

California State University, San Bernardino

**CSUSB ScholarWorks**

---

Theses Digitization Project

John M. Pfau Library

---

2000

## Remote view manager for visual FoxPro application

Sustanie Harding

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd-project>



Part of the [Databases and Information Systems Commons](#)

---

### Recommended Citation

Harding, Sustanie, "Remote view manager for visual FoxPro application" (2000). *Theses Digitization Project*. 4317.

<https://scholarworks.lib.csusb.edu/etd-project/4317>

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact [scholarworks@csusb.edu](mailto:scholarworks@csusb.edu).

REMOTE VIEW MANAGER FOR VISUAL FOXPRO APPLICATION

---

A Project  
Presented to the  
Faculty of  
California State University  
San Bernardino

---

In Partial Fulfillment  
of the Requirements for the Degree  
Master of Science  
in  
Computer Science

---

by  
Sustanie Harding  
December 2000

REMOTE VIEW MANAGER FOR VISUAL FOXPRO APPLICATION

---

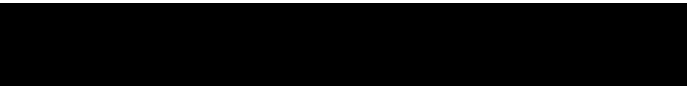
A Project  
Presented to the  
Faculty of  
California State University  
San Bernardino

---

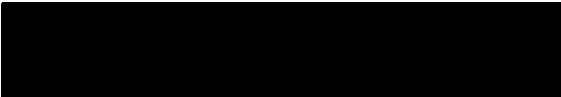
by  
Sustanie Harding

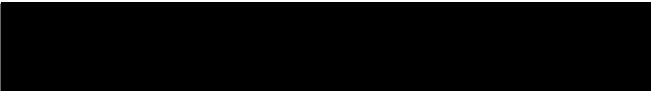
December 2000

Approved by:

  
Dr. George M. Georgiou, Chair

10/24/00  
Date

  
Dr. Josephine Mendoza

  
Dr. Kerstin Voigt

## ABSTRACT

In a distributed programming environment, there are many applications that use a single Database Container (DBC). Within these programming applications, programmers are able to modify the DBC. In turn, these modifications make maintenance of the DBC very difficult as the number of applications increase in the system. In this project, the Remote View Mapping (RVM) tool has been developed to assist programmers in the maintenance of the DBC. The purpose of the RVM is to establish a relationship between open applications and the DBC. This tool is dynamic with the applications constantly modifying the DBC.

## TABLE OF CONTENTS

ABSTRACT .....	iii
LIST OF TABLES .....	viii
LIST OF FIGURES .....	x
1. CHAPTER 1 SOFTWARE OVERVIEW.....	1
1.1 Introduction.....	1
1.2 Product Overview and Summary.....	6
1.3 Development and Operating Environment.....	8
2. CHAPTER 2 METHODOLOGY.....	9
2.1 Structure of a Visual FoxPro Project.....	9
2.2 Search Algorithm.....	18
2.3 Entity Relationship Diagram.....	21
2.4 Implementation.....	25
2.5 Security .....	29
2.6 Exception Handling.....	29
2.7 Conclusion.....	29
2.8 Future Modification and Enhancement.....	30
3. CHAPTER 3 SOFTWARE DESCRIPTION.....	32
3.1 General Layout (Main Form).....	32

3.1.1 Title Bar and Menu Items.....	34
3.1.2 Synchronized Applications.....	38
3.1.2.1 Directory Selector .....	40
3.1.2.1.1 Visual FoxPro Fox Icon and Title Bar	41
3.1.2.1.2 Close Button.....	42
3.1.2.1.3 Directory Tree Structure.....	42
3.1.2.1.4 Select Command Button.....	43
3.1.2.1.5 Cancel Command Button.....	44
3.1.2.1.6 Drive.....	44
3.1.3 Database Container ComboBox (DBC) .....	45
3.1.4 File List Command Button.....	46
3.1.4.1 File Selector .....	47
3.1.4.1.1 Directory Selector ComboBox.....	48
3.1.4.1.2 Question Mark Button.....	48
3.1.4.1.3 Close Button.....	49
3.1.4.1.4 Up one level Command Button.....	50
3.1.4.1.5 View Desktop Button.....	50
3.1.4.1.6 Create New Folder Button.....	51
3.1.4.1.7 List View Button.....	52
3.1.4.1.8 Detail View Button.....	52
3.1.4.1.9 Folder/File List.....	53
3.1.4.1.10 Open DBC.....	54
3.1.4.1.11 File Types ComboBox.....	55

3.1.4.1.12 OK Command Button.....	55
3.1.4.1.13 Cancel Command Button.....	56
3.1.4.1.14 Help Command Button.....	57
3.1.5 Main Container.....	58
3.1.5.1 'Display All' Radio Button .....	59
3.1.5.2 'Display By Applications' Radio Button ....	60
3.1.5.3 'Display By Remote View' Radio Button .....	60
3.1.5.4 Applications Grid .....	61
3.1.5.5 Remote Views Grid .....	63
3.1.5.6 Add Command Button .....	66
3.1.5.7 Delete Command Button .....	68
3.1.6 Secondary Container.....	69
3.1.6.1 View Name Textbox .....	70
3.1.6.2 SQL Statement Listbox .....	71
3.1.6.3 First VCR Command Button .....	72
3.1.6.4 Previous VCR Button .....	73
3.1.6.5 Next VCR Button .....	74
3.1.6.6 Last VCR Button .....	75
3.1.6.7 Save Button .....	76
3.1.6.8 Done/Cancel Button .....	77
3.1.7 Report Command Button.....	79
3.1.8 Print Search Tree Command Button.....	80
3.1.9 Quit Command Button.....	81

4. CHAPTER 4 UNIT TESTING.....	82
APPENDIX A: GLOSSAY OF TERMS .....	83
APPENDIX B: SOURCE CODE OF REMOTE VIEW MANAGER (RVM) ....	85
REFERENCE ,.....	121



## LIST OF TABLES

Table 1: Contents of the Visual FoxPro Project Table	
(test.pjx) .....	11
Table 2: Contents of the Visual FoxPro Form Table	
(test.scx) .....,...	13
Table 3: Contents of the Visual FoxPro Form Table	
(test.scx) .....	15
Table 4: Contents of the Visual FoxPro Database Container	
Table (test.dbc) .....	16
Table 5a: Tables used in the Applications .....	24
Table 5: Title Bar Buttons .....	37
Table 6: Synchronized Applications .....	39
Table 7: Visual FoxPro Fox Icon and Title Bar .....	41
Table 8: Close Button .....	42
Table 9: Directory Structure .....	43
Table 10: Select Command Button .....	43
Table 11: Cancel Command Button .....	44
Table 12: Drive ComboBox .....	45
Table 10: DBC Combobox .....,...	45
Table 14: File List Command Button .....	46
Table 15: Directory Selector ComboBox .....	48
Table 16: Question Mark Button .....	49

Table 17:	Close Button .....	49
Table 18:	Up One Level .....	50
Table 19:	View Desktop Button .....	51
Table 20:	Create New Folder Button .....	51
Table 21:	List View Button .....	52
Table 22:	Detail View Button .....	53
Table 23:	Folder File List .....	54
Table 24:	Open DBC .....	54
Table 25:	File Types ComboBox .....	55
Table 26:	OK Command Button .....	56
Table 27:	Cancel Command Button .....	56
Table 28:	Help Command Button .....	57
Table 29:	'Display All' Radio Button .....	59
Table 30:	'Display By Applications' Radio Button .....	60
Table 31:	'Display by Remote View' Radio Button .....	61
Table 32:	Data structure of the "viewappl" table .....	62
Table 33:	The 'Applications' grid .....	63
Table 34:	Data structure of the DBC. ....	64
Table 35:	The 'Remote Views' grid .....	65
Table 36:	The Add Command Button .....	67
Table 37:	The Delete Command Button .....	68
Table 38:	The 'View Name' textbox .....	71
Table 39:	The 'SQL Statement' Listbox .....	72

Table 40:	First VCR Button .....	73
Table 41:	Previous VCR Button .....	74
Table 42:	Next VCR Button .....	75
Table 43:	Last VCR Button .....	76
Table 44:	Save Button .....	77
Table 45:	Done Button .....	78
Table 46:	Report Button .....	79
Table 47:	Print Search Tree Button .....	80
Table 48:	Quit Button .....	81
Table 49:	Test Result .....	83

## LIST OF FIGURES

Figure 1:	Visual FoxPro Applications are independent of the backends. ....	1
Figure 2:	A Database Container with Remote Views .....	4
Figure 3:	Remote Views are shared between applications. ..	7
Figure 4:	Visual FoxPro Project. ....	9
Figure 5:	Sample Visual FoxPro Form. ....	11
Figure 6:	Example of a possible search path. ....	15
Figure 7:	Entity relationship Diagram .....	21
Figure 8:	Relationship between Applications and Remote Views(Display All) .....	25
Figure 9:	Relationship between Applications and Remote Views(Display By Applications) .....	26
Figure 10:	Relationship between Applications and Remote Views(Display by Remote View) .....	28
Figure 11:	Main Form .....	32
Figure 12:	Main Form, Second Container. ....	33
Figure 13:	Title Bar .....	34
Figure 14:	File Drop Down Menu Item .....	34
Figure 15:	Find Drop Down Menu Item .....	35
Figure 16:	Find Application and Find Remote View Form. ..	35
Figure 17:	Help Drop Down Menu Item .....	36

Figure 18:	Object on the Main Form .....	38
Figure 19:	Directory Selector .....	40
Figure 20:	File Selector .....	47
Figure 21:	Main Container. ....	58
Figure 22:	Secondary Container .....	69

## 1. CHAPTER 1 SOFTWARE OVERVIEW

### 1.1 Introduction

There are two different ways in which Visual FoxPro Applications can retrieve data independently of the Database in a Client/Server environment using: 1) Sequential Query Language (SQL) Pass-Through Functions; and 2) Remote Views in a Database Container (DBC). Both SQL Pass-Through and the DBC use Open Database Connectivity (ODBC) to connect and communicate with the different databases (Figure 1).

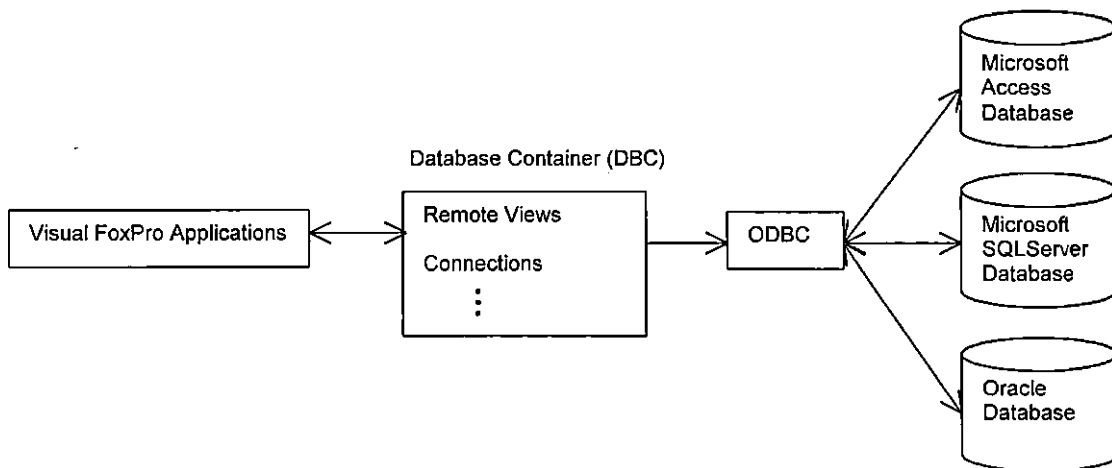


Figure 1: Visual FoxPro Applications are independent of the backends.

ODBC is the protocol for accessing data from various database engines such as SQL Server, Oracle, Informix and, text files. This ability allows Visual FoxPro to be a powerful development language. Applications can be written independent of the database. SQL Pass-through is the term given to the technique of sending instructions directly to the data source [4]. The data source is the database that contains data the application will be using. SQL statements are then passed directly through ODBC to the database to perform the necessary database functions.

In a Client/Server environment the data and the application are usually stored on two separate machines. The client asks for data and the server figures out what data is required, and sends it back [4]. The DBC is a database that stores the connection, tables, indexes, views, rules and code to maintain the relationship between the tables. The DBC database stores local tables, local views, local stored procedures, remote views and the connection information needed to access the remote tables that are being used by the remote views. A view is a "window" on a set of data returned by the data source

indicating that this data matches some specified criteria in the definition of the view [4].

An example of this would be a listing of all Order records for a particular customers, i.e. Order\_by\_Cust.CustId = 'GOLD1'. Views are persistent and somewhat similar to Cursor (CuRrent Set Of Records) except views are not temporary, read only tables like Cursors. Views can be referred to as 'virtual tables' [1].

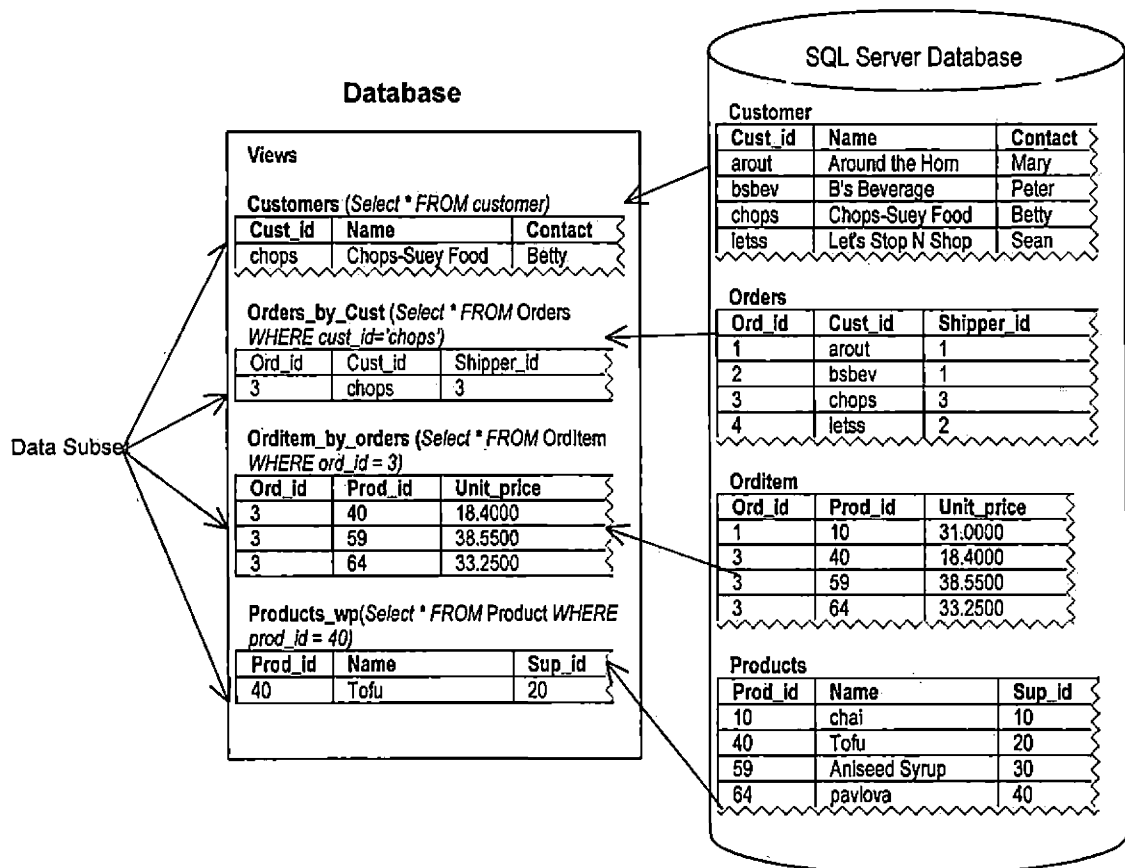




Figure 2: A Database Container with Remote Views

Local views deal with local FoxPro data and remote views deal with data from a remote data source, i.e. the server e.g. SQL Server. Remote views use the connection to connect to the remote database and retrieve the result set. "The connection is an object that stores information about the current data source connection, such as user ID and password" [4]. This allows programmers to make changes to the view as it appears to them and not affect the remote view.

The advantages of using a DBC for data access are numerous. For example, using a DBC allows the programmer to access information quickly and thus keeping the flow of traffic on the network from becoming sluggish. It also allows the programmer to specify criteria and access only the data needed.

When a traditional application needs data from a file server it must open the table where the data is located and then download the data to the user's machine. This takes up valuable computer time in cases where the table is large. The network becomes crowded retrieving data that may not be used in the application.

With the DBC and remote views, the application (client) asks the server to retrieve the data with the criteria listed in the view. The server searches through the data source and returns to the application only the records that match the criteria specified in the view. The network traffic is greatly reduced and the application does not have to sort out unwanted data. Given that the DBC is the central location for all views used in a set of application, the data can be retrieved repeatedly by an application without rewriting the same SQL statement on each instance.

In the cases where a parameter is used to limit the result set, the view can retrieve a different small result set each time by just changing a parameter. In addition, applications that perform similar processes can share the DBC.

## 1.2 Product Overview and Summary

The Remote View Manager (RVM) is a valuable programming tool that displays a mapping of remote views to the applications. This is a necessary tool that is needed for efficient programming in a development environment. In this environment, there are many programmers working independently at several sites. Remote views are created and added to the DBC as needed without any verification of whether a similar view exists and that would otherwise be shared between applications.

In cases where remote views are shared between applications as shown by Figure 3, the deletion or modification of remote views can have serious repercussions. In Figure 3, deleting or modifying the view "customer" on the customer table will affect three applications: the Customer Editor, the Customer Orders and the Customer Profile. The purpose of the RVM is to provide an easy-to-use tool that allows developers to quickly and efficiently check the relationship between remote views and an open application. In addition RVM allows the programmer to add, edit and delete views as needed.

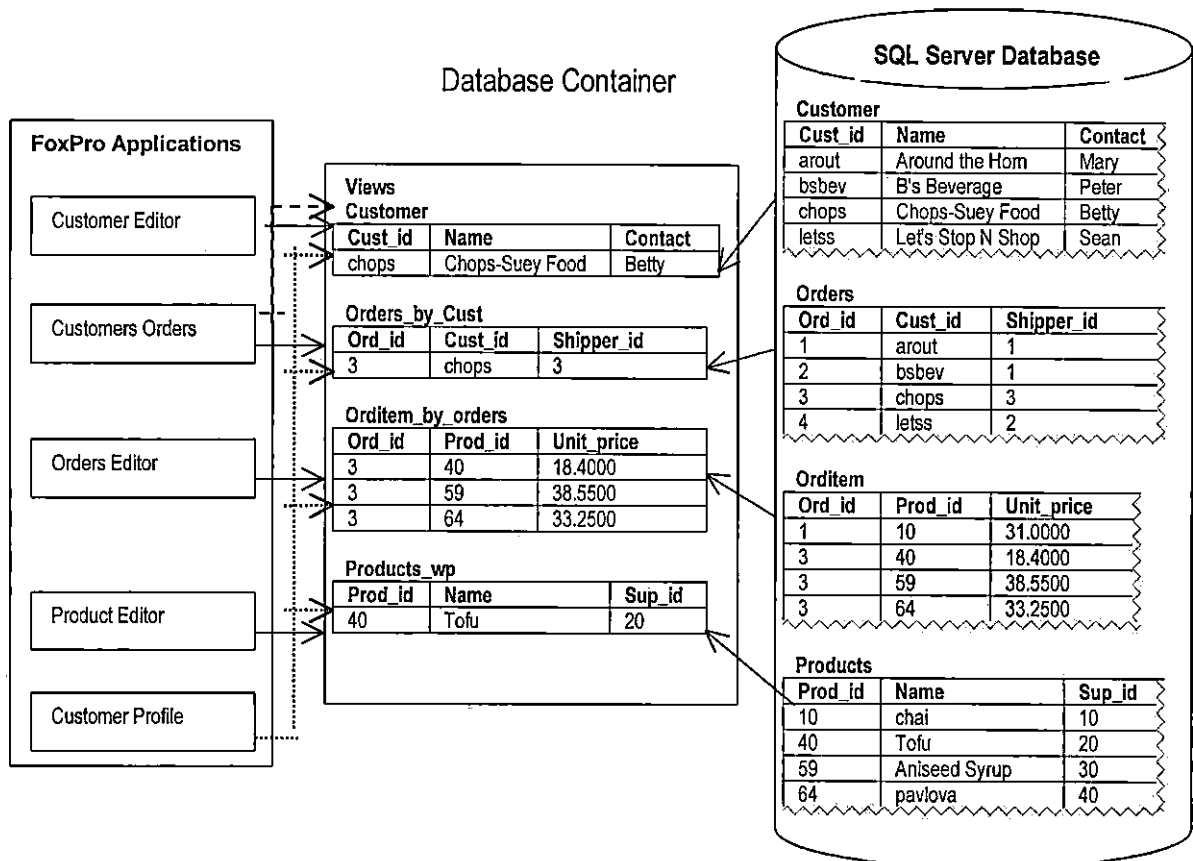


Figure 3. Remote Views are shared between applications.

### **1.3 Development and Operating Environment**

The following is the hardware and software requirements that will be used in the development and maintenance of the RVM application.

1. Hardware: IBM Compatible System with the following:

a. Pentium 160 MHz Processor.

b. 64 MB RAM.

c. 2 GB HD.

d. 3 ½" floppy drive

e. 8x IDE CD-ROM drive

f. 17" SVGA Monitor

g. 104 standard keyboard

h. MS compatible mouse

2. Software:

a. Microsoft Windows NT/95 Operating System.

b. Microsoft Access

3. Language:

a. Visual FoxPro 6.0.

b. ANSI Standard Query Language (SQL)

## 2. CHAPTER 2 METHODOLOGY

### 2.1 Structure of a Visual FoxPro Project

All Visual FoxPro projects are composed of a project file (.pjx). The project file contains all information about the application. The forms, classes, data and

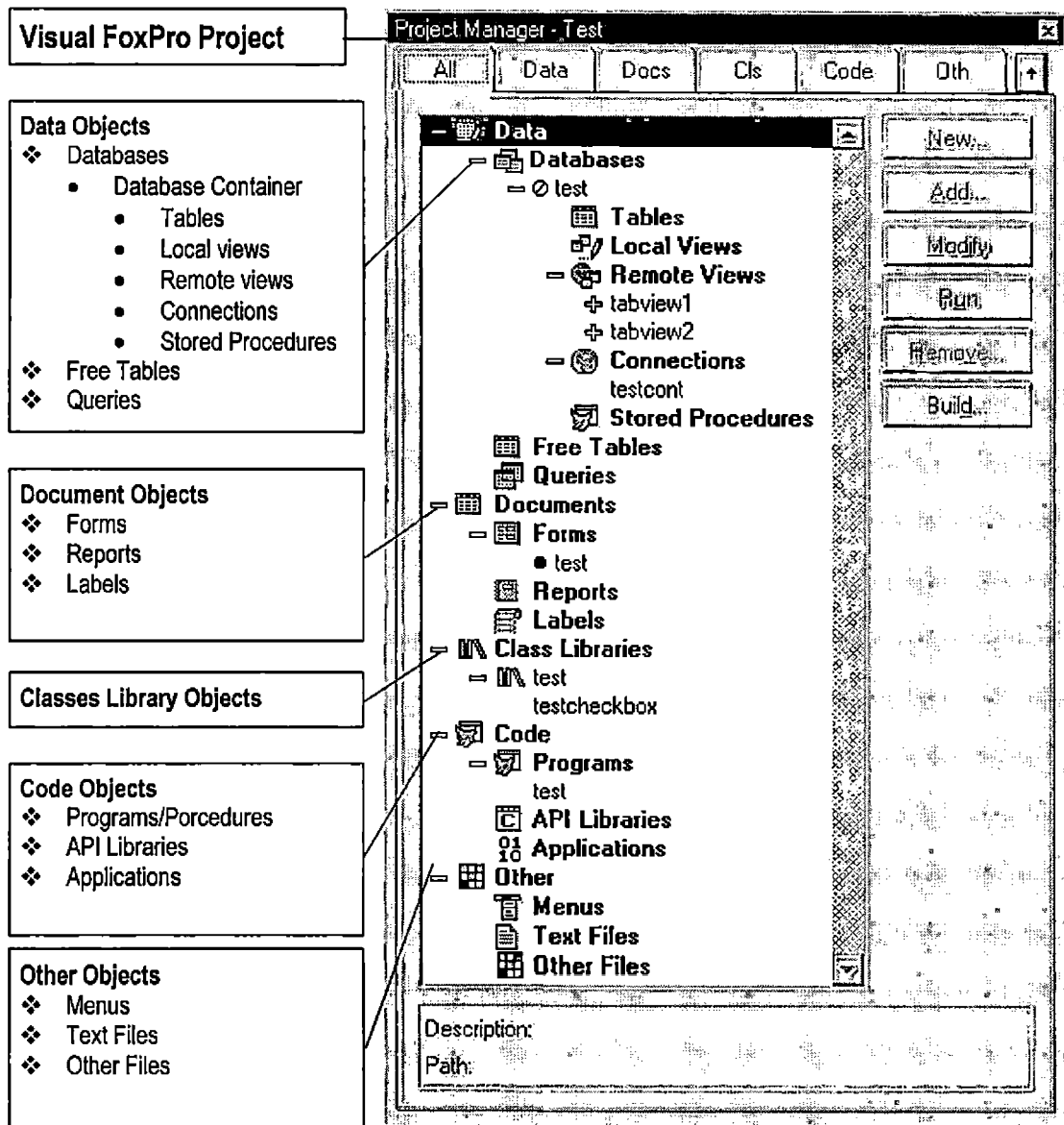


Figure 4. Visual FoxPro Project.

programs that are used in the making of the application are located in the project file (Figure 4).

The project file is a table that is maintained by Visual FoxPro and presented to the programmer in a user friendly GUI. This table can be browsed and edited by the programmer. Please note, this is valuable and volatile information that, if not handled carefully, can result in the programmer not being able to access the program using the Visual FoxPro GUI ever again. Table 1, shows an example of a Visual FoxPro Project. In this case, the project illustrated in Figure 4 shows the contents of the Visual FoxPro Project Table '**test**'. The Visual FoxPro Project '**test**' is stored in a table '**test.pjx**'. The project file has the following structure, however not all columns are illustrated, as many are used by FoxPro to maintain the project and have no significance to this Masters' project.

test.pjx						
Name	<sup>1</sup> Type	Outfile	Homedir	Exclude	Mainprog	Savecode
C:\Project\test\test.pjx	H	<Source>	C:\Project\test			T
test.dbc	D			T		
test.scx	K			F	T	
Test.vcx	V			F		
C:\snetcs\classes\attend.vcx	V			F		
C:\snetcs\classes\controls.vcx	V			F		
test.prg	P			F	F	

Table 1: Contents of the Visual FoxPro Project Table (test.pjx)

The first record contains the name of the project along with the search path. The next four entries show the name of the DBC, form, classes and program used in the Visual FoxPro Project 'test'.

The image shows a Visual FoxPro form titled "Test Form". The form has a grid background and contains several controls. Labels with leader lines point to various elements:

- Visual FoxPro Form**: Points to the title bar of the form.
- test**: Points to the form's name in the project browser.
- lblname**: Points to the label "Name" above the "txtName" text box.
- txtName**: Points to the text box for the name.
- Attmonths**: Points to the "Attendance: Month" radio button.
- lblCourse**: Points to the label "Course" above the "Course" text box.
- Course**: Points to the text box for the course.
- txtCourse**: Points to the text box for the course.
- From Month**: Points to the "From Month" label.
- From Date**: Points to the "From Date" label.
- To Month**: Points to the "To Month" label.
- To Date**: Points to the "To Date" label.
- Save**: Points to the "Save" button.
- Quit**: Points to the "Quit" button.
- CmdSave**: Points to the "Save" button.
- CmdQuit**: Points to the "Quit" button.

Figure 5. Sample Visual FoxPro Form.

<sup>1</sup> H-Project, d-DBC, K-Form, V-Classes, P-Programs.



In developing an application, controls are added to the Visual FoxPro Form. These controls are objects that have Properties and Methods that associate them with remote views. In Figure 5, the form '**test**' has objects of different types that have been added to the form. Not all objects that are added to a form are associated with a remote view, for example the labels 'lblName' and 'lblCourse'. All the same, many objects added to the form can be directly associated to a remote view. For example, the textbox 'txtname' is a simple FoxPro object and has a <sup>2</sup>'ControlSource' of tabview1.descript. Other objects can be indirectly associated with remote views. The form 'test' opens a remote view for the 'Attmonths' object. The Visual FoxPro Form '**test**' is stored in the table '**test.scx**'.

The form file has the structure illustrated in Table 2, however not all columns are illustrated as many are used by FoxPro and have no significance to this Masters project.

---

<sup>2</sup> Specifies the source of data to which the object is bound.

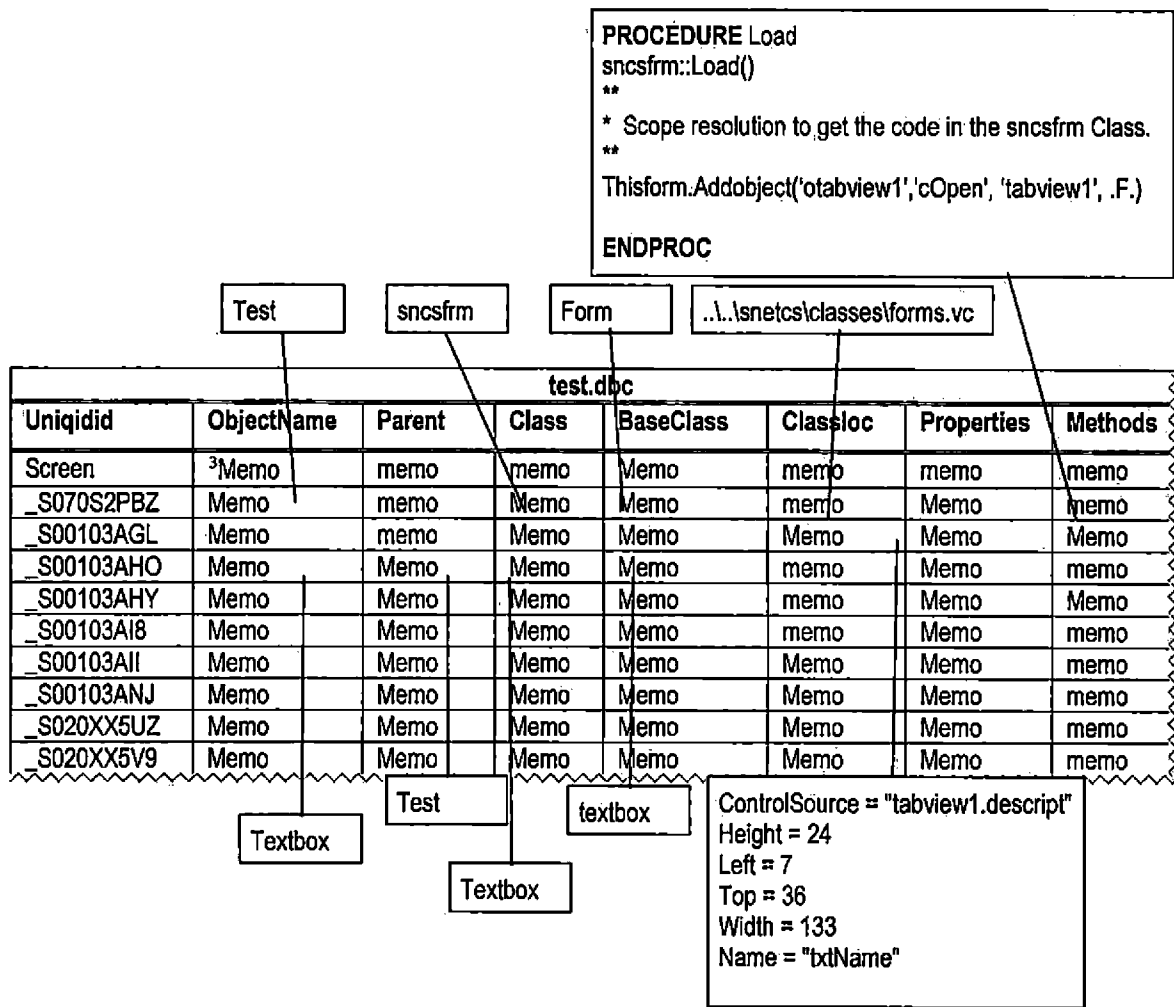


Table 2: Contents of the Visual FoxPro Form Table (test.scx)

The problem of collecting the views that are used in an application was tackled by first recursively searching through the project file to find the forms and classes used in the application. Once this was collected, the forms were recursively searched for references to remote views.

References to remote views in Methods (Table 2) occurs with following syntax:

- `AddObject('o[RemoteView]', 'cOpenTable', '[Remote View]')`

References to remote views in Properties (Table 2) occur in the following syntax:

- `ControlSource = '[Remote View].fieldname'`
- `RecordSource = '[Remote View]'`

The search routine was able to search through many layers of recursion objects to collect all references. As shown in Figure 6, each object that makes up a project can reference other objects. These objects might have a reference to Remote Views and must be searched until the recursion ends.

---

<sup>3</sup> Lowercase memo fields have no data. Uppercase Memo fields have data that are too large to display and callouts will be used to show example of data.

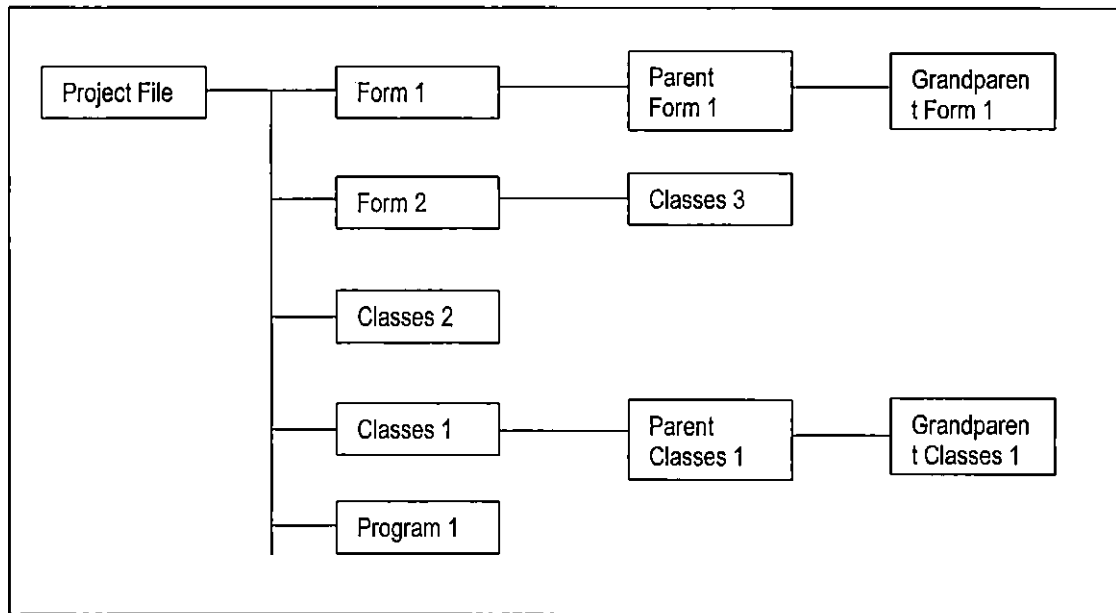


Figure 6. Example of a possible search path.

All distinct incidents of views in an application were stored in the table ViewAppl with the following data structure.

ViewAppl		
	Fieldname	Type
PK <sup>4</sup>	Viewuniq	Int
U1 <sup>5</sup>	ViewName	Varchar(32)
U1	App!Name	Varchar(32)

Table 3: Contents of the Visual FoxPro Form Table (test.scx)

<sup>4</sup> PK - Primary Key

<sup>5</sup> UK - Unique Key.

In the example illustrated by Figure 4, the DBC for the Visual FoxPro Project '**test**' is stored in a table '**test.dbc**'. The table '**test.dbc**' has the following entries.

test.dbc							
Objectid	Parentid	Objecttype	ObjectName	Property	Code	User	Riinfo
1	1	Database	Database	6Memo	memo	memo	
2	1	Database	TransactionLog	Memo	memo	memo	
3	1	Database	StoredProcedureSource	Memo	memo	memo	
4	1	Database	StoredProcedureObject	Memo	memo	memo	
5	1	Connection	Testcont	Memo	memo	memo	
6	1	View	Tabview1	Memo	memo	memo	
7	6	Field	Atype	Memo	memo	memo	
8	6	Field	Descript	Memo	memo	memo	
9	6	Field	Whichuniq	Memo	memo	memo	
10	6	Field	Priority	Memo	memo	memo	
11	1	View	Tabview2	Memo	memo	memo	
12	11	Field	Asbaccuniq	Memo	memo	memo	
13	11	Field	Auniq	Memo	memo	memo	
14	11	Field	Atype	Memo	memo	memo	
15	11	Field	Itmgrpuniq	Memo	memo	memo	
16	11	Field	Workbal	Memo	memo	memo	
17	11	Field	Negbalok	Memo	memo	memo	

Table 4: Contents of the Visual FoxPro Database Container Table (test.dbc)

Note the relationship between the Parentid and the Objectid. The parentid shows the hierarchy of the entries in the table '**test.dbc**'. Records with objectid 6 and 11 have a parentid of 1 and are of Objecttype 'View'. These are the tabview1 and tabview2 views in test.dbc illustrated in Figure 4. Similarly, records with objectid 7,8,9 and 10

have a parentid of 6, and are of objecttype 'Field'. These are the fields that were selected in the RVM search. This Project searches through the DBC collecting the view names and storing them in a table. This table is dynamic, as the DBC is dynamic, and changes with each new and/or updated application.

---

<sup>6</sup> Memo fields in Visual FoxPro are fields that store block data.

## 2.2 Search Algorithm

A depth-first algorithm was used as a search algorithm. This algorithm was chosen because of the data structure of Visual FoxPro projects, forms and classes. Depth-First was chosen over Breathe-First because Depth-First search uses less memory than breathe first. Breath-first search searches the nodes on the same level first and stores each node at that level, so that they can be refereed to see if it have any children, when the search at that level is completed. Depth-First search requires that you keep track of the parent node only. The pseudo-code for the algorithm follows.

```
IF BuildProjectList() == .T.  
    BuildControlList()  
    RemoveDuplicates()  
ENDIF
```

Procedure BuildProjectList()

BEGIN

    Get the root directory

    Scan and redirect DOS <dir> function to a file

    Scan file for all directories and insert into table (**dirstruct**)

    Open all directories found in the **dirstruct** table and scan for Visual FoxPro (VFP)

    Project files i.e. a file with the .pjx extension

    IF the directory has a .pjx

        BEGIN

            Scan for VFP Forms i.e. all files with the .scx extension and Insert all references into a table (**vfarmmst**) with the project name, form name.

            Open all .scx files and check if the form is marked as the main form and tag all forms as such.

            Open forms find the method on the forms and scan method for all references to views i.e. Strings with "ADDOBJECT" and "COPENTABLE" in the same line. If the strings are found parse the string and insert the values in the **viewappl** table

        END

END

Procedure BuildControlList ()

BEGIN

    Open the **vfarmmst** table

    Do While !EOF ()                      &&End of File

        BEGIN

            Open Form and check if the form is not the base class (i.e. The form has a parent)

            Save the record pointer

            FetchControls() && on the class

            Go to the record

            FetchControls() && on the parent

        END

END

Procedure FetchControls()

BEGIN

    Open the class or form

    Do While !EOF ()                      &&End of File

        BEGIN

            Open Class or Form and check if the form is not the base class

            IF not baseclass

                FetchControls()

            ELSE

                Insert reference into table **vfarmctl** to show what control was searched.

            Scan method of control for all references to views i.e. Strings with

                "ADDOBJECT" and "COPENTABLE" in the same line. If the strings are found parse the string and insert the values in the table **viewappl**.

            IF object has Record Source and/or Control Source Reference to the **viewappl** table

        END



```
END  
PROCEDURE RemoveDuplicates()  
BEGIN  
    Select all tables and remove all duplicates  
END
```

## 2.3 Entity Relationship Diagram

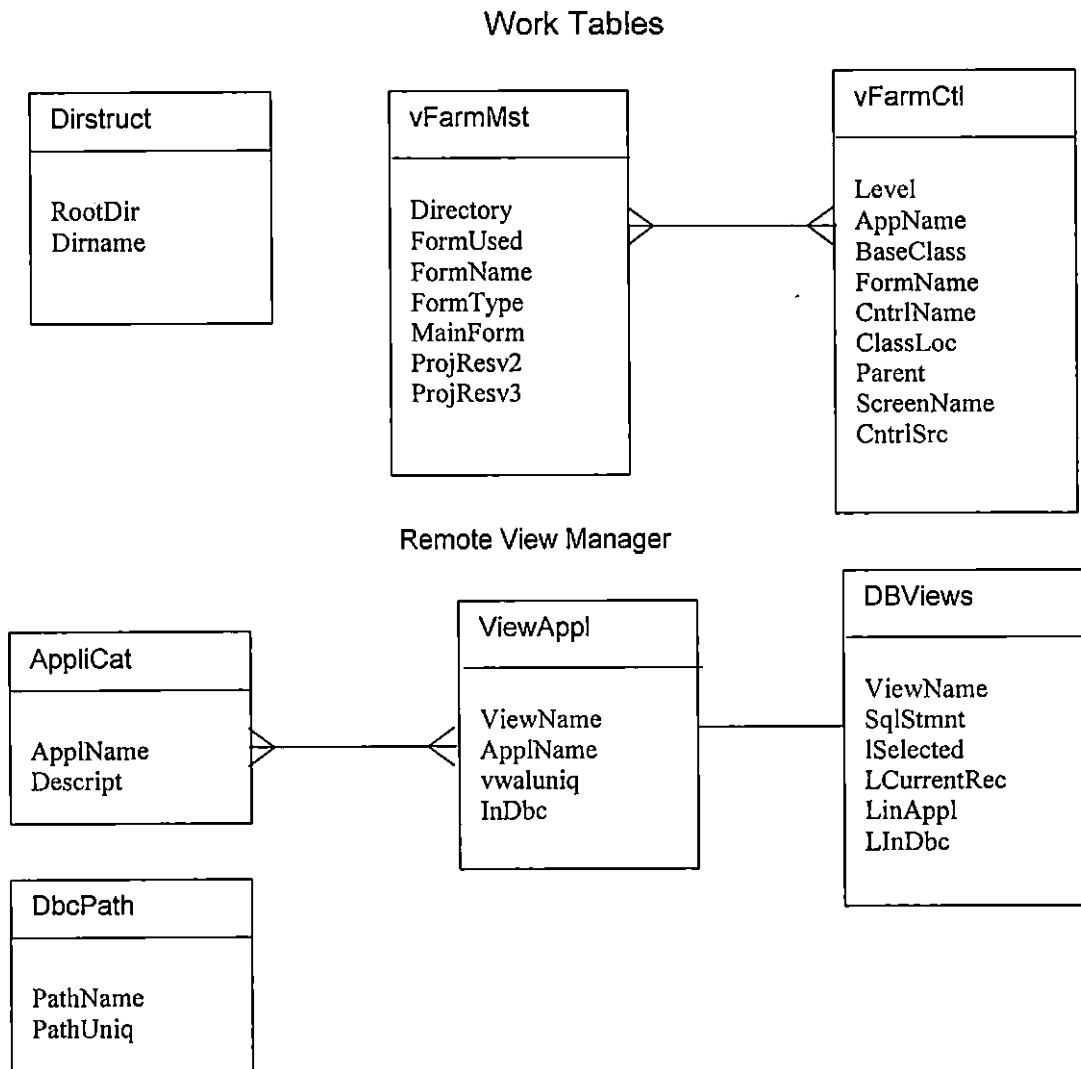


Figure 7. Entity relationship Diagram

The tables in the Entity Relationship Diagram are explained in the following table.

Table Name	Description
Dirstruct	Used to store the directories that will be traversed in the search process.
vfarmMst	Used to store the form names found in each directory and tag each form if it is the main form of the project. The form is tagged because the search routine will need to find a root form per directory.

vFarmCtl	Every Visual FoxPro Form and control searched is inserted into this table. This table is a validation of the search process. If the search program was not successful this table can be used to debug the search algorithm. All records are not distinct in this table.
DBCPath	Stores the last ten references of DBC's and their paths.
AppliCat	Applications that were searched. This table is updated from the Search routine.

ViewAppl	Stores the application names and the views that are used in by the applications. This table is not totally normalized for ease of reference. All records are distinct in this table.
DBViews	Stores the views and the SQL statements used by the views.

Table 5a: Tables used in the Applications

## 2.4 Implementation

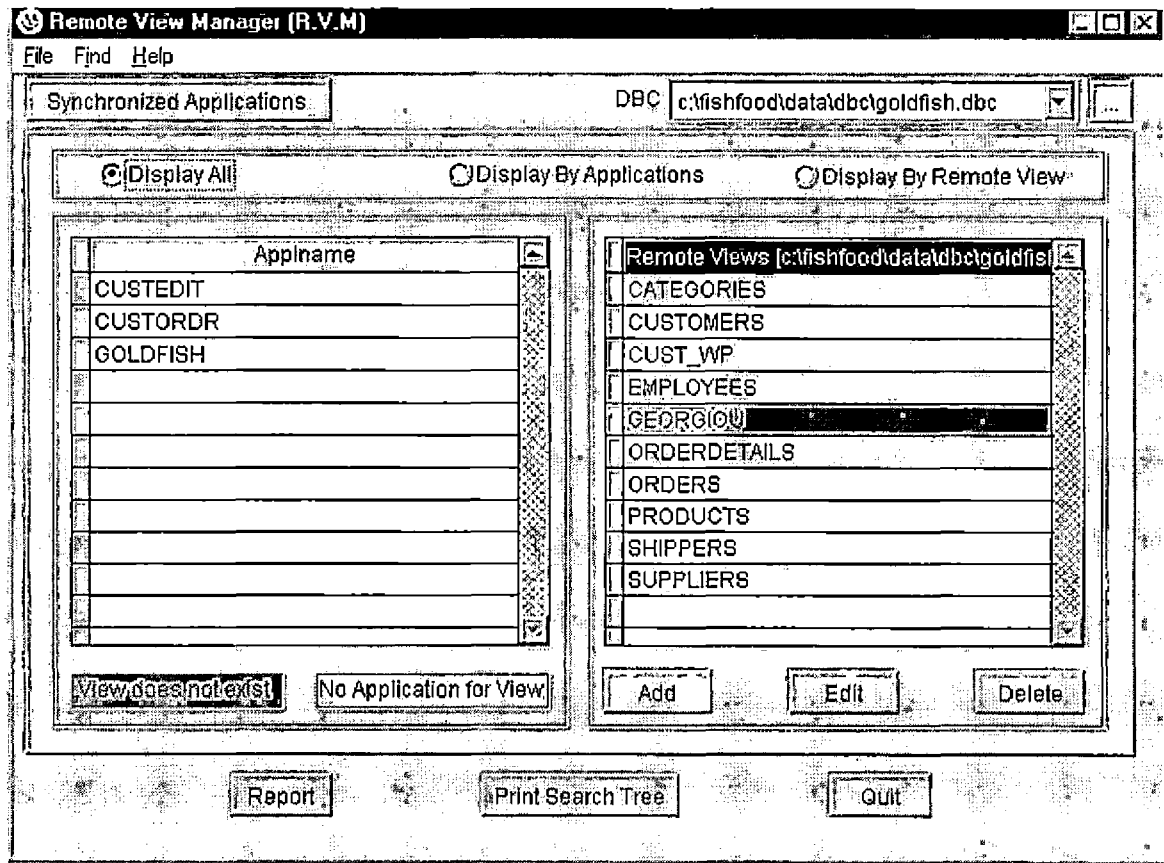


Figure 8. Relationship between Applications and Remote Views (Display All)

By selecting the 'Display All' option button the user is able to see all application and all remote views that are found in the applications. The 'Remote View' grid is color coded to show views that are not referenced by any application and view that are referenced in an application but does not exist in the selected DBC. In Figure 8 the views *CATEGORIES*, *CUST\_WP*, *EMPLOYEES*, *ORDERDETAIL* and

*SUPPLIERS* are not used by any of the applications. The view *GEORGIOU* on the other hand is referenced in an application but does not exist in the DBC. The other views *CUSTOMERS*, *ORDERS*, *PRODUCTS* and *SHIPPERS* were referenced in the applications *CUSTEDIT*, *CUSTORDR* and *FISHFOOD* and exist in the DBC.

On further investigation the user can choose the 'Display by Applications' option button. When the user

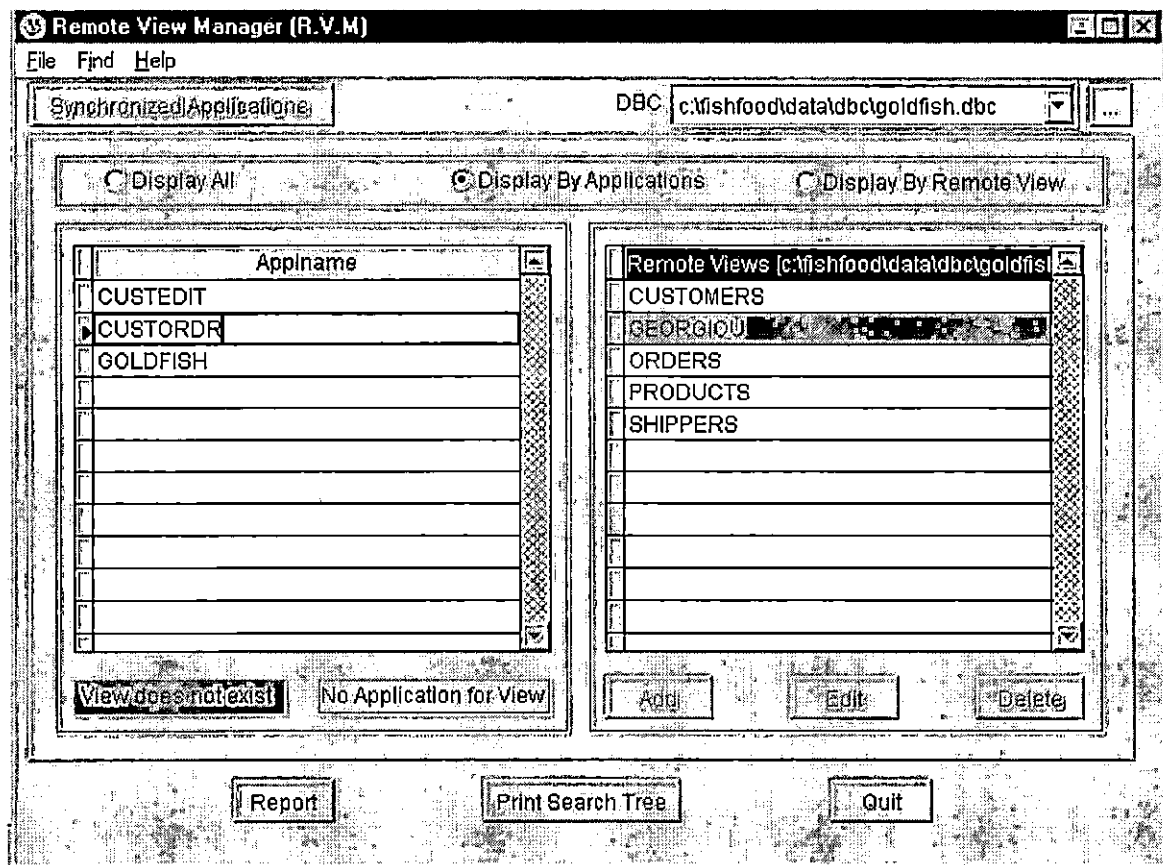


Figure 9. Relationship between Applications and Remote Views (Display By Applications)

selects this option the Remote View Grid refreshes and shows the user all views that are referenced by the selected application. In Figure 9, the application selected is *CUSTORDR* and the views referenced in this application are *CUSTOMERS*, *GEORGIOU*, *ORDERS*, *PRODUCTS* and *SHIPPERS*. The view *GEORGIOU* does not exist in the DBC and as such could have the diverse effect on the implementation of the application *CUSTORDR*. The view *GEORGIOU* is coded red so that programmers can easily identify potential problems with their applications. This view can be added to the DBC by evoking the ADD button. After adding this view to the DBC the Remote View Grid refreshes and the View *GEORGIOU* is no longer coded red as the view now exist in the DBC.



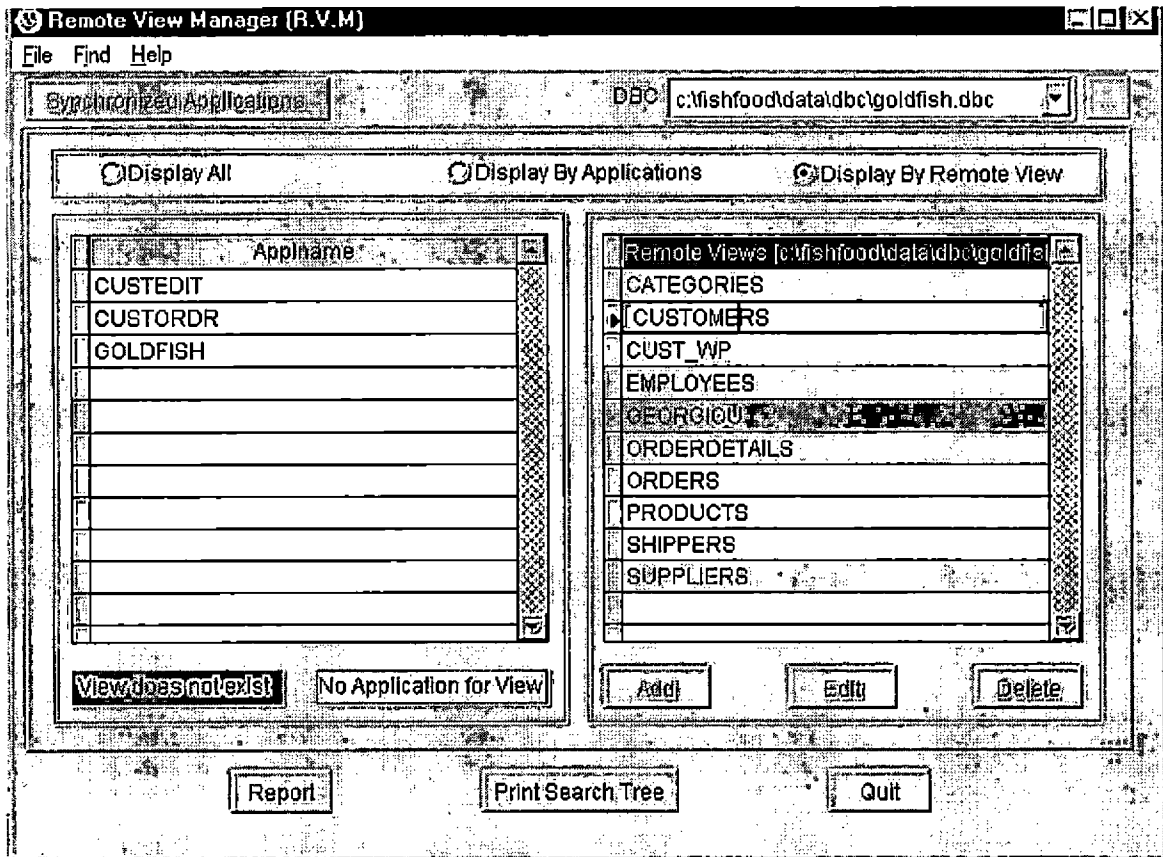


Figure 10. Relationship between Applications and Remote Views(Display by Remote View)

When the user chooses the 'Display by Remote View' option button the applications Grid refreshes to show the applications that are using a Remote View. In Figure 11, the Remote view *CUSTOMER* is being used by the applications *CUSTEDIT*, *CUSTORDR* and *FISHFOOD*. The user cannot edit or delete this view because this would affect the three applications. The *SUPPLIERS* view on the other hand is not

being referenced by any applications and so can be edited or deleted.

This functionality allows RVM to be a powerful tool for developer.

## **2.5 Security**

The RVM is intended to be a multiple user product. Many programmers can view the relationship between applications and remote views simultaneously. RVM does not allow the user to edit or delete remote views that are used in multiple applications.

## **2.6 Exception Handling**

When detected, error messages are displayed using Visual FoxPro Message boxes.

## **2.7 Conclusion.**

In conclusion, the Remote View Manager, RVM was developed according to the software requirement specification. RVM provides an easy to use tool that allows programmers to quickly cross-reference remote views and applications.

The flexibility of the design of RVM allows easy modification for RVM to search for other references in Visual FoxPro applications. The structure of RVM allows it to be integrated to other existing software development tools.

## **2.8 Future Modification and Enhancement**

1. To facilitate the optimization of Remote Views, RVM can be expanded to show the SQL statements that are used for the Remote Views.
2. RVM can be expanded to show the relationship between the application, Remote Views and database tables. This will ensure a smooth transition when database tables are modified.

3. RVM should be integrated with the application that compiles the projects to executables. This will ensure that the relationship between applications and Remote Views are always current

### 3. CHAPTER 3 SOFTWARE DESCRIPTION

#### 3.1 General Layout(Main Form)

The RVM has one main form that consists of several command buttons, a ComboBox, two containers and 3 menu items. The main container has several command buttons, two radio buttons and two grids. An example is shown below.

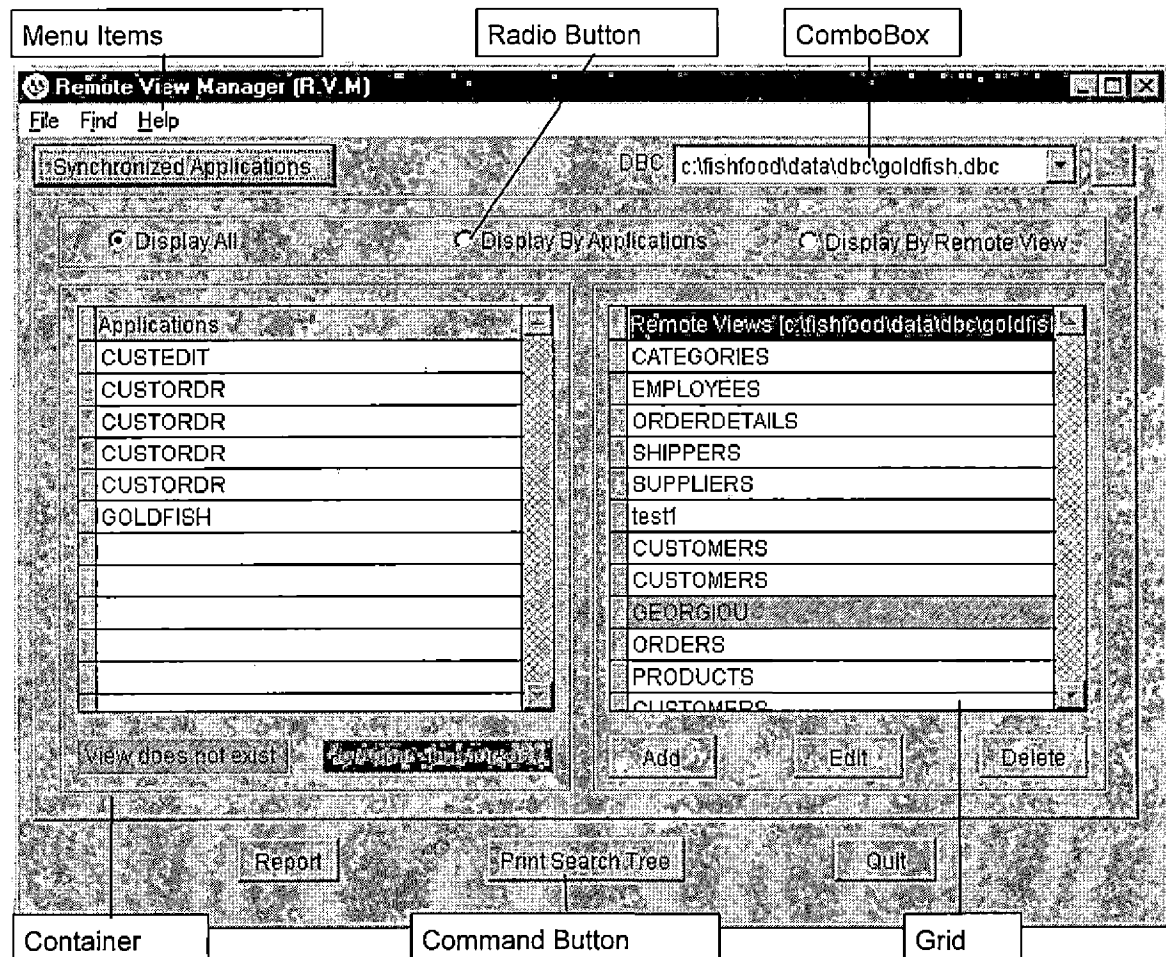


Figure 11. Main Form

The second container is hidden and contains a listbox, a textbox and several command buttons. The visibility of the second container is controlled by the second grid on the first container, along with the Add and Edit command buttons (see Figure 11). The second container will be discussed in further detail in section 3.1.6. A detailed explanation of each element on the form will be discussed in the sections that follow.

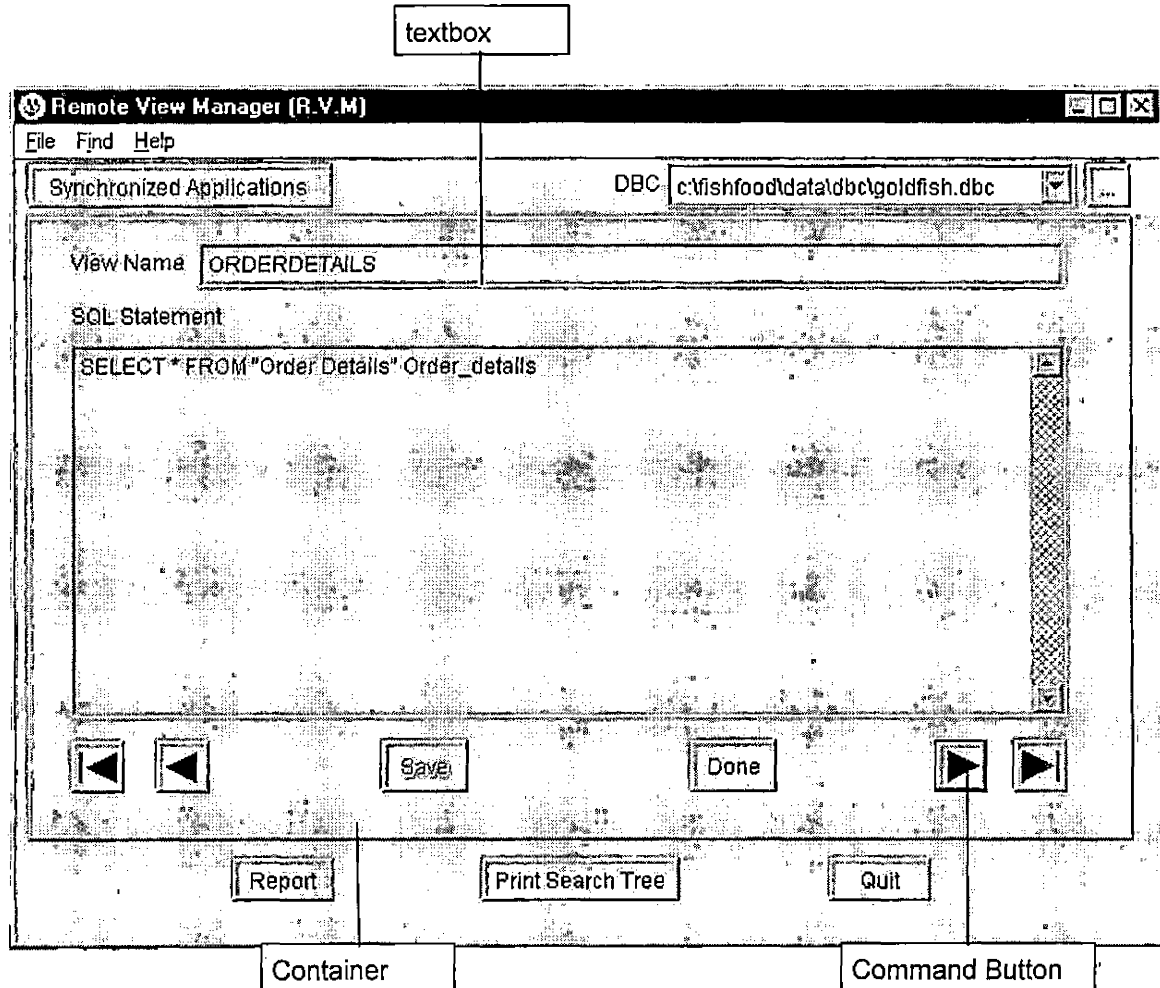
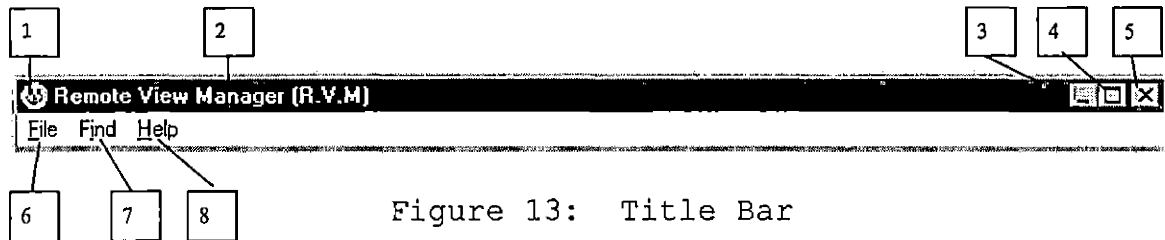
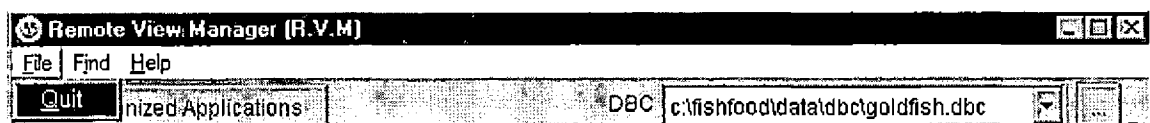


Figure 12: Main Form, Second Container.

### 3.1.1 Title Bar and Menu Items.



The Title Bar consists of five major items (Figure 13). The RVM icon/logo is located in the leftmost corner (item 1) followed by the Title of the application, "Remote View Manager (R.V.M.)" (item 2). The Minimize (item 3), Maximize (item 4) and Close-Forced Exit buttons (item 5) are located in the rightmost corner of the title bar. The Menu Items are as follows: item 6 is the File Menu Item, item 7 is the Find Menu Item and item 8 is the About Menu Item. The resulting actions of user input are summarized in Table 5.



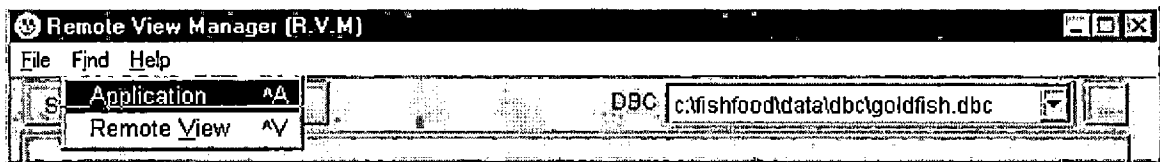


Figure 15: Find Drop Down Menu Item

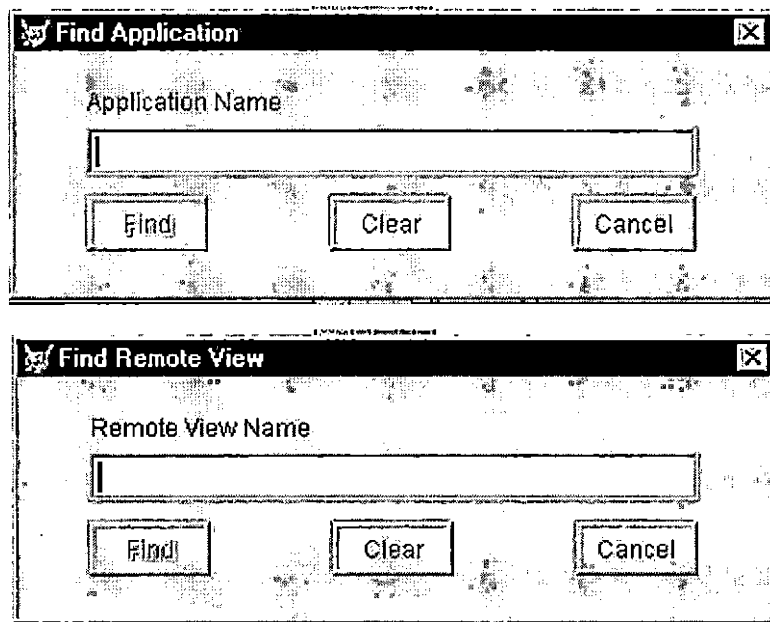


Figure 16: Find Application and Find Remote View Form.



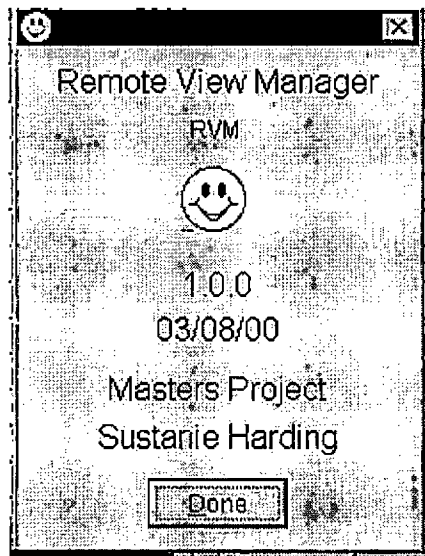


Figure 17: Help Drop Down Menu Item

User Input	Result
Click the Left Mouse Button on the RVM Icon	A standard windows dropdown menu with the following choices: Restore, Move, Size, Minimize, Maximize, Close and Next.
Click the Left Mouse Button on the Maximize Button	Maximizes the window.
Click the Left Mouse Button on the Minimize Button	Minimizes the window.
Click the Left Mouse Button on the Close Button	Closes the window.

Click the Left Mouse Button on the File Menu Item	A menu drops down with a menu choice of Quit. (See Figure 14)
Click the Left Mouse Button on the Find Menu Item	A menu drops down with two menu choices: Applications and Remote View. (See Figure 15 and 13). If the user chooses Applications, the Application form pops up and the user can enter the name of the view to be found in the Applications Grid. The same process is repeated for remote views.
Click the Left Mouse Button on the Help Menu Item	A menu drops down with a menu choice of About (See Figure 17)
	Closes the window.

Table 5: Title Bar Buttons

### 3.1.2 Synchronized Applications

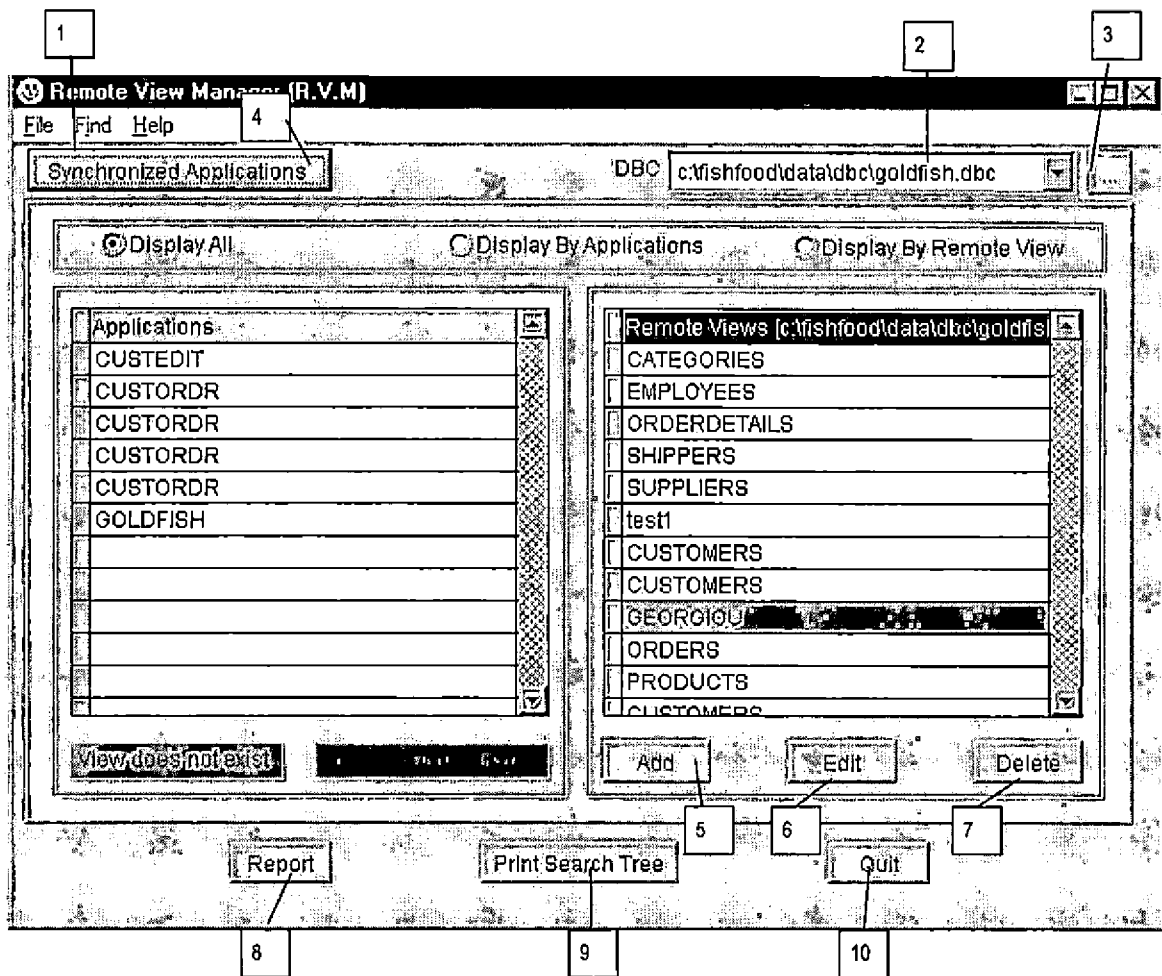


Figure 18: Object on the Main Form

The 'Synchronized Applications' Command Button (Figure 18, item 1) calls one of the main procedures in the application. It executes the routines that recursively search through the Visual FoxPro Applications, collecting references to remote views. Before starting the recursive

search, the routine first asks the user to select the root directory from which the search will be initiated. The resulting actions of user input on this command button are summarized in Table 6.

User Input	Result
Click the Left Mouse Button on the 'Synchronized Applications' Command Button	Calls the directory selector form (Figure 19, Section 3.1.2.1. Directory Selector). Recursively searches through the applications collecting references to remote views. Inserts these references into the database to illustrate the relationship between the remote views and the applications.

Table 6: Synchronized Applications

### 3.1.2.1 Directory Selector

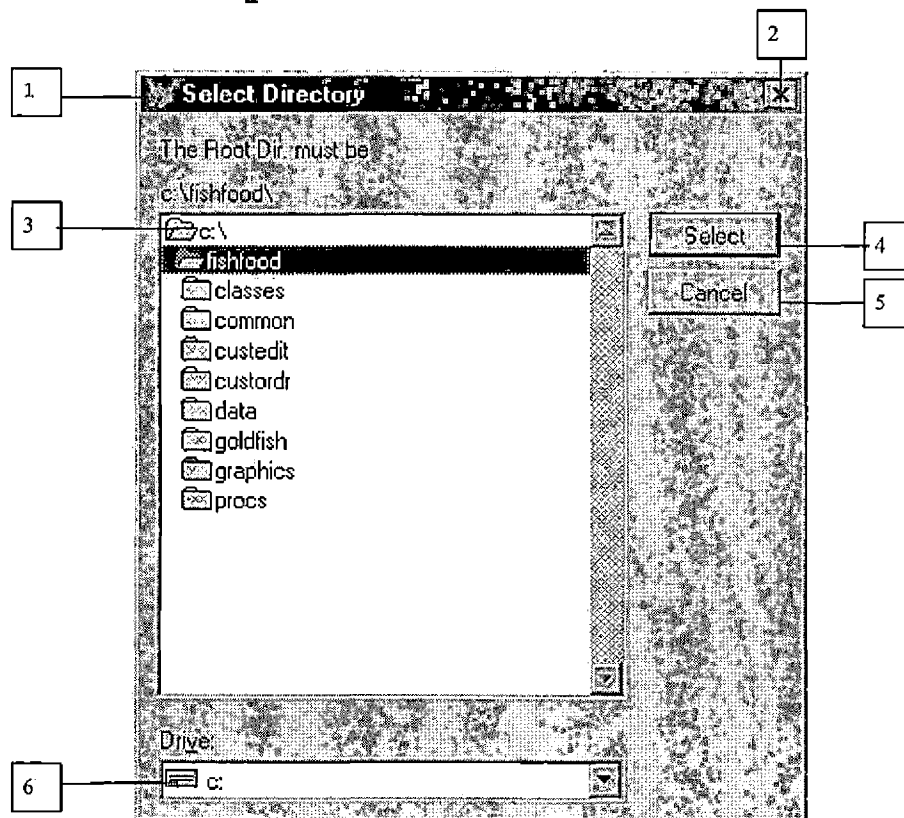


Figure 19: Directory Selector

The Directory Selector is a standard Visual FoxPro provided form. This form allows the user to choose the root of the directory structure that will be traversed in the application. The Directory Selector form consists of six major items (Figure 19). The items on this form will be discussed in sections '3.1.2.1.1 Visual FoxPro Icon and Title Bar' through '3.1.2.1.6 Drive'.

### 3.1.2.1.1 Visual FoxPro Fox Icon and Title Bar

The Visual FoxPro Fox icon and Title (Figure 19, item 1) are supplied by Visual FoxPro. The resulting actions of user input are summarized in Table 7.

User Input	Result
Click the Left Mouse Button on the Visual FoxPro Fox Icon	Invokes a drop down menu with two choices: 'Move' and 'Close'. Selecting 'Move' will allow the user to reposition the form using the arrow keys. Selecting 'Close' will close the form. This is equivalent to canceling the form.
Click the Left Mouse Button on the Title	No effect

Table 7: Visual FoxPro Fox Icon and Title Bar

### 3.1.2.1.2 Close Button

The 'Close' button (Figure 19, item 2) is also supplied by Visual FoxPro. The resulting actions of user input are summarized in Table 8.

User Input	Result
Click the Left Mouse Button on the 'Close' button	Closes the form. This is equivalent to canceling the form.

Table 8: Close Button

### 3.1.2.1.3 Directory Tree Structure

The directory structure (Figure 19, item 3) illustrates the file structure on the machine in a tree representation. The selector defaults to the present directory. However, the user is able to select the directory they want by physically traversing the tree. The resulting actions of user input are summarized in Table 9.

User Input	Result
Double Click the Left Mouse Button on a Folder.	Opens the folder and displays any directory that it contains.

Table 9: Directory Structure

#### 3.1.2.1.4 Select Command Button

The 'Select' Command Button (item 4) allows the user to select the directory that they wish to start the recursive search in. The resulting actions of user input are summarized in Table 10.

User Input	Result
Click the Left Mouse Button on the Select Command Button.	Returns to the calling program the name of the directory that is selected in the directory tree structure, Figure 19, item 3.

Table 10: Select Command Button



#### 3.1.2.1.5 Cancel Command Button

The 'Cancel' Command Button (Figure 19, items 5) allows the user to cancel the form without any selection. The resulting actions of user input are summarized in Table 11.

User Input	Result
Click the Left Mouse Button on the Cancel Command Button.	Cancels any selection made in the directory tree structure, Figure 19, item 3. Returns an empty string to the calling program.

Table 11: Cancel Command Button

#### 3.1.2.1.6 Drive

The 'Drive' ComboBox (Figure 19, item 6) allows the user to select the drive that will be used for the directory tree structure. The resulting actions of user input are summarized in Table 12.

User Input	Result
Click the Left Mouse Button on the Drive ComboBox.	A drop down list allows the user to choose which drive the directory structure will be using.

Table 12: Drive ComboBox

### 3.1.3 Database Container ComboBox (DBC)

The ComboBox (Figure 14, item 2) is a scrollable list containing the most recently selected search paths for the DBC. This is the path that the RVM will be cross-referencing with the application. The resulting actions of user input are summarized in Table 10.

User Input	Result
Click the Left Mouse Button on the DBC ComboBox	A scrollable list appears under the ComboBox and the user can select a DBC path.

Table 10: DBC Combobox

### 3.1.4 File List Command Button

The 'File List' Command Button (Figure 14, item 3) allows the user to select a new DBC. It calls the file selector form (Figure 20, Section 3.1.4.1 File Selector). The resulting actions of user input are summarized in Table 14.

User Input	Result
Click the Left Mouse Button on the File List Command Button	Calls the file selector form, Figure 20, Section 3.1.4.1 File Selector.

Table 14: File List Command Button

### 3.1.4.1 File Selector

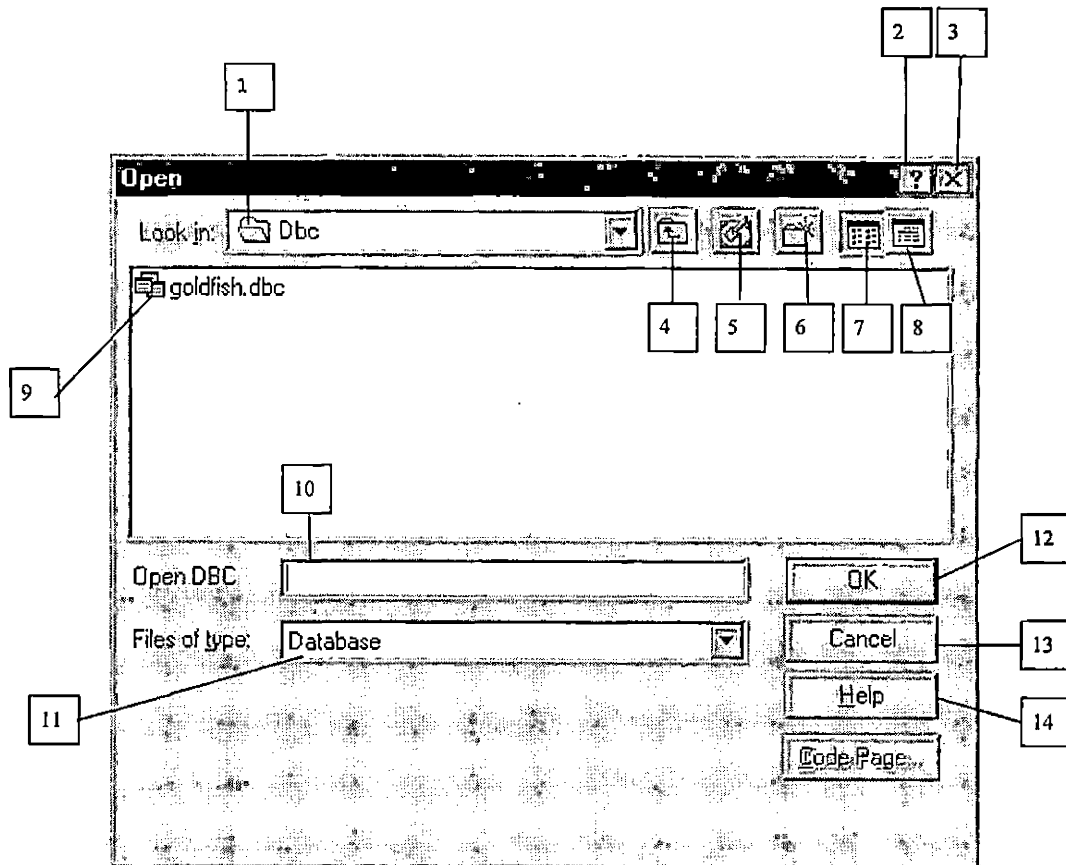


Figure 20: File Selector

The File Selector is a standard Visual FoxPro dialog box from which a file is chosen. This form allows the user to search through the directory structure to locate the required file. The File Selector form consists of fourteen major items (Figure 20). The items on this form will be discussed in sections 3.1.4.1.1 Directory Selector through 3.1.4.1.14 Drive.

#### 3.1.4.1.1 Directory Selector ComboBox

The Directory Selector ComboBox (Figure 20, item 1) is the standard Windows directory selector. It lists the available folders. The resulting actions of user input are summarized in Table 15.

User Input	Result
Click the Left Mouse Button on the Directory Selector ComboBox	Invokes a drop down tree that illustrates where the current folder is located in the hierarchy of folders.

Table 15: Directory Selector ComboBox

#### 3.1.4.1.2 Question Mark Button

The 'Question Mark' button in the title bar (Figure 20, item 2) is the windows provided quick help. This will display a pop-up help screen when the item is clicked after clicking on the question mark button. The resulting actions of user input are summarized in Table 16.

User Input	Result
Click the Left Mouse Button on the Question Mark Button.	The cursor changes to a question mark. Moving the question mark cursor over an item in the form will invoke a help pop-up Window (if available).

Table 16: Question Mark Button

#### 3.1.4.1.3 Close Button

The 'Close' button in the title bar (Figure 20, item 3) allows the user to close the form. The resulting actions of user input are summarized in Table 17.

User Input	Result
Click the Left Mouse Button on the Close Button	Closes the form. This is equivalent to canceling the form.

Table 17: Close Button

#### 3.1.4.1.4 Up one level Command Button

The 'Up one level' command button (Figure 20, Item 4) allow the user to select the next level up in the directory structure. The resulting action of user input is summarized in Table 18.

User Input	Result
Click the Left Mouse Button on the 'Up One Level' folder button	Moves the directory selector, (Figure 20, Item 1) up one level in the directory hierarchy.

Table 18: Up One Level

#### 3.1.4.1.5 View Desktop Button

Item 5 is the 'View Desktop' command button, it allows the user to select the desktop view of the directory structure. The resulting action of user input is summarized in Table 19.

User Input	Result
Click the Left Mouse Button on the 'View Desktop' Button	Moves the directory selector (Figure 20, Item 1) to the desktop view of the directory hierarchy.

Table 19: View Desktop Button

#### 3.1.4.1.6 Create New Folder Button

The 'Create New Folder' Button (Figure 20, item 6) allows the user to create a new folder. This button does not apply in this application as the user is searching for an existing DBC, not creating a DBC. The resulting action of user input is summarized in Table 20.

User Input	Result
Click the Left Mouse Button on the 'Create New Folder' Button	A new folder appears with a temporary name. Type a name for the folder, and then press ENTER.

Table 20: Create New Folder Button



#### 3.1.4.1.7 List View Button

The 'List View' Button (Figure 20, item 7) allows the user to change the view of the folder/file list, (Figure 20, item 9). The resulting action of user input is summarized in Table 21.

User Input	Result
Click the Left Mouse Button on the 'List View' Button	The folder/file list (Figure 20, item 9) shows only file names of all files and folders on the selected directory.

Table 21: List View Button

#### 3.1.4.1.8 Detail View Button

The 'Detail View' Button (Figure 20, item 8) allows the user to change the view of the folder/file list to a detail list, (Figure 20, item 9). The resulting action of user input is summarized in Table 22.

User Input	Result
Click the Left Mouse Button on the 'Detail View' Button	The folder/file list (Figure 20, item 9) shows a detail list of all files and folders on the selected directory.

Table 22: Detail View Button

#### 3.1.4.1.9 Folder/File List

The 'Folder File' List (Figure 20, item 9) lists the folders and files in the selected location. The resulting action of user input is summarized in Table 23.

User Input	Result
Click the Left Mouse Button on the File in the list	The file changes color to blue. The file name is copied into the Open DBC text box item 10 on Figure 20.

Double Click the Left Mouse Button on a Folder in the list	Opens the folder
Double Click the Left Mouse Button on a File in the list	Selects the file. The file name is then returned to the calling program.

Table 23: Folder File List

#### 3.1.4.1.10 Open DBC

The 'Open DBC' textbox (Figure 20, item 10) allows the user to type the name of the DBC they wish to open. The resulting action of user input is summarized in Table 24.

User Input	Result
Type in the Textbox	The typed letters appear in the Textbox. The Textbox works concurrently with the open button (Figure 20, item 12).

Table 24: Open DBC

#### 3.1.4.1.11 File Types ComboBox

The 'File Types' ComboBox (Figure 20, item 11) allows the user to select the file types that will be shown in the folder/file list (Figure 20, item 9). Visual FoxPro gives the user two choices: 'Databases' which is the default file type or 'All Files'. The resulting action of user input is summarized in Table 25.

User Input	Result
Click the Left Mouse Button in the 'File Types' ComboBox	Invokes a drop down list that allows the user to select the file types that will be shown in the Folder/File list.

Table 25: File Types ComboBox

#### 3.1.4.1.12 OK Command Button

The 'OK' Command Button (Figure 20, item 12) opens the DBC. The resulting action of user input is summarized in Table 26.

User Input	Result
Click the Left Mouse Button on the OK Command Button	Opens the DBC Selected from the Folder/File list (Figure 20, item 9) or opens the DBC named in Figure 20, item 10.

Table 26: OK Command Button

#### 3.1.4.1.13 Cancel Command Button

The 'Cancel' Command Button (Figure 20, item 13) closes the form without making a selection. The form returns an empty string to the calling program. The resulting action of user input is summarized in Table 27.

User Input	Result
Click the Left Mouse Button on the Cancel Command Button	Cancels the form and returns an empty string to the calling program.

Table 27: Cancel Command Button

#### 3.1.4.1.14 Help Command Button

The 'Help' Command Button calls the Windows help files if they are installed. The resulting action of user input is summarized in Table 28.

User Input	Result
Click the Left Mouse Button on the Command Button	Invokes the Windows help pop-up form.

Table 28: Help Command Button

### 3.1.5 Main Container

The main container (Figure 12, item 4) is one of two containers on the form. This container can be seen in detail in Figure 21. The main container consists of two radio buttons: item 1 and 2, two grids: item 3 and 4, and a command button: item 5 in Figure 21. The objects on the container will be discussed in sections 3.1.5.1 through 3.1.5.5.

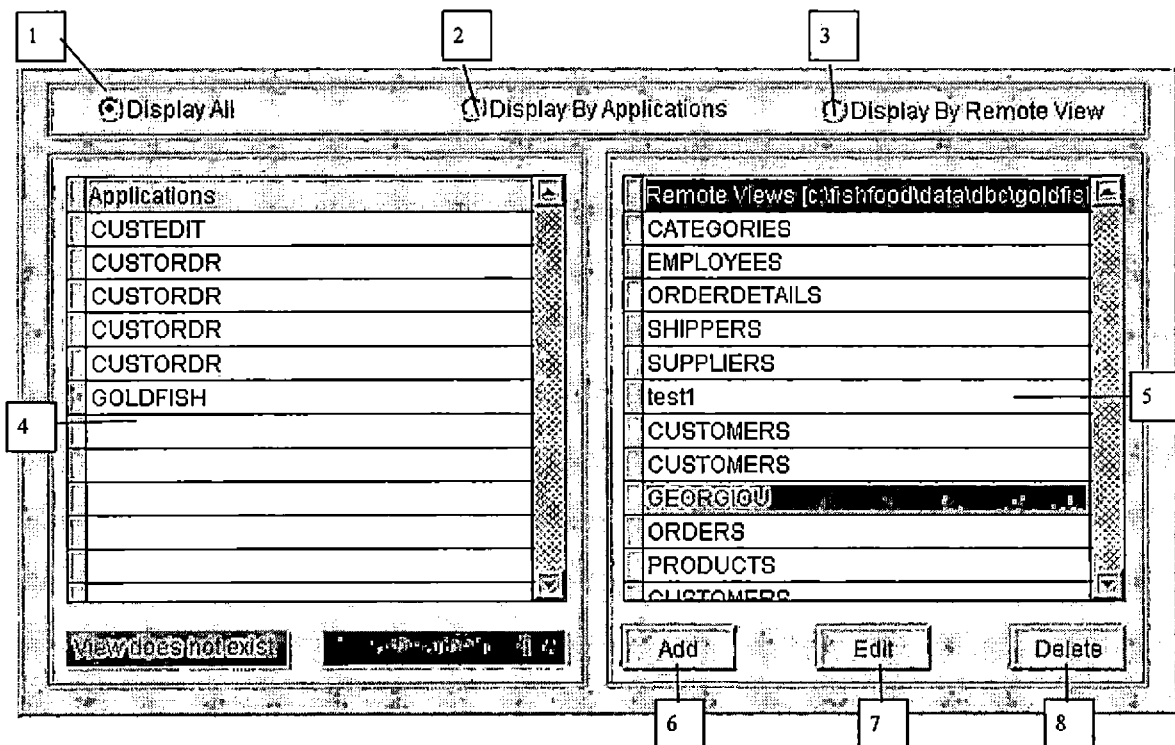


Figure 21: Main Container.

### 3.1.5.1 'Display All' Radio Button

The 'Display All' Radio button (Figure 21, item 1) works concurrently with the 'Display By Application' and 'Display By Remote View' radio buttons and the two grids on the form. Selecting the 'Display All' radio button has the resulting action summarized in Table 29.

User Input	Result
Left Click on the 'Display All' Radio Button	Toggles value from 'OFF' to 'ON' and vice versa. If 'Display All' is 'ON' then 'Display by Applications' and 'Display by Remote View' will be 'OFF'. It resets Applications Grid and Remote View grids.

Table 29: 'Display All' Radio Button



### 3.1.5.2 'Display By Applications' Radio Button

The 'Display By Application' radio button (Figure 21, item 2) works concurrently with the 'Display By Remote View' radio button and the two grids on the form.

Selecting the 'Display By Application' radio button has the resulting action summarized in Table 30.

User Input	Result
Left Click on the 'Display by Applications' Radio Button	Toggles value from 'OFF' to 'ON' and vice versa. If 'Display by Applications' is 'ON' then 'Display by Remote View' will be 'OFF'. Its effect on the grids will be discussed in section 1.4.6.4 Applications Grid.

Table 30: 'Display By Applications' Radio Button

### 3.1.5.3 'Display By Remote View' Radio Button

The 'Display by Remote View' radio button (Figure 21, item 3) works concurrently with the 'Display by Applications' radio button and the two grids on the form. Selecting the 'Display by Remote View' radio button has the resulting action summarized in Table 31.

User Input	Result
Left Click on the 'Display by Remote View' Radio Button	Toggles value from 'OFF' to 'ON' and vice versa. If 'Display by Remote View' is 'ON' then 'Display by Applications' will be 'OFF'. Its effect on the grids will be discussed in section 1.4.6.5 Remote Views Grid.

Table 31: 'Display by Remote View' Radio Button

#### 3.1.5.4 Applications Grid

The result set from the synchronized applications with a remote views algorithm is displayed in the Applications grid. This grid (Figure 21, item 3) contains

a list of all applications that were found under the root directory that has some reference to a remote view. This grid is populated by the table "viewappl" whose data structure is shown in Table 31.

	Fieldname	Type
PK <sup>7</sup>	Viewuniq	Int
U1 <sup>8</sup>	ViewName	Varchar(32)
U1	ApplName	Varchar(32)

Table 32. Data structure of the "viewappl" table

The 'Applications' grid works concurrently with the 'Display by Applications' radio button, 'Display by Remote View' radio button and the 'Remote Views' grid. The resulting action of user input is summarized in Table 33.

---

<sup>7</sup> Primary Key  
<sup>8</sup> Unique Index

User Input	Result
Click the Left Mouse Button on a cell in the applications grid (i.e. a application name)	If "Display By Applications" is "ON", selecting an application name in the 'Applications' grid causes the 'Remote View' grid to refresh and reflect all views that are used in that application. If the "Display By Applications" is "OFF", selecting an application name in the 'Applications' grid has no effect.

Table 33: The 'Applications' grid

#### 3.1.5.5 Remote Views Grid

The 'Remote View' Grid lists all remote views in the DBC. This grid (Figure 21, item 4) is populated by the DBC that was selected in the ComboBox (Figure 14, item 2) or by the 'File list' command button (Figure 14,item 3).

The DBC is a database file that has the following data structure (Table 34).

	Fieldname	Type
PK	objectid	Int
	parentid	Int
	Objecttype	Varchar(32)
	ObjectName	Varchar(32)
	property	Memo
	Code	Memo
	user	Memo
	Riinfo	Memo

Table 34. Data structure of the DBC.

The 'Remote Views' grid works concurrently with the 'Display by Remote View' radio button, 'Display by Applications' radio button and the 'Applications' grid. The resulting action of user input is summarized in Table 35.

User Input	Result
Click the Left Mouse Button on a cell in the grid (i.e. a remote view name)	If "Display By Remote View" is "ON", selecting a remote view on the 'Remote Views' grid causes the 'Applications' grid to refresh and reflect all applications that use that view. If the "Display By Remote View" is "OFF", selecting a remote view in the 'Remote View' grid has no effect.
Click the Right Mouse Button on a cell in the grid (i.e. a remote view name)	The main container is made invisible and the second container becomes visible, Figure 5. The second container is moved to the foreground and the main container is moved to the background. This will be discussed in detail in Section 1.4.7 Secondary Container.

Table 35: The 'Remote Views' grid

#### 3.1.5.6 Add Command Button

The Add command button (Figure 21, item 5) allows the user to add remote views from the DBC. The Add operation uses the secondary container described in section 1.4.7 Secondary Container. The resulting action of user input is summarized in Table 36.

User Input	Result
Click the Left Mouse Button on the Add Button	The main container is made invisible and the second container becomes visible, Figure 5. The second container is moved to the foreground and the main container is moved to the background. The fields in the secondary container are enabled allowing the user to add the pertinent information needed to add the remote view. This will be discussed in detail in Section 1.4.7 Secondary Container.

Table 36: The Add Command Button



### 3.1.5.7 Delete Command Button

The Delete command button (Figure 21, item 6) allows the user to delete remote views from the DBC. It should be noted that deletion should be done with caution, as there is no way to recover a deleted record. The resulting action of user input is summarized in Table 37.

User Input	Result
Click the Left Mouse Button on the Delete Button	Deletes the remote view selected in the 'Remote Views' grid with the necessary warning messages.

Table 37: The Delete Command Button

### 3.1.6 Secondary Container

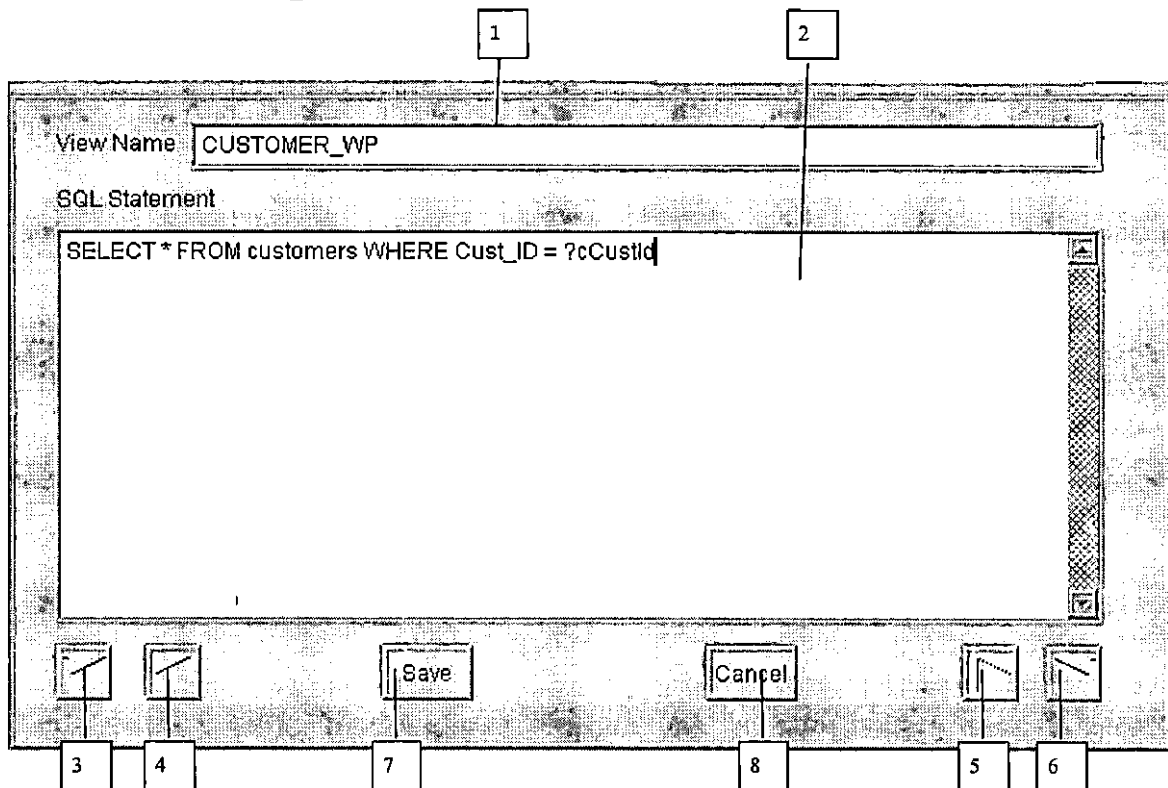


Figure 22: Secondary Container

This container allows the user to specify new views and shows the user the fields being selected in a remote view. The buttons are enabled as needed, depending on the mode that the secondary container is being operated. When adding a new remote view, the 'VCR' (Figure 22, item 3,4,5 and 6) buttons are disabled. The Save button (Figure 22, item 7) is only enabled when all fields have been entered.

In cases where the view is being displayed, the 'VCR' (Figure 22, item 3,4,5 and 6) buttons are enabled. The Save button (Figure 22, item 7) is disabled and the Cancel Button has the Caption of 'Done'. The buttons on the main form are all disabled except for the Quit Button and have no effect on this container. The objects in this container are discussed in detail in the sections that follows.

#### **3.1.6.1 View Name Textbox**

The 'View-Name' textbox (Figure 22, item 1) is a read only control when viewing a remote view. In this case, it shows the name of the view whose SQL Statement is being displayed in Figure 22, item 2. The user has no direct effect on this control in this mode. It is affected indirectly by user input on items 3,4,5, and 6 in Figure 22. In cases where the user is adding a view, the field is editable and the user can type in the name of the new view. The program then checks for duplicate view names and issues the relevant error message. The resulting action of user input is summarized in Table 38.

User Input	Result
Type in the textbox.	If the 'SQL Statement' Listbox (Figure 22, item 2) is not empty, the Save Button (Figure 22, item 7) is enabled.

Table 38: The 'View Name' textbox

#### 3.1.6.2 SQL Statement Listbox

The 'SQL Statement' Listbox (Figure 22, item 2) is also a read only control when viewing a remote view. It shows the SQL Statement that is being executed by the DBC. The user has no direct effect on the control. It is affected indirectly by user input on items 3,4,5, and 6 in Figure 22. In cases where the user is adding a view, the field is editable and the user can type in the SQL Statement that will be executed when the view is built. The resulting action of user input is summarized in Table 39.

User Input	Result
Type in the textbox.	If 'View Name' textbox (Figure 22, item 1) is not empty, the Save Button (Figure 22, item 7) is enabled.

Table 39: The 'SQL Statement' Listbox

#### 3.1.6.3 First VCR Command Button

The 'First VCR' command button (Figure 22, Item 3) moves the record pointer in the Remote View grid to the first record in the grid. The resulting action of user input is summarized in Table 40.

User Input	Result
Click the Left Mouse Button on the First VCR Button	Moves the record pointer to the first record in the table. The ViewName textbox refreshes to show the name of the first view in the table. The SQL Statement Listbox refreshes to show the SQL statement that is used by that view.

Table 40: First VCR Button

#### 3.1.6.4 Previous VCR Button

The 'Previous VCR' command button (Figure 22, Item 4) moves the record pointer in the Remote View grid to the previous record, relative to the present record in the grid. The resulting action of user input is summarized in Table 41.

User Input	Result
Click the Left Mouse Button on the Previous VCR Button	Moves the record pointer to the previous record in the table. The ViewName textbox refreshes to show the name of the previous view in the table. The SQL Statement Listbox refreshes to show the SQL statement that is used by that view.

Table 41: Previous VCR Button

#### 3.1.6.5 Next VCR Button

The 'Next VCR' command button (Figure 22, Item 5) moves the record pointer in the Remote View grid to the next record, relative to the present record in the grid. The resulting action of user input is summarized in Table 42.

User Input	Result
Click the Left Mouse Button on the Next VCR Button	Moves the record pointer to the next record in the table. The ViewName textbox refreshes to show the name of the next view in the table. The SQL Statement Listbox refreshes to show the SQL statement that is used by that view.

Table 42: Next VCR Button

#### 3.1.6.6 Last VCR Button

The 'Last VCR' command button (Figure 22, Item 6) moves the record pointer in the Remote View grid to the last record in the grid. The resulting action of user input is summarized in Table 43.



User Input	Result
Click the Left Mouse Button on the Last VCR Button	Moves the record pointer to the last record in the table. The ViewName textbox refreshes to show the name of the last view in the table. The SQL Statement Listbox refreshes to show the SQL statement that is used by that view.

Table 43: Last VCR Button

#### 3.1.6.7 Save Button

The 'Save VCR' command button (Figure 22, Item 7) saves the new view and returns control to the Main Container. The resulting action of user input is summarized in Table 44.

User Input	Result
Click the Left Mouse Button on the Done Button	The new view is added to the DBC. The secondary container is made invisible and the main container becomes visible. The main container is moved to the foreground and the secondary container is moved to the background.

Table 44: Save Button

#### 3.1.6.8 Done/Cancel Button

The 'Done/Cancel VCR' command button (Figure 22, Item 8) returns control to the main container. This button caption is dependent on the mode. If the user is displaying the existing view, the caption is 'Done' and if the user is adding a new view, the caption is 'Cancel'. The resulting action of user input is summarized in Table 45.

User Input	Result
Click the Left Mouse Button on the Done Button	The secondary container is made invisible and the main container becomes visible.  The main container is moved to the foreground and the secondary container is moved to the background.

Table 45: Done Button

### 3.1.7 Report Command Button

The 'Print' Command button prints two reports. The resulting action of user input is summarized in Table 46.

User Input	Result
Click the Left Mouse Button on the Print Button	Prints the mapping based on the radio button in the Main Container, Figure 21, items 1 & 2. If the view 'Display By Application' is "ON", then the report generated by the print will show the applications and the views that are used in the applications. IF the 'Display By Remote View' is "ON", then the report generated by the print will show the Remote Views and in which application they are used.

Table 46: Report Button

### 3.1.8 Print Search Tree Command Button

The 'Print Search Tree' Command button prints the result set from the recursive search. It prints a report that details the search path that was used in collecting the report view references for the applications. The resulting action of user input is summarized in Table 47.

User Input	Result
Click the Left Mouse Button on the Print Search Tree Button	Prints the search path as a tree structure.

Table 47: Print Search Tree Button

### 3.1.9 Quit Command Button

The 'Quit' button closes the application. The resulting action of user input is summarized in Table 48.

User Input	Result
Click the Left Mouse Button on the Quit Button	Quits the application.

Table 48: Quit Button

#### 4. CHAPTER 4 UNIT TESTING

The program was tested by setting up test scenario where the references to remote views were placed in Visual FoxPro objects.

Project	ViewName	View found
Test1	Remote View on Form	yes
Test2	Remote View in Class	yes
Test3	Remote View on PageFrame	yes
Test4	Remote View on Page	yes
Test5	Remote View in Grid	yes
Test6	Remote View in Column	yes
Test7	Remote View on ToolBar	yes
Test8	Remote View on Option Button Group	yes
Test9	Remote View on Command Button Group	yes
Test10	Remote View in Check Box	yes
Test11	Remote View in Combo Box	yes
Test12	Remote View in Command Button	yes

Test13	Remote View in Edit Box	yes
Test14	Remote View in List Box	yes
Test15	Remote View in Text Box	yes
Test16	Remote View in Container	yes
Test17	Remote View in Spinner	yes
Test18	Remote View in Timer	yes

Table 49: Test Result



## **APPENDIX A: GLOSSARY OF TERMS**

**DBC - Database Container.** The Visual FoxPro container tool that manages tables, queries, forms, fields, persistent relations and index tags.

**SQL - Structure Query Language.** A standardized language used to manipulate data stored in tables.

**View -** A definition of a virtual table derived from one or more tables or other views.

**ODBC (Open Database Connectivity) -** Standard protocol for database servers.

## APPENDIX B SOURCE CODE OF RVM

### Dataenvironment

#### vfarmform

PROCEDURE menuinit

    cMenuName = "menu1"

    cFontname='Arial'

    nFontSize=10

    cFontstyle='N'

    DEFINE MENU (cMenuName) BAR AT LINE 0 FONT cFontname, nFontSize STYLE cFontstyle COLOR  
    SCHEME 4;

        IN WINDOW (Thisform.Name)

    DEFINE PAD filepad OF (cMenuname) PROMPT '\<File' KEY ALT+F COLOR SCHEME 3

    DEFINE PAD findpad OF (cMenuname) PROMPT 'F\<ind' KEY ALT+I COLOR SCHEME 3

    DEFINE PAD helppad OF (cMenuname) PROMPT '\<Help' KEY ALT+H COLOR SCHEME 3

    DEFINE POPUP (cMenuname+'file') FONT cFontname, nFontSize STYLE cFontstyle MARGIN RELATIVE

    DEFINE BAR 1 OF (cMenuname+'file') PROMPT '\<Quit' COLOR SCHEME 3

    ON SELECTION BAR 1 OF (cMenuname+'file') \_SCREEN.ActiveForm.QueryUnload()

    DEFINE POPUP (cMenuname+'find') FONT cFontname, nFontSize STYLE cFontstyle MARGIN RELATIVE

    \*! DEFINE BAR 1 OF (cMenuname+'find') PROMPT '\<Application' KEY CTRL+A, '^A'

    \*! ON SELECTION BAR 1 OF (cMenuname+'find') DO FORM RVMFind WITH 0 TO ntemp

    \*! DEFINE BAR 2 OF (cMenuname+'find') PROMPT 'Remote \<View' KEY CTRL+V, '^V'

    \*! ON SELECTION BAR 2 OF (cMenuname+'find') DO FORM RVMFind WITH 1 TO ntemp

    DEFINE BAR 1 OF (cMenuname+'find') PROMPT '\<Application' KEY CTRL+A, '^A'

    ON SELECTION BAR 1 OF (cMenuname+'find') \_SCREEN.ActiveForm.Find(0)

    DEFINE BAR 2 OF (cMenuname+'find') PROMPT 'Remote \<View' KEY CTRL+V, '^V'

    ON SELECTION BAR 2 OF (cMenuname+'find') \_SCREEN.ActiveForm.Find(1)

    DEFINE POPUP (cMenuname+'help') FONT cFontname, nFontSize STYLE cFontstyle MARGIN RELATIVE

    DEFINE BAR 1 OF (cMenuname+'help') PROMPT 'About'

    ON SELECTION BAR 1 OF (cMenuname+'help') DO FORM About

    ON PAD filepad OF (cMenuname) ACTIVATE POPUP (cMenuname+'file')

    ON PAD findpad OF (cMenuname) ACTIVATE POPUP (cMenuname+'find')

    ON PAD helppad OF (cMenuname) ACTIVATE POPUP (cMenuname+'help')

    ACTIVATE MENU (cMenuname) NOWAIT

ENDPROC

PROCEDURE setdbcpath

```

LPARAMETERS cPath, oObj

LOCAL nl
**
*- Set the dbc path.
**

FOR nl = 1 TO oObj.Listcount
    IF ALLTRIM(cPath) == ALLTRIM(oObj.List(nl))
        oObj.Listindex = nl
        RETURN .T.
    ENDIF
ENDFOR

oObj.Listindex = 0

RETURN .F.
ENDPROC

PROCEDURE adddbcpath
LPARAMETERS cPath, oObj

LOCAL nl, nRec, tPathuniq
**
*- Check if the path already exist in the combo list. If it is already
*- in the list select the item and return. Otherwise add the path to the
*- dbc path table and set the list index to the item.
**

FOR nl = 1 TO oObj.Listcount
    IF ALLTRIM(cPath) == ALLTRIM(oObj.List(nl))
        oObj.Listindex = nl
        RETURN
    ENDIF
ENDFOR

nRec = 0
tPathuniq = DATETIME()

** IF USED('dbcpath')
**     SELECT dbcpath
** ELSE
**     USE vfarm\dbcpath
** ENDIF

SELECT dbcpath
SEEK cPath ORDER TAG PName IN dbcpath
IF !FOUND('dbcpath')
    INSERT INTO dbcpath VALUES(tPathuniq, cPath)
ENDIF

oObj.Requery()

```

```

oObj.Listindex = 1

RETURN
ENDPROC

PROCEDURE opendbc
LOCAL nAnswer, nCnt, cStr, nI, nJ, cWait
LOCAL cModified, dModified, nHandle, cName, cSQL

ThisForm.OpeningData = .T.
ThisForm.OpenDBCErrors = .F.

cWait = "Opening the DBC, Please Wait..."
WAIT cWait WINDOW AT 20,30 NOCLEAR NOWAIT

ThisForm.VCntrDBC.VGridDBC.RecordSource = 'CurDummyDBC'

ThisForm.CloseDBC()

ThisForm.cCreateTo = "
IF ThisForm.vcboDBC.Listindex > 0
    ThisForm.cCreateTo = LOWER(ALLTRIM(ThisForm.vcboDBC.List(ThisForm.vcboDBC.Listindex)))
ENDIF

IF USED('curDBviews')
    SELECT curDBviews
    USE
ENDIF
LOCAL ARRAY aDBCViews(1)
CREATE CURSOR CurDBViews (ViewName C(32) NULL, cSQLStatement C(250) NULL, ISelected L,
ICurrentRec L, IInAppI L NULL, IInDbc L NULL, BackColor I NULL)
INDEX ON viewname TAG viewnmt

IF (IEMPTY(ThisForm.cCreateTo))AND ThisForm.OpenDBCErrors == .F.
    IF !FILE(ThisForm.cCreateTo)
        =MESSAGEBOX("The DBC file " + ThisForm.cCreateTo + " does not contain any records")
    ELSE
        OPEN DATABASE (ThisForm.cCreateTo) EXCLUSIVE
        SET DATABASE TO (ThisForm.cCreateTo)
        WAIT CLEAR
        ThisForm.DBCOpen = .T.
        WAIT cWait WINDOW AT 20,30 NOCLEAR NOWAIT
        nCnt = ADOBJECTS(aDBCViews,"View")
        IF nCnt > 0
            =ASORT(aDBCViews)

            FOR i = 1 TO ALLEN(aDBCViews,1)
                IF ThisForm.OptionVerbose
                    WAIT "Loading " + ALLTRIM(aDBCViews(i)) WINDOW AT 20,30 NOCLEAR
                NOWAIT
            
```

```

        ENDIF
        cSQLStmnt = "
        cSQLStmnt = ALLTRIM(DBGETPROP(aDBCViews(i), 'View', 'SQL'))
        INSERT INTO curDBViews VALUES (aDBCViews(i), cSQLStmnt, .F., .F., .F.,
.T., RGB(255,255,255))
    ENDFOR

    IF Thisform.OptionVerbose
        WAIT CLEAR
    ENDIF

    RELEASE aDBCViews
ENDIF

SELECT curDBviews
GO TOP
INDEX ON UPPER(ALLTRIM(ViewName)) TAG DDCidx

WITH Thisform.VCntnrDBC.VGridDBC
    .RecordSource = 'CurDBViews'
    .Column1.ControlSource = 'ViewName'
ENDWITH
ENDIF
ENDIF

WAIT CLEAR

ThisForm.VCntnrDBC.VGridDBC.Column1.HdrViews.Caption = 'Remote Views [' +
ALLTRIM(ThisForm.cCreateTo) + ']'
ThisForm.OpeningData = .F.

** Thisform.Refresh()
ENDPROC

PROCEDURE closedbc
LOCAL nl, cDBC
**
*- Close the DBC.
**
IF Thisform.DBCOpen = .T.
**      WAIT 'Closing DBC' WINDOW AT 20,30 NOCLEAR NOWAIT
**      cDBC = DBC()
**      SET DATABASE TO (ThisForm.cCreateTo)
**      FOR nl = 1 TO Thisform.nToConnect
**          =SQLDISCONNECT(Thisform.aToConnect[nl, 2])
**      ENDFOR
**      Thisform.nToConnect = 0
*      IF Thisform.PackTo
*          WAIT 'Packing DBC' WINDOW AT 20,30 NOCLEAR NOWAIT
*          PACK DATABASE
*          Thisform.PackTo = .F.

```

```

*!* *   ENDIF
*!*   CLOSE DATABASE
*!*   Thisform.DBCOpen = .F.
*!*   Thisform.RecNoDBC = -1
*   Thisform.txtDataSource.Value = ""
*   Thisform.txtDBType.Value = "
*   Thisform.cDBType = "
*   Thisform.txtDBCcnt.Value = "
      Thisform.VCntrDBC.VGridDBC.RecordSource = 'CurDummyDBC'
      Thisform.VCntrDBC.VGridDBC.Refresh()
      WAIT CLEAR
ENDIF
ENDPROC

```

PROCEDURE buildprojectlist

```

LOCAL IHasPjx, cDir
LOCAL cStr, cDirName, nHandle, cDataDir, cFilename

```

```

SET MEMOWIDTH TO 150
SET DELETED ON

```

```

nHandle = 0
nErr = .F.
cDataDir = SET("DEFAULT") + SYS(2003) + "\

```

```

cTmpDir = GETDIR('FishFood\','The Root Dir. must be selected')
IF IEMPTY(cTmpDir)
    ThisForm.cDataDir = cTmpDir
ELSE
    RETURN .F.
ENDIF

```

```

SET DEFA TO (ThisForm.cDataDir)

```

```

cFilename = ThisForm.cDataDir + "vfarmdir.out"
RUN "dir *. > &cFilename."

```

```

IF FILE(cFilename) == .T.
    nHandle = FOPEN(cFilename)
    IF nHandle <= 0
        =MESSAGEBOX("Can't open file list")
        RETURN .F.
    ENDIF
ELSE
    RETURN .F.
ENDIF

```

```

SELECT dirstruct
ZAP
PACK

```

```

**
** Find all the directories at the root and insert them in the directory structure table.
** dirsttruct.dbf
**
cWait = "Scanning Directory, Please Wait..."
WAIT cWait WINDOW AT 20,30 NOCLEAR NOWAIT

DO WHILE (!FEOF(nHandle))
    cStr = FGETS(nHandle)

    IF SUBSTR(cStr, 1, 1) <> "." AND SUBSTR(cStr, 1, 2) <> ".."
        IF AT("<DIR>",cStr) > 0
            cDirName = SUBSTR(cStr, 1, (RAT("<DIR>",cStr)-1))
            IF UPPER(ALLTRIM(cDirName)) <> "VFARM"
                INSERT INTO dirsttruct (RootDir, DirName, HasPjx) ;
                VALUES (ThisForm.cDataDir, cDirName, .F.)
            ENDIF
        ENDIF
    ENDIF
ENDDO
nErr = FCLOSE(nHandle)
IF nErr <> .T.

ENDIF

**
** Find all the directories that have projects and update the table dirsttruct to reflect this.
**

SELECT dirsttruct
GO TOP
SCAN
    cDir = ALLTRIM(UPPER(dirsttruct.DirName))
    IHasPjx= FILE(FULLPATH(cDir)+"\"+(cDir)+".pjx")
    IF IHasPjx ==.T.
        REPLACE HasPjx WITH .T.
    ENDIF
ENDSCAN
GO TOP

SELECT vfarmmst
ZAP
PACK

SELECT dirsttruct
GO TOP

cWait = "Scanning Projects: Collecting Form References, Please Wait..."
WAIT cWait WINDOW AT 20,30 NOCLEAR NOWAIT

SCAN
    IF dirsttruct.hasPjx ==.T.

```

```

cProjname = ALLTRIM(dirstuct.DirName)
cExt = ".pjx"
aShortpath = FILE(FULLPATH(ThisForm.cDataDir + cProjName+"\\" + cProjname + cExt))
IF aShortpath = .T.
    SET DEFA TO (FULLPATH(ThisForm.cDataDir + cProjName))
    SELECT cProjname AS directry, name AS Formused, type AS formtype, mainprog;
FROM (cProjname + cExt);
    WHERE type = 'K' .OR. type = 'R';
    INTO CURSOR CurFormList
    *--
    *-- Closing the Projects before continuing.
    *--
    IF USED (cProjname)
        SELECT (cProjname)
        USE
    ENDIF
    *--
    SELECT CurFormList
    GO TOP

DO WHILE !EOF()
    IMainProg = .F.
    cdirectry = ALLTRIM(CurFormList.directry)
    cFormused = ALLTRIM(CurFormList.Formused)
    cFormType = ALLTRIM(CurFormList.FormType)
    IF CurFormList.MainProg == .T.
        IMainProg = .T.
    ENDIF
    IF cFormType = 'R'
        nEnd1 = ATC('.',cFormused, 1) + 3
        nEnd2 = ATC('.',cFormused, 1) - 1
    ELSE
        nEnd1 = ATC('.',cFormused, 1)+3
        nEnd2 = ATC('.',cFormused, 1)-1
    ENDIF
    IF SUBSTR(cFormused,3,1)="\\" && Traps "c:\\" and ".\\"
        nStart1 = 1
        nStart2 = RATC('\',cFormused,1)+1
        cName = SUBSTR(cFormused, nStart1, (nEnd1 - (nStart1-1)))
        cFormName = SUBSTR(cFormused, nStart2, (nEnd2 - (nStart2-1)))
    ELSE
        nStart1 = 1
        cName = SUBSTR(cFormused, nStart1, (nEnd1 - (nStart1 -1)))
        cFormName = SUBSTR(cFormused, nStart1, (nEnd2 - (nStart1 -1 )))
    ENDIF
    INSERT INTO vformmst (Directry, Formused, Formname, FormType, MainForm) ;
        VALUES (cdirectry, cName, cFormname, cFormType, IMainProg)

    SELECT CurFormList
    SKIP
ENDDO
*--

```



```

        IF USED ('CurFormList')
            SELECT CurFormList
            USE
        ENDIF
    ENDIF
    ENDIF
    SELECT dirstruct
*!*  SKIP
ENDSCAN

ThisForm.LockScreen = .T.
ThisForm.VCntrDBC.VGridAppl.RecordSource = 'CurDummyAppl'
ThisForm.VCntrDBC.VGridAppl.Column1.ControlSource = 'ApplName'
ThisForm.VCntrDBC.VGridAppl.Refresh()
SELECT viewappl
ZAP
PACK
ThisForm.LockScreen = .F.

cExt = ".scx"
cWait = "Scanning Forms: Collecting View References, Please Wait..."
WAIT cWait WINDOW AT 20,30 NOCLEAR NOWAIT

SELECT vfarmmst
GO TOP

SCAN FOR vfarmmst.FormType == 'K'
    cForm = ALLTRIM(vfarmmst.Formused)
    IF SUBSTR(cForm,3,1)="\" && Traps "c:\" and ".\"
        cFrm = SUBSTR(cForm,4)
        cProjpath = SUBSTR(cForm,4,(RAT("\",cForm)-3))
        cTblRef = SUBSTR(cForm,RATC("\",cForm)+1)
        *--
        cDirectry = ALLTRIM(vfarmmst.directry)
        cFormused = SUBSTR(cForm,RATC("\",cForm)+1)
        cFormtype = ALLTRIM(vfarmmst.formtype)
        cFormname = ALLTRIM(vfarmmst.formname)
    ELSE
        cFrm = ALLTRIM(vfarmmst.directry)+"\"+(cForm)
        cProjpath = ALLTRIM(vfarmmst.directry)+"\"
        cTblRef = ALLTRIM(cForm)
        *--
        cDirectry = ALLTRIM(vfarmmst.directry)
        cFormused = ALLTRIM(cForm)
        cFormtype = ALLTRIM(vfarmmst.formtype)
        cFormname = ALLTRIM(vfarmmst.formname)
    ENDIF

    SET DEFA TO (ThisForm.cDataDir)
    * cShortpath = FILE(FULLPATH (cprojpath)+ cFormname +(cExt))
    IF FILE(FULLPATH (cProjPath)+ cFormname +(cExt)) == .T.
    *     SET DEFA TO (ThisForm.cDataDir + cProjPath)

```

```

SET DEFA TO FULLPATH(ThisForm.cDataDir + cProjPath)
cSource = (ThisForm.cDataDir + cFrm)
IF USED((cTblRef))
    SELECT (cTblRef)
ELSE
    USE (cTblRef) IN 18
    SELECT (cTblRef)
ENDIF
GO TOP
IFldchkr = TYPE(SUBSTR(cTblRef,1,LEN(ctblref)-4)+'.METHODS')
IF IFldchkr = 'M'
    SCAN FOR 'COPENTABLE' $ ALLTRIM(UPPER(methods))
        ThisForm.ScanMethod(methods, cDirectry)
        SELECT (cTblRef)
    ENDSCAN
ENDIF
SELECT (cTblRef)
USE
ENDIF
ENDSCAN

!* nNextUniq = NextID('viewappl', 'vwaluniq')
!* IF nNextUniq <> -1
!*     INSERT INTO viewappl (vwaluniq, applname, viewname, InDBC)    ;
!*     VALUES(nNextUniq, 'No Applications', '', 0)
!* ENDIF

WAIT CLEAR

!*  !*      aShortpath = FILE(FULLPATH(ThisForm.cDataDir+cdirectry +"\\"+ cdirectry +cExt))
!*      IF cFormType = 'K'
!*      IF FILE(cName) = .T.
!*          SET DEFA TO (FULLPATH(ThisForm.cDataDir + cdirectry))

!*          SELECT class AS clsname, classloc AS clsused, baseclass AS bsclass,
parent AS parent ;
!*          FROM (cName) ;
!*          INTO CURSOR CurClassList
!*          *--
!*          *-- Closing the Form before continuing.
!*          *--
!*          IF USED (cName)
!*              SELECT (cName)
!*              USE
!*          ENDIF
!*          *--
!*          SELECT CurClassList
!*          GO TOP

!*          SCAN FOR (clsused <> " AND !EMPTY(clsused))
!*          INSERT INTO VfarmCls (Formname, clsused, clsname, bsclass,
parent, level) ;

```

```

*!*                               VALUES ((cName),CurClassList.clsused,CurClassList.clsname, ;
*!*                               CurClassList.bsclass, CurClassList.parent, 1)
*!*                               ENDSCAN
*!*
*!* *!*                           IF USED ('cName')
*!*                               SELECT (cName)
*!*                               USE
*!* *!*                           ENDIF

*!*                               IF USED ('CurClassList')
*!*                               SELECT CurClassList
*!*                               USE
*!*                               ENDIF
*!*                               ENDIF
*!*                               ENDIF
ENDPROC

```

#### PROCEDURE buildcontrollist

```

LOCAL cHomeDir, cPath, tclassloc, tclass, pObj, cClass, tObjname

```

```

SELECT vfarmctl
ZAP

```

```

SELECT vfarmmst
SET ORDER TO
SET FILTER TO formtype == 'K'

```

```

GO TOP

```

```

cHomeDir = ThisForm.cDataDir
IF EMPTY(ThisForm.cDataDir)
    WAIT WINDOW ('Navigate to the Root Directory to start the search')
    cHomeDir = GETDIR('FishFood\','The Root Dir. must be selected')
    RETURN .F.
ENDIF

```

```

SELECT vfarmmst
REPLACE projresv2 WITH .F.
GO TOP
DO WHILE (IEOF())
    cAppName = directry
    CD (cHomeDir + cAppName)
    cScreen = formused
    SELECT 19
    IF (EMPTY(ALIAS()))
        USE
    ENDIF
    IF FILE(FULLPATH(cScreen))
        IF USED(cScreen)
            SELECT (cScreen)
        ELSE

```

```

        USE (cScreen)
    ENDIF
    LOCATE FOR AT("form",BASECLASS)<>0
    cPath      =   ObjName
    tclassloc  =   classloc
    tclass     =   CLASS
    cObj       =   classloc
    cClass     =   CLASS
    tObjname=   ObjName
    USE
    SELECT vfarmmst
    Y=RECNO()
        =ThisForm.FetchControls(cAppName,cPath,cScreen,LTRIM(RTRIM(cPath)),
NoNulls(LTRIM(RTRIM(cScreen))),cPath)
    SELECT vfarmmst
    GOTO Y
        =ThisForm.FetchControls(cAppName,cPath,cScreen,LTRIM(RTRIM(cPath)),cObj,cClass)

    SELECT vfarmmst
    GOTO Y
    SKIP
ELSE
    SELECT vfarmmst
    REPLACE projresv2 WITH .T.
    IF IEOF()
        SKIP
    ELSE
    ENDIF
ENDIF
    SELECT vfarmmst
ENDDO
SELECT 18
USE
SELECT 17
USE

*****
*!*  SELECT vfarmctl
*!*  BROWSE LAST
ENDPROC

PROCEDURE fetchcontrols
    LPARAMETER cAppName, cFormName, cScreen, cPath, pObj, cClass

    LOCAL oldDB, nRecNo, tmpParent, tpcount, tObjname2, tObjname
    LOCAL tpage, tproperties, tBaseClass, tclassloc, tObjname, tParent, tClass
    LOCAL tproperties, tprop2, nMemCnt, nMemLines, n1, n2, optname1, optname2
    *LOCAL cAppName, cFormName, cScreen
    SET EXCLUSIVE OFF

    IF (EMPTY(pObj))

```

```

RETURN
ENDIF

*!* The Object doesn't exist so return
IF (!FILE(pObj))
RETURN
ENDIF

SELECT 17
USE (pObj) IN 17 SHARED

GO TOP

DO WHILE (!EOF())
tBaseClass = BASECLASS
tclassloc = classloc
tObjname= ObjName
tParent = PARENT
tClass = CLASS
tproperties = properties
tprop2 = properties
tmethods = methods
IF (!EMPTY(tBaseClass) ;
AND !INLIST(BASECLASS,"label","shape","dataenvironment","custom","pageobject") ;
AND tObjname <> "")
IF (UPPER(tParent) == UPPER(cClass)) && Removed
IF (INLIST(tBaseClass, "form", "container", "pageframe"))
IF (ATC('forms.vcx',tclassloc) = 0 AND !EMPTY(tclassloc))
SELECT 17
oldDB = ALIAS()
nRecNo = RECNO()
IF (!INLIST(tBaseClass, "pageframe"))
INSERT INTO vfarmctl (screenNme, AppName, FormName, PARENT,
cntrlname, ;
classloc, BaseClass, CntrlSrc, Level);
VALUES (cScreen, cAppName, cFormName, tParent, cPath + "." +
tObjname, ;
tclassloc, tBaseClass, GetProp('ControlSource',tproperties),0)
ThisForm.ScanMethod(tMethods, cAppName)
ENDIF
tmpparent = tParent + "." + ObjName
tObjname2 = ObjName
IF (ATC('controls.vcx',tclassloc)=0)
= ThisForm.FetchControls(cAppName, cFormName, cScreen, ;
cPath + '.' + tObjname, tclassloc, tClass)
ELSE
IF UPPER(tBaseClass) = 'PAGEFRAME'
*!cnt=val(getprop("pagecount",tproperties))
cnt = VAL(SUBSTR(ALLTRIM(tproperties),
ATC('pagecount',tproperties) + 12, 2))
IF cnt == 0 && Strange Foxpro behavior when no additional pages
are added to the default of 2

```

```

        lcnt = 2
    ENDIF
    FOR Xcnt = 1 TO lcnt
        tpage = "Page" + LTRIM(STR(Xcnt,2))
        tpage = GetRealName(tpage,tproperties)
        tpcount = tmpparent+"."+tpage
* may have to actually find it's caption...
* should test that.
        =ThisForm.FetchControls(cAppName, cFormName, cScreen, ;
            cPath + "." + tObjname2 + "." + tpage, pObj, tpcount)
    NEXT Xcnt
ELSE
    =ThisForm.FetchControls(cAppName, cFormName, cScreen, ;
        cPath + "." + tObjname2, pObj, tmpparent)
ENDIF
ENDIF
IF (EMPTY(pObj) AND FILE(pObj))
    USE (pObj) IN 17
    GOTO nRecNo
ENDIF
ELSE
ENDIF
ELSE
    IF (INLIST(tBaseClass,"optiongroup"))
        INSERT INTO vfarmctl (screenNme, AppName, FormName, PARENT, cntrlname, ;
            classloc, BASECLASS, CntrlSrc, Level) ;
            VALUES (cScreen, cAppName, cFormName, tParent, ;
                cPath + "." + tObjname, tclassloc, tBaseClass, ;
                GetProp('ControlSource', tproperties),0)
        ThisForm.ScanMethod(tMethods, cAppName)

        IF .F.    && This section will include the individual options ie; option1,option2
of the optiongroup
            nMemLines = MEMLINES(tproperties)
            FOR nMemCnt = 1 TO nMemLines
                tmp = MLINE(tproperties,nMemCnt)
                IF (LEFT(tmp,6) = 'Option') THEN && Only look for Option Line
                    IF (AT('.',tmp) <> 0) THEN && Get the Options Name
                        n1 = AT(".",tmp)+1
                        n2 = RAT(".",tmp)-1
                        optname2 = SUBSTR(tmp,n1,n2-n1+1)
                        n1 = AT('.',tmp)-1
                        optname1=SUBSTR(tmp,1,n1)
                        INSERT INTO vfarmctl (screenNme, AppName,
FormName, PARENT, ;
                                cntrlname, classloc, BASECLASS, CntrlSrc,
Level) ;
                                VALUES (cScreen, cAppName, cFormName,tParent, ;
                                    cPath + "." + tObjname + "." + optname2, ;
                                    tclassloc, tBaseClass,
GetProp('ControlSource',tproperties),0)

```

```

                                ThisForm.ScanMethod(tMethods, cAppName)
                                ENDIF
                                ENDIF
                                NEXT nMemCnt
                                ENDIF
                                ELSE
                                INSERT INTO vformctl (screenNme, AppName, FormName, PARENT, ;
                                cntriname, classloc, BASECLASS, CntrlSrc, Level) ;
                                VALUES (cScreen, cAppName, cFormName, tParent, ;
                                cPath+"."+tObjname, tclassloc, tBaseClass, ;
                                GetProp('ControlSource', tproperties), 0)
                                ThisForm.ScanMethod(tMethods, cAppName)
                                ENDIF
                                SELECT 17
                                ENDIF
                                ELSE
                                IF (tBaseClass == "form")
                                SELECT 17
                                oldDB = ALIAS()
                                nRecNo = RECNO()
                                ThisForm.ScanMethod(tMethods, cAppName)
                                ENDIF
                                ENDIF
                                ENDIF
                                SELECT 17
                                SKIP
                                ENDDO

                                * select whatever db was active before you came in here.
                                SELECT 17

                                ***

                                RETURN
                                ENDPROC

```

```

PROCEDURE scanmethod
    LPARAMETER pMethods, cAppName
    LOCAL cLine, i, ix, nStart, cStr, nQuote, cViewname

    ix = MEMLINES(pMethods)
    STORE 0 TO _MLINE
    FOR i = 1 TO ix
        nStart = 0
        cLine = UPPER(MLINE(pMethods, i))
        nLineEnd = LEN(cLine)
        IF ".ADDOBJECT" $ cLine AND 'COPENTABLE' $ cLine
            nStart = AT("COPENTABLE", cLine) + 12
            cViewname = ""
            cStr = SUBSTR(cLine, nStart, 1)
            nQuote = 0

```

```

    *!* Clear all white spaces and tabs to the quote.
    DO WHILE ((cStr <> "") AND (cStr <> " ") AND (nStart < nLineEnd))
        nStart = nStart + 1
        cStr = SUBSTR(cLine, nStart, 1)
        IF ((cStr == "'") .OR. (cStr == " ")) AND (nQuote == 0)
            nQuote = nQuote + 1
            cViewname = cViewname + cStr
        ENDIF
    ENDDO
    *!* Collect all the cahracters between the quotes.
    nEnd = nStart + 1
    cStr = SUBSTR(cLine, nEnd, 1)
    cViewname = cViewname + cStr
    DO WHILE ((cStr <> "") AND (cStr <> " ") AND (nEnd < nLineEnd))
        nEnd = nEnd + 1
        cStr = SUBSTR(cLine, nEnd, 1)
        cViewname = cViewname + cStr
        IF ((cStr == "'") .OR. (cStr == " ")) AND (nQuote == 1)
            nQuote = nQuote + 1
        ENDIF
    ENDDO

    IF nQuote == 2
        *--
        *-- After gathering the data it is inserted into viewappl
        *--
        cView = StripQuotes(cViewname)
        nNextUniq = NextID('viewappl', 'vwaluniq')
        IF nNextUniq <> -1
            INSERT INTO viewappl (vwaluniq, applname, viewname, lnDBC) ;
                VALUES(nNextUniq, UPPER(cAppName),UPPER(cView),0)
        ENDIF
    ENDIF
ENDIF
ENDIF
NEXT i
ENDPROC

```

```

PROCEDURE removeduplicates
    LOCAL cAppName, cCntrlName, cScreenNme

    SET DELETED OFF
    SELECT vfarmctl

    IF RECCOUNT('vfarmctl') <=1
        *SET STEP ON
        RETURN
    ENDIF

    INDEX ON apname+LEFT(cntrlname,80)+screenNme TAG SORTED
    GO TOP

```



```

CNT=0

cAppName=appname
cCntrlName=cntrlname
cScreenNme=screenNme
SKIP
* Note: the first record can't be a duplicate!

DO WHILE (!EOF())
* is it a duplicate?
    IF (cAppName = appname .AND. cCntrlName = cntrlname .AND. cScreenNme = screenNme)
        *? appname,cntrlname,screenNme
        *?? recno()
        DELETE
        CNT=CNT+1
    ELSE
        cAppName=appname
        cCntrlName=cntrlname
        cScreenNme=screenNme
    ENDIF
    SKIP
ENDDO
PACK
***
*!* RETURN .T.
ENDPROC

```

```

PROCEDURE dupcheck
    LPARAMETER cViewName
    LOCAL nRecCnt

    nRecCnt = 0

    SELECT RECCOUNT() AS nRecCount ;
    FROM curDBViews INTO CURSOR CurCnt ;
    WHERE ALLTRIM(ViewName) = ALLTRIM(cViewName)

    SELECT CurCnt
    SCAN
        nRecCnt = nRecCount
    ENDSCAN

    IF USED ('CurCnt')
        USE IN CurCnt
    ENDIF

    RETURN nRecCnt
ENDPROC

```

```

PROCEDURE find

```

```

LPARAMETERS nArg

IF nArg == 0
    DO FORM RVMFind WITH 0, Thisform.VCntnrDBC.DisplayBy.Value TO nRecNo

    IF Thisform.VCntnrDBC.DisplayBy.Value == 3
        cCursorName = 'CurApplByViews'
    ELSE
        cCursorName = 'ViewAppl_Distinct'
    ENDIF
    IF nRecNo > 0 AND nRecNo <= RECCOUNT(cCursorName)
        SELECT (cCursorName)
        GO nRecNo
    ENDIF
    ThisForm.VCntnrDBC.VGridAppl.SetFocus()
ELSE
    IF nArg == 1
        DO FORM RVMFind WITH 1, Thisform.VCntnrDBC.DisplayBy.Value TO nRecNo
        IF Thisform.VCntnrDBC.DisplayBy.Value == 2
            cCursorName = 'CurViewsByAppl'
        ELSE
            cCursorName = 'curDBViews'
        ENDIF
        IF nRecNo > 0 AND nRecNo <= RECCOUNT(cCursorName)
            SELECT (cCursorName)
            GO nRecNo
        ENDIF
        ThisForm.VCntnrDBC.VGridDBC.SetFocus()
    ENDIF
    ThisForm.Refresh()
ENDPROC

PROCEDURE QueryUnload
    ThisForm.VCBQuit.Click()
ENDPROC

PROCEDURE Unload
    CLOSE DATABASES ALL

    CLEAR EVENTS
ENDPROC

PROCEDURE Init
    *DO vfarm.mpr &&WITH This, .T.
    Vfarmform::Init()
    Thisform.MenuInit()
    ThisForm.VfarmCntnrViews.Visible = .F.
    ThisForm.VCntnrDBC.Visible = .T.

```

\*!\* Set the list index and load the dbc grid.

```
ThisForm.vCboDBC.ListIndex = 1
ThisForm.OpenDBC()
ThisForm.VCntnrDBC.VGridAppl.RecordSource = 'ViewAppl_distinct'
ThisForm.VCntnrDBC.VGridAppl.Column1.ControlSource = 'ApplName'
```

```
ThisForm.Refresh()
```

```
*!* ThisForm.cCreateFrom = "
```

```
*!* ThisForm.cCreateTo = "
```

```
ENDPROC
```

#### PROCEDURE Load

```
Vfarmform::Load()
```

```
ThisForm.IProcessCanceled = .F.
```

```
SET BELL OFF
SET TALK OFF
SET ECHO OFF
SET SAFETY OFF
SET MULTLOCKS ON
SET DELETED ON
SET STATUS BAR ON
SET CLOCK STATUS
SET CENTURY ON
SET EXCLUSIVE ON
SET DEFAULT TO CURDIR()
SET ESCAPE ON
SET EXACT ON
SET HOURS TO 24
```

```
SET PROCEDURE TO ..\PROGS\utility.prg ADDITIVE
SET CLASSLIB TO ..\LIBS\vfbase.vcx ADDITIVE
SET CLASSLIB TO ..\LIBS\vfarm.vcx ADDITIVE
SET CLASSLIB TO ..\LIBS\utilities.vcx ADDITIVE
```

```
ON ERROR DO errhand WITH ;
    ERROR( ), MESSAGE( ), MESSAGE(1), PROGRAM( ), LINENO( )
```

```
OPEN DATABASE c:\vfarm\data\vfarm EXCLUSIVE
SET DATABASE TO c:\vfarm\data\vfarm
```

```
ThisForm.Addobject('odbcpath',          'cOpenTable', 'dbcpath',      .T.)
ThisForm.Addobject('odirstruct',        'cOpenTable', 'dirstruct', .T.)
ThisForm.Addobject('oviewappl',         'cOpenTable', 'viewappl',   .T.)
```

```

Thisform.Addobject('ovfarmmst', 'cOpenTable', 'vfarmmst', .T.)
Thisform.Addobject('oDBViews', 'cOpenTable', 'DBViews', .T.)
Thisform.Addobject('ovfarmctl', 'cOpenTable', 'vfarmctl', .T.)
Thisform.Addobject('odbviews_viewappl', 'cOpenTable', 'DBViews_viewappl', .T.)
Thisform.Addobject('oviewappl_distinct', 'cOpenTable', 'viewappl_distinct', .T.)
Thisform.Addobject('oviewname_distinct', 'cOpenTable', 'viewname_distinct', .T.)
Thisform.Addobject('oDBViews_nop', 'cOpenTable', 'DBViews_nop', .T.)
Thisform.Addobject('oViewAppl_By_ApplName', 'cOpenTable', 'ViewAppl_By_ApplName', .F.)
Thisform.Addobject('oViewAppl_By_ViewName', 'cOpenTable', 'ViewAppl_By_ViewName', .F.)

```

```

CREATE CURSOR CurDBViews (ViewName C(32) NULL, cSQLStatement C(250) NULL, ISelected L,
ICurrentRec L, linAppl L NULL, linDbc L NULL, BackColor I NULL)
CREATE CURSOR CurDummyDBC (ViewName C(32) NULL, cSQLStatement C(250) NULL, ISelected L,
ICurrentRec L, linAppl L NULL, linDbc L NULL, BackColor I NULL)
CREATE CURSOR CurViewsByAppl (ViewName C(32) NULL, cSQLStatement C(250) NULL, ISelected L,
ICurrentRec L, linAppl L NULL, linDbc L NULL, BackColor I NULL)
CREATE CURSOR CurApplByViews (ApplName C(32) NULL, ViewName C(32) NULL, ISelected L,
ICurrentRec L)
CREATE CURSOR CurDummyAppl (ApplName C(32) NULL, ViewName C(32) NULL, ISelected L,
ICurrentRec L)

```

```

ISuccess = CURSORSETPROP("Buffering", 5, "CurDBViews")
ISuccess = CURSORSETPROP("Buffering", 5, "CurViewsByAppl")
ISuccess = CURSORSETPROP("Buffering", 5, "CurApplByViews")
ISuccess = CURSORSETPROP("Buffering", 5, "dbcpath")
*ISuccess = CURSORSETPROP("Buffering", 1, "dirstruct")

```

ENDPROC

```

PROCEDURE Refresh
    IF Thisform.VCntrDBC.DisplayBy.Value == 2      && View by Applicatio
        ThisForm.VCntrDBC.VGridDBC.Column1.DynamicBackColor = "CurViewsByAppl.BackColor"
    ELSE
        ThisForm.VCntrDBC.VGridDBC.Column1.DynamicBackColor = "CurDBViews.BackColor"
    ENDIF
ENDPROC

```

## VfarmCntrViews

## VfarmcmdFirst

```

PROCEDURE Click
    IF Thisform.VCntrDBC.DisplayBy.Value <> 2      && View by Applications
        cCursorname = 'curDBViews'
    ENDIF
    ThisForm.LockScreen = .T.
    SELECT (cCursorname)

```

GO TOP

ThisForm.VfarmCntnrViews.VFarmCmdFirst.Enabled = .F.  
ThisForm.VfarmCntnrViews.VFarmCmdPrev.Enabled = .F.

ThisForm.VfarmCntnrViews.VFarmCmdNext.Enabled = .T.  
ThisForm.VfarmCntnrViews.VFarmCmdLast.Enabled = .T.

ThisForm.VfarmCntnrViews.Refresh()  
\*!\* ThisForm.Refresh()  
ThisForm.LockScreen = .F.  
ENDPROC

### VfarmcmdSave

```
PROCEDURE Click
LOCAL cViewName, cSql
IF ThisForm.IMode = 'A'
    cViewName = ALLTRIM(ThisForm.VfarmCntnrViews.VfarmViewName.Value)
    cSql = ALLTRIM(ThisForm.VfarmCntnrViews.VfarmSqlStmnt.Value)
    IF EMPTY(cViewName)
        =MESSAGEBOX("Complete Required Fields ?", 16)
        ThisForm.VfarmCntnrