet. This string is in the format accepted by the DECODE_EDA() and EDIT_AREA() functions.
$$brfld	String specification of the browse fields associated with the current data view. Each field name is separated by semicolons.
$$file	Current view file name: custom view, worksheet, or document.
$$keys[15]	Key fields names.
$$key_types[15]	Case Conversion types.
$$key_alias[15]	Key alias.
$$key_prompt	Key prompt type: 0 = auto, 1 = bar, 2 = Pop Up

Variable	Description
##num_keys	Number of keys.
$$path	Path of the current view spreadsheet or document.
$$updel_expr	Logical expression, if evaluated as true, UPDATE and DELETE are disabled, blank value to nullifies effect.
##border	Border On = 1, Border Off = 0. Note: scroll bars do not appear with border off.
##post	If = 1, there is a $$updel_expr which should be checked, if = 0, no update/delete checking is done.

Miscellaneous Status Variables

Variable	Description
$$answer1 $$answer2 $$answer3	These three variables contain the archive answers. They are set when an archive job is run.
$$default1 $$default2 $$default3	These three variables contain the archive defaults. They are set when an archive job is run.
$$clm	This is a versatile variable, used to transfer information between SPL programs. When one SPL program invokes another, it may set this variable to a text string which will be recognized by the destination program. In this way, a program may be called under a number of different circumstances, and the program can use this variable to determine which function to perform. Warning: Do not rely on the value of this variable remaining constant for any length of time. It is used by many functions and programs and thus its value changes often.
$$cpsid	Current Procedure Step help text ID. This variable contains the name of the help text file associated with the step being currently executed in the current procedure or job stream.

Variable	Description
$$proc	Path of directory containing the help files associated with the currently-running procedure or job stream.
$sure	This variable contains a user response to continue an audited program. Audited programs always display an input screen prior to commencing. A choice must be supplied on the screen to allow users to either continue or abort program execution. The results of this choice MUST be placed in the $sure variable. Note: the RAD App creates the $sure input automatically.
##br	Browse mode flag: 0=off, 1=on.
##cpstep	Current procedure step number (if ##cpstep=0, no procedure is currently running).
##dsmch	Disable Module Changes: flag indicating whether switching to another module should be allowed from the menu system. If ##dsmch=1 and a user selects an item from a menu that causes a new module to be invoked, a message stating that module changes are disabled will be displayed and the menu item will not function.
##pushn	Flag indicating whether motion is up the menu structure hierarchy (##pushn=1) or down (##pushn=0).

Appendix C

System Files

File Name Extensions

RAD Apps accesses files of various types. Some of these files are maintained by the Developer and the RAD Shell, while others are created and maintained by SmartWare.

The following table describes each type of file, the file name extension which identifies it and, its usage in a RAD system.

For more information on SmartWare extensions refer to Appendix A of the Software System manual.

Extension	File Type	Description/Uses
.AHP	App Help	Help text regarding a menu (overview) or menu item (specific). Linked to a menu or item via the Menu File.
.BTX .BDC .BWS etc.	Backup	These, and often other files whose extensions begin with “B” (except .BAT and .BMP) are Backup files. The last two characters indicate what type of file is being backed up. In this case, it is a text file etc. backup. Backup files can be safely erased as long as the original file is known to be intact.
.BMP	Bitmap	Graphics files

Extension	File Type	Description/Uses
.CFG	Configuration	ANGOSS.CFG existing in the application home directory and may be modified in the text editor. The APPLICTN.CFG exists in the system directory. This file is automatically maintained and should not be modified.
.CFR	Configuration	Compiled (rf3) angoss.cfr.
.CTD	Calc Total	Definition for a Calculate Total operation.
.DB	Database	This type of file holds the actual data contained in a single data file.
.DBQ	Query	Backup Definition of a Query operation.
.DFQ	Query	Definition of a Query operation, which specifies the criteria by which a subset of database records are selected.
.DFR	Report	Data File Definition specifying the details of a database report printout.
.DMP	Dump	This temporary file is created by a RAD App whenever a switch between modules is made. It serves to pass environment information to the program running in the new module.
.DOC	Document	Word processor document in SmartWare format.
.DSL	DSL	Definition of how information is transferred between a data view and a spreadsheet view (either directions).
.DWL	DWL	Definition of how information is transferred to a word processor view (document) from a data view.
.FSI	FSI	This is a project file that contains information pertaining to an information set.

Extension	File Type	Description/Uses
.GDB .GDH .GDS .GDE .GDT .GDC	Graphics	Graph definition files.
.GEN	Documentation	Temporary file created by the documentation generator.
.HLP	Help	Help text file similar to Application Help File but linked to a menu or item via the Autohelp File.
.IDX	Index File	A list of physical database record numbers in a predefined order.
.IS?	Input Screen	Input screen definition.
.ISV	Report	This file activates public variables used in a RAD Report input screens. It is a compiled project file located in the home directory.
.KEY	Key	Contains sorting information displaying records in a predefined order. The order is based on special fields called keys.
.LOG	Log	This text file keeps track of a particular user's activities Log within the system (for audited menus only). When open, it indicates that the user is logged on and so, this file also detects multiple log-on attempts by the same User ID. It is stored in the system/log directory.
.LST	List	These text files are automatically created by a RAD App and should not be tampered with.
.MAC	Macros	This type of file contains the definitions for a set of remapped key strokes.
.MNU	Menu	Stores a RAD App menu information. One for each menu in a system.

Extension	File Type	Description/Uses
.MRG	Merge	Definition of a Mail Merge operation. Contains information detailing how data is to be sent from a data view to a word processing document.
.OSA	OS App	A script file, which is copied to “OSAPL.BAT” in the user's work directory and then executed. Such files hold Operating System commands which cause a third-party software package to be invoked.
.PF?	SPL source	Text file containing a Smart Programming Language program.
.PHP	Help	Similar to .AHP and .HLP files, this type of file contains help text for use with a job stream or procedure.
.PIX	Index	Physical index file. Refer to the SmartWare Database manual.
.RCM	Commands	Contains specifications for commands which are to be added to a SmartWare menu. An example of this occurs during a SUSPEND action - Resume_Application appears on the Quit menu.
.RF?	SPL compiled	Binary file containing a compiled Smart Programming Language program.
.RPP	RAD Report	When a RAD report is executed by a procedure, this file is created to communicate to the procedure which report options should be used. It is a temporary file stored in the user's work directory.
.STK	Stack	A temporary file used by RAD Apps to store stack data.
.TMP	Temporary	A temporary file created by RAD Apps. These files may be deleted at any time except during the execution of the process which generates the file.

Extension	File Type	Description/Uses
.TXT	Text	A general-purpose text file.
.UDC	Dictionary	SmartWare custom dictionary file for use with the spell-checker. ANGOSS.UDC is a custom dictionary for use with RAD App documents.
.USR	Nodes	Node protection files containing serial numbers and information regarding the number of copies which can run concurrently.
.VW	Custom View	Definition file for a database custom view.
.VWS	Standard View	Definition file for a database standard view. Created by SmartWare when a data file is created.
.WS	Worksheet	Worksheet file (spreadsheet).

Input Screen Files

The SmartWare Input Screen editor presents easy-to-use commands for drawing lines and boxes, inserting text, creating menus, and adding alphanumeric input fields. However, you may want to use a text-editor to edit the layout directly. The following describes the Input Screen layout file:

First Column **Value**	First Column **Description**	**Remaining Columns**	Remaining Columns **Description**
1	clear screen	2/3	foreground color
		4/5	background color
2	text line	2/3	row coordinate
		4/5	column coordinate
		6/7	foreground color
		8/9	background color
		10>	actual text

First Column **Value**	First Column **Description**	**Remaining Columns**	Remaining Columns **Description**
3	input field	2/3	row coordinate
		4/5	column coordinate
		6/7	foreground color
		8/9	background color
		10/11	field length
		12>	variable name
4	line or box	2/5	top left coordinates
		6/9	bottom right coordinates
		10/11	foreground color
5	table (menu)	2/5	top left coordinates
		6/9	bottom right coordinates
		10/11	foreground color
		12/13	background color
		14>	variable and choices

Appendix D

Actions

The action codes described below are the codes contained in the $$act variable during the execution of an item from a RAD App menu. These actions appear on the Item Information Screen, accessed with the Developer's command sequence Modify [select menu item] Item_Information. They are also visible on the Menu Hierarchy Diagram, accessible from the Management and Utilities menu. The Object and Function paragraphs describe the values, if any, required for the $$obj and $$fct variables. These values are also visible on the Item_Information dialog box and the Hierarchy Diagram.

ASVIEW

Invokes an associated view, as indicated in the spreadsheet info, or in the DV_INFO system data file.

Object: Not used.

Function: Not used.

BROWSE

Toggle between one-record-per-screen and one-record-per-line viewing mode. Only available for use with data views.

Object: Not used.

Function: Not used.

CA

Change Application. This action causes a new RAD application to be invoked.

Object: Contains the home directory of the destination application.

Function: Not used.

CALC

Calculate totals of various fields on the current record according to a Totals Definition file.

Object: Name of totals definition file (.CTD)

Function: Not used.

CRW

Execute a Crystal Report.

Object: The report file name. Assumed to be in the home directory unless the path is hard coded.

Function: Destination of report: “P” for printer, “S” for screen, “A” for ask user.

DELETE

Delete the current record. Toggles on or off the DEL status of the current record. Only available for use with data views.

Object: Not used.

Function: Not used

DOCV

Invokes a document view. A word processing document will be loaded and a view menu.

Object: Menu short name.

Function: Name of the DWL (Database to Word Processor Link definition. Only used when a document view is called from a data view.

DV

Invoke a data view. A database custom view will be displayed and a view menu supplied.

Object: View name.

Function: Name of DDL definition. Only used when a data view is called from another view.

EDITAREA

Edit one or more specified areas on the current spreadsheet.

Object: Specification of areas to use. Format identical to that accepted by DECODE_EDA and EDIT_AREA functions.

Function: Not used.

ENTER

Move to a new, blank record and allow entry of data. Only available for use with data views.

Object: Not used.

Function: Not used.

FCT

Calls a function.

Object: The function name. The function must have one parameter.

Function: The parameter passed to the function.

FIND

Search for an item in the database. Only available for use with data views.

Object: Usually "", may be key field name of "seq:"

Function: Not used.

GRAPH

Prints a graph according to a predefined set of specifications.

Object: Name of graph definition.

Function: Contains two words (separated by a space). The first specifies the type of graph. The second specifies the destination: landscape_printer, portrait_printer, xy-plot, metafile, ask_user, preview.

KRUNCH

Remove all records with DEL status (see DELETE above). Only available for use with data views.

Object: Not used.

Function: Not used.

MACRO

Access the end user macros system, where a browse view of all available macro definitions are listed, from which the user may select one for editing or loading. Only available in Database Module.

Object: Not used.

Function: Not used.

M

Invoke a menu.

Object: Menu short name. This name will form a menu file name with extension .MNU.

Function: Not used.

MERGE

Perform a mail merge operation (end user tool). Data will be sent in a predefined format to the word processor. A browse view of all available Merge Definitions will be provided, from which the user may select one to edit or invoke.

Object: Not used.

Function: Not used.

ORDER

Change the order in which database records are displayed. A menu containing all available key fields, plus the Special_Sort and Physical commands, is provided. Only available for use with data views.

Object: Not used.

Function: Not used.

OS

Execute an OS-level command keeping SmartWare active.

Object: Contains command to be executed example, dir *.*.

Function: Continuation of command if longer than 50 characters.

OSAPL

Execute a batch file from the OS level. This file may invoke third party software; SmartWare and the RAD App are temporarily unloaded.

Object: Name of script file to run. Copies $$obj|".OSA" to "OSAPL.BAT" and then executes the latter.

Function: Not used.

PCR

Print the current record as it appears on the screen.

Object: Not used.

Function: Not used.

PROC

Invoke a procedure.

Object: If Null, the user is presented with a browse view of all available procedures. Otherwise, $$obj contains name of procedure.

Function: Not used.

QUERY

Access the end-user Query system. A browse view of available Query Definitions is provided where the user is free to choose one to edit or invoke.

Object: Not used.

Function: Not used.

REPGEN

Allows a user to access the custom report generator.

Object: Not used.

Function: Not used.

RP

Print a predefined report.

Object: Internal format string (No access).

Function: Default name of report components.

RT

Returns from a menu. If a document or spreadsheet is active, it is automatically saved.

Object: Not used.

Function: Not used.

RTAS

Return from a spreadsheet or document view and ask user to save the spreadsheet or document.

Object: Not used.

Function: Not used.

RTNS

Returns to previous menu without saving the current spreadsheet or document.

Object: Not used.

Function: Not used.

SENDSS

Perform a send to spreadsheet operation (end-user tool). A browse view of available Send to SS Definitions is provided from which the user may select one to edit or invoke.

Object: Not used.

Function: Not used.

SSR

Print a predefined Spreadsheet Report. Always prints to the printer (not disk or screen), one copy only.

Object: Name of report definition to execute.

Function: Not used.

SUSPEND

Suspend execution of the RAD App and allow direct access to the current SmartWare module. The RAD App is resumed when exit is selected.

Object: Not used.

Function: Not used.

SV

Invoke a spreadsheet view. A worksheet will be loaded and a view menu supplied.

Object: View name.

Function: Name of DSL definition. Only used if the spreadsheet is being called from a data view.

SYSDEF

Access System Defaults.

Object: If not Null, $$obj codes for the name of an SPL program to run in addition to the program which is generated automatically to set the public variables from these defaults. This program can be used to perform special functions with the system defaults.

Function: Not used.

TEXT

Display a text file.

Object: An expression that evaluates to the text file name. Precede this expression with a colon, so that any public path variables will not be prematurely decoded (e.g., :$$advr|"file").

Function: Edit modes: “normal” indicates that only developers will be given the option to edit the file when displayed; “edit_always” indicates that all users should be able to edit the file; “view_only” indicates that NO user will be given edit privileges when the file is displayed.

UPDATE

Data view update mode.

Object: Usually not used. Enter 1 to use update only one.

Function: Not used.

UPDATESS

Allows access to the SmartWare's spreadsheet module.

Object: Not used.

Function: Not used.

UPDATEWP

Allows access to the SmartWare's word processor module.

Object: Not used.

Function: Not used.

X

Execute an SPL program. When an executed program terminates, control is passed back to the RAD App.

The X<n> variation of this action, where n specifies the module that must be active, is only valid on No Change menus. For example, X3 indicates that the database must be loaded before execution.

Object: Program name.

Function: May contain a unique string which identifies the calling position. This provides multiple access points to one program.

XAUD

Execute an audited SPL program. Each execution is recorded in the error/audit log.

Object: Program name.

Function: See X action.

ZAP

Removes all records from a file deleted or not. Also requests whether related files should be zapped.

Object: File name.

Function: Not used.

Appendix E

Converting Applications

This section covers the required steps to convert an existing SPL based SmartWare application into a RAD application.

NOTE: SPL program based applications are fully supported: programs will run without modification. However, to take advantage of the features delivered by RAD applications, you must perform this conversion.

Before attempting the conversion, it is recommended that you try the Tutorial in Chapter 2 in order to familiarize yourself with RAD App concepts and terms.

Conceptual Overview

An SPL program based application can be converted easily. Existing data files, definitions and programs of the SPL program application can be used by copying them into the proper directories within the RAD application. Menus from the original application should be rebuilt to utilize the powerful menus and maintenance features.

After installation, the conversion can be done in three broad steps:

- Create the RAD application.
- Copy existing data files, definitions and programs from the original application to the RAD application.
- Rebuild the menu structure with the Developer.

Creating the RAD Application

RAD Apps are created through SmartWare's Remember Make-Application command series. Enter the application's directory and start up name into the dialog box, select the appropriate start up devices (icon, batch, script) and select OK.

Copying Files and Converting Menus

Copying Database Files

Normally, RAD application data files are stored in the files subdirectory below the home directory of the system. Copy all data files from your SmartWare application to this subdirectory. The file extensions that relate to data files are:

.vw - Custom View

.db - Database Information

.vws - Standard View

.key - Key file

.pix - Physical Index (Variable Length Files Only)

For example, if your SmartWare files exist in a directory named \ORDENT and the RAD application is in \OE, issue the following commands at the DOS prompt:

```
COPY \ORDENT\*.VW \OE\FILES
COPY \ORDENT\*.DB \OE\FILES
COPY \ORDENT\*.VWS \OE\FILES
COPY \ORDENT\*.KEY \OE\FILES
COPY \ORDENT\*.PIX \OE\FILES
```

Converting Menus

A RAD application consists of a series of integrated menus. Navigation menus, for traversing through an application and view menus, for manipulating views.

Generally, menus done by an SPL program should be recreated with the Developer. This allows you to take advantage of auditing options, user control, automatic documentation, a consistent user interface, and numerous other features. Menus are easily built within RAD Apps using the Ctrl A hot key to invoke the Developer. The Developer's Insert command allows you to quickly build the menu structure. Simply inserting a view will cause the Developer to generate a view menu.

All menus are controlled by usermode defaults allowing a System Administrator to decide which users can access particular choices. In other words, an administrator can set up usermodes so that a data entry clerk has a different set of menus than a manager or an administrator.

Converting Programs

Program files should be copied into the home directory or main directory of the application. Make sure you do not overwrite any programs that are already in the main directory. Any programs, or parts of programs used to control the menus should be deleted and recreated with the Developer.

The Developer provides a number of features so that programs can be easily integrated with the application's menus.

One feature is the function ($$fct) designator. The $$fct public variable is set to the contents of the menu item's function field before specified programs are executed. This is useful for passing information from the menu the program and allows different parts of an application to call the same program (passing different parameters to indicate where the program is being called from).

To create a new menu option that executes a program, use the Ctrl A Insert Program command sequence. To modify an existing view command, such as Enter or Update, so that it executes a program, use Ctrl A Modify [command] Item_Information and change the action text box to X (indicating that the choice should execute a program). Enter the program name in the object text box and, if necessary, enter a function in the function field to be passed to the object program.

Another feature provided by the Developer is the Application Libraries that are automatically loaded when the RAD App is started or a menu is activated. This library allows the programmer to create commonly used functions, and store them in one file to be accessed by the application and programs as needed. These functions can be used in view calculations, formulas in cells, or any other program. To add functions to the application library, use the Ctrl A Utilities Remember Application_Libraries command series.

RAD Apps also provide a means by which a program can be audited. Each time an audited program is run, a log entry is made containing the user identification, the initial setup, and any other information pertinent to program execution. Audited programs can also be added to job streaming routines. However, audited programs requires certain RAD App functions which are normally generated when such a program is created. In order to audit an existing program, a new program should be created through the menu options. This new program should be modified with the existing program read into it. Some moving of blocks will be required for any defined functions in the program to maintain proper syntax restraints. For more information on audited project files, refer to the Audited Programs section in Chapter 6.

Converting Definitions

RAD Apps provide users with the means to create, maintain and run their own reports, queries, sorts, mail merges, spreadsheet sends, and macros. As a developer, you can also can create “System” definitions which can be run by end users but not modified.

Converting the SPL program based application's definitions to the RAD App, requires some forethought about the use of the definitions. If the definition is to be used as a system definition, then the definition should be copied into the home directory of the system. Once again, this requires that you do not overwrite any existing files. The definitions can then be accessed by the programs or menus of the application.

Report Definitions

System report definitions should be placed in the home directory of the system. These reports are generally placed on the Print Menu by using the Developer's Insert Additional_Functions Print series of commands. Attach the report definition with the Report Layout option. Refer to the Report Components for Data Views section in Chapter 4 for more information.

Converting user modifiable report definitions requires some additional steps to be performed. The application maintains a listing of end user reports. To have the RAD App recognize user modifiable report definitions an appropriate entry must be made. User modifiable report definitions are contained in the report subdirectory of the home directory.

Once these reports have been copied into the proper directory, start the RAD App. Go to the appropriate data view (previously created) and select the Print option. This displays a menu of the available printing options. Select Custom Report and create entries for the appropriate reports.

Query Definitions

User modifiable query definitions work similarly to the report definitions in that the RAD App maintains a listing of all the available queries. System queries that are accessed by programs should go into the home directory of the system. User modifiable queries should be copied in to the query subdirectory below the home directory of the application.

User modifiable query definitions are added to the application by using the Query command on view menus.

Sort Definitions

User modifiable sort definitions work similarly to the report definitions in that the application also maintains a listing of all the available sorts. System sorts that are accessed by programs should go in the home directory of the system. User modifiable sorts should be copied to the sort subdirectory off the home directory of the application.

User modifiable sort definitions are added to the application by using the Order command on the view menus.

Cross-Tab Definitions

The Send to Spreadsheet option will allow you to use a Cross Tab definition on the data before it is sent to the spreadsheet. User modifiable Send to Spreadsheet Definitions work similarly to the Report Definitions in that the application maintains a listing of all the available Send to Spreadsheets. System Cross-Tab definitions that are accessed by programs should go in the home directory of the system. User modifiable Cross-Tabs should be copied to the sendss subdirectory off the home directory of the application.

User modifiable cross-tab definitions are added to the application by accessing the Print menu and selecting the Send to Spreadsheet option.

Mail Merge Definitions

RAD Apps allow end users to create, modify and run mail merges. If you want to convert a mail merge from your SPL program application to user modifiable mail merge, copy the documents to the merge subdirectory off the home directory of the application.

After copying the merge documents, you must create the remaining elements of the mail merge definition. To do this, run the application and go to a data view from which the mail merge can be performed. Access the Print command and select the Mail Merge option. Create a mail merge definition with the same name as the document previously copied.

Macro Definitions

Macro definitions work similarly to the report definitions in that the application maintains a listing of all the available macros. System macros that are accessed by programs should go in the main directory of the system. User modifiable macros should be copied to the macro subdirectory off the main directory of the application.

Macro Definitions are entered into a RAD App by using the Macro option on the System Maintenance menu.

Macro Definitions can be automatically loaded for users by their usermodes. The Menu option of Usermode_Info from the Usermode Maintenance allows you or the Administrator to specify the macro to load by placing the macro's definition name in the autoload macro field. This allows you to automatically load a macro set at a particular menu.

Index

Symbols

##activeimp, 231
##admin, 233
##ah, 233
##aphlp, 233
##audlg, 231
##audon, 234
##barbg, 234
##barfg, 235
##border, 238
##br, 239
##chlog, 231
##cmplt, 231
##combg, 235
##comfg, 235
##cpstep, 239
##curblock, 231
##curch, 235
##defin, 233
##dks, 231
##doslog, 233
##dsmch, 239
##grafg, 235
##lang, 233
##lev, 235
##lgfh, 231
##lmks, 231
##maxlv, 235
##mmod, 231
##num_keys, 238
##numlang, 231
##numum, 231
##pidx, 232
##post, 238
##prhlp, 233
##prhnd, 232
##pushn, 239
##tchlp, 233
##umks, 232
##wrkhnd, 232
##wrkhnd2, 232
##wrkhnd3, 232
##wrkhnd4, 232
#abort, 128
#choice, 165
#copies, 128
#end_page, 128
#key, 165
#mnu, 165
#mode, 165
#no_interrupt, 129
#start_page, 129
$$act, 234
$$adev, 176, 229
$$advr, 176, 229
$$ah, 234

$$answer1, 238
$$answer2, 238
$$answer3, 238
$$archpath, 229
$$areas, 237
$$botp, 229
$$brfld, 237
$$clm, 238
$$clttl, 234
$$cpsid, 238
$$crw_path, 229
$$default1, 238
$$default2, 238
$$default3, 238
$$desc, 234
$$dev_name, 230
$$dirsep, 230
$$fct, 234
$$file, 237
$$homedir, 176, 230
$$hphp, 234
$$hppt, 234
$$infoset, 232
$$key_prompt, 237
$$lang, 232
$$lpath, 232
$$lprog, 234
$$macpath, 176, 230
$$merpath, 176, 230
$$mttl, 234
$$obj, 234
$$opsys, 230
$$path, 238
$$persdir, 176, 232
$$proc, 239
$$procpath, 230
$$quepath, 177, 230
$$reppath, 177, 230
$$senpath, 177, 230
$$smen, 232
$$sorpath, 230
$$sysfiles, 177, 230
$$sysfiles2, 177, 230
$$sysfiles3, 177, 230
$$sysfiles4, 177, 230
$$sysfiles5, 177, 230
$$sysid, 230
$$syspath, 177, 230
$$topp, 231
$$tpltpath, 177, 231
$$um, 232
$$updel_expr, 238
$$usrid, 232
$$workdir, 177, 233
$_ttl, 231
$data1, 165
$dest_file, 128
$destination, 128
$one_record, 129
$records_used, 129
$report_type, 129
$sure, 239

.AHP, 241
.BDC, 241
.BMP, 241
.BTX, 241
.BWS, 241
.CFG, 242
.CFR, 242
.CTD, 242
.DB, 242
.DBQ, 242
.DFQ, 242
.DFR, 242
.DMP, 242
.DOC, 242
.DSL, 242
.DWL, 242
.FSI, 242
.GDB, 243
.GDC, 243
.GDE, 243
.GDH, 243
.GDS, 243
.GDT, 243
.GEN, 243
.HLP, 243
.IDX, 243
.IS?, 243
.ISV, 243
.KEY, 243
.LOG, 243
.LST, 243
.MAC, 243
.MNU, 243
.MRG, 244
.OSA, 244
.PF?, 244
.PHP, 244
.PIX, 244
.RCM, 244
.RF?, 244
.RPP, 244
.STK, 244
.TMP, 244
.TXT, 245
.UDC, 245
.USR, 245
.VW, 245
.VWS, 245
.WS, 245

Numerics

##cur, 235
##mod, 235
$$actm, 235
$$mac, 235
$$sn, 235
$$stat, 235
$$typ, 235
$$key_alias, 237
$$key_types, 237

$$keys, 237

A

Accessing SmartWare, 154
Accessing the Developer, 75
Actions, 247
ACTIVE_LEV, 183
Activity Log, 134
Add_Key, 93
ADD_LOG, 184
Adding Keys, 47
ANGOSS_Configuration, 103
appdoc, 179
Application Libraries, 161
Application_Library, 102
Applications
    New, 16
    Starting, 18
Archive, 82
Archive Job, 134
Archiving, 138
archjobs, 179
archlog, 179
ASK_FILENAME, 184
ASK_FILENAME_DIR, 184
ASVIEW, 247
Audited, 79
Audited Programs, 157

B

Background_Graphic, 104
Batch File, 178
Batch Update, 134
Beautify, 102
BROWSE, 247
Browse Mode, 32
Browse_Fields, 89

C

CA, 248
CALC, 248
Calling an Application, 160
Change-Color, 175
Changes_Log, 104
Changing Modules, 159
CHAP, 185
Characteristics, 95
CHECK_FILE_NAME, 185
CHECK_PATH, 186
CHOICE_TITLE, 186
CLEAR_KEYBOARD, 186
CLOSE_ALLWIN, 187
CLOSE_CUR_MENU, 187
Colors, 173
Communications, 88, 103, 104
Compile_All, 102

COMPLETE_LOG, 194
Converting
    Cross-Tab Definitions, 263
    Definitions, 262
    Macro Definitions, 263
    Mail Merge Definitions, 263
    Menus, 260
    Programs, 261
    Query Definitions, 262
    Report Definitions, 262
    Sort Definitions, 263
Converting Applications, 259
COUNT_WORDS, 195
Creating
    a New Application, 151
Creating Fields, 42
CRW, 248
Crystal Reports, 83, 98, 220
Custom Commands, 155
Custom Reports, 49
Custom-View Editor, 27

D

Data Paths, 176
Data View, 30, 38, 110
    Commands, 31
Data_Entry, 85
Database, 77, 103, 104
Data-File Menus, 45
db_info, 179
DECODE_EDA, 195
DEF_EDIT_AREAS, 196
Default_Edit_Areas, 90
Definition Files, 107
DELETE, 248
Delete, 32, 76
Delete_Key, 93
Developer Menu, 23
Developer System
    Accessing, 22
DISPLAY_MESSAGE, 196
DO_CALC, 196
Document Information, 122
Document View, 122
Documentation, 133, 144
DOCV, 249
DOS, 18
dsl, 179
DV, 249
dv_info, 179
DV_Info_(Sysdef), 137
DV_Info_(sysdef), 90
dwl, 179
DWL Definition, 124

E

Edit Areas Definition, 120
Edit_Area, 87, 90
EDIT_AREAS, 197

EDITAREA, 249
Editor, 96
ENTER, 249
ERRORREP, 198
Event Handling, 164
Execute, 79, 102
Extended Field Options, 44

F

FCT, 250
File Name Extensions, 241
File Structure Maintenance, 134
FIND, 250
FIND_COMMAND(), 198
Finding Records, 32
FINDRECORD, 198
form letter, 54
FPR_Keys, 91
FPR_Template, 91
FREE_CELL, 199
Function Call, 79
Function Listing, 183

G

GET_ANGII_VER, 199
GET_CHOICE, 200
GET_LINE, 203
GET_PATH, 204
Global, 103
GRAPH, 250
Graph, 86, 92

H

HardWare, 103
Help, 76
Help Files, 144
Hierarchy Diagram, 68, 137

I

Icon
    Editor, 175
Icon_Editor, 104
Icons, 174
Index
    Sort, 36
info_set, 179
Information, 92
Information Set, 178
Input Order, 47
Input Screen, 158
Input Screen Files, 245
Input_Screen, 92
Insert, 75
Inserting
    an Item, 76

Inserting a Data View, 26
Inserting a Menu, 23
INT_PATH, 204
Introduction, 13
Item_Information, 93, 109

J

Job Stream, 133
Job Streaming, 70
Job_Stream, 84

K

Key Field Order, 36
Key Fields, 28
KEY_FIELD_PROMPT, 205
Keyboard_Macros, 84
Keys, 35
KRUNCH, 250
Krunch, 32
KRUNCH_SCREEN, 205

L

language, 179
Language Maintenance, 134
Libraries, 161
Library, 94
Link_Definition_(DB), 94
Link_Definition_(SS), 94
Link_Definition_(WP), 95
Lint, 102
LOADDATAVIEW, 206
Loading/Unloading Programs, 163
loc_list, 179
LOCK_REC, 206
Login Program, 170

M

M, 251
MACRO, 251
Macro Definitions, 133
Mail Merge, 54
Make Application, 16
Management and Utilities, 133
MENU, 207
Menu, 76, 95
menu, 179
Menu Hierarchy Diagram, 133
Menu Libraries, 162
MENU_CALL, 208
Menu_Characteristics, 107
Menu_Colors, 103
MENU_TITLE, 209
Menus, 107
MERGE, 251

Modify, 75
Modifying
    Data View, 41
MODULE_RETURN, 210
Modules
    Changing, 159
Moving Systems, 177

N

$$stat, 236
Navigation Menu Programs, 167
New_Application, 82

O

ONE_RECORD, 209, 210
OPEN_FILE, 211
OPEN_LOG, 212
ORDER, 251
ORDER_COMMAND, 212
Ordering, 35
OS, 103, 252
OS_APL, 213
OS_Call, 80
OSA_Script, 96
OSAPL, 252
Other Software, 167

P

PASSWORD, 213
Password, 96
password, 22
Passwords, 134, 171
PCR, 252
Physical Order, 36
POP, 214
Preferences, 103
Print, 83
Print Menu, 48
Print_Crystal, 83
PRINT_DOC, 214
PRINT_T_FLD, 215
Printing, 48
PROC, 252
proced, 179
Procedure, 133
Procedures, 70
Process Log, 134
Processing Routines, 157
proclog, 179
Program, 79, 96
Program_(Loading/Unloading),
        97, 163
Programming, 149
PUSH, 215

Q

QUERY, 253
Query Definition, 33
Querying, 32
Quick Key Access, 134
Quit, 105

R

RAD Objects, 107
RAD Report, 56
RADHR, 17
READ_CHOICE, 216
READ_HEADER, 217
rel_info, 179
RELOAD, 218
Remember, 101
REORDER, 218
REPGEN, 253
REPLICATE DATAFILE, 139
Report, 87, 97
Report Components, 125
Report Generator, 50
Reposition, 76
Repositioning
    Menu Item, 40
RESTORE_ALLWIN, 219
Restructuring, 180
Return, 32, 84
RP, 253
RP_EXEC, 219
RT, 253
RTAS, 254
RTNS, 254
RUN_CRYSTAL, 220
RUN_QUERY_STR, 221
Running, 153

S

Screen Information, 110
Screen_and_DB's, 98
SCROLL, 222
Sending Data
    Spreadsheet, 52
    Word Processor, 54
SENDSS, 254
Sort, 35
Sort Index, 36
SORT_STRING, 222
SPL, 149
SPLIT_PFE, 222
Spreadsheet, 78, 88, 103, 104
Spreadsheet Information, 117
SSR, 254
Start Program, 170
Start Up, 152
Status Array Layout, 236
steps, 180

SUSPEND, 255
SUSPEND_APP, 223
SV, 255
Switching Modules, 167
sysactlg, 180
SYSDEF, 255
sysdef, 180
System Administration, 62
System Database Paths, 179
System Defaults, 134, 136
System Files, 241
System Title, 21
System_Defaults, 82, 99, 137

T

TEMP_EXIT, 223
TEXT, 255
Text_File, 81, 99
TOOLS_FILE, 224
Totals, 85, 99
Totals Definition, 116

U

Unix, 18
UP_DB_STAT, 224
UPDATE, 256
Update, 86
UPDATESS, 256
UPDATEWP, 256
Updating Records, 32
Upgrading, 180
user, 180
User Data Base, 133
User Mode Maintenance, 133
User Modes, 62, 135
User_Utilities, 85
usermode, 180
Users, 65
Utilities, 76, 101

V

Variables, 229
View Archive, 134
View_SYSDEF_Program, 100, 137

W

Windows, 18
Word_Processor, 88
Wordprocessor, 78, 103, 104
Worksheet, 100
WRITE_NAME, 225
WRITE_NUM, 225
WRITE_ULOG, 226

X

X, 256
X Window, 18
XAUD, 257

Z

ZAP, 257
ZAP_DB, 227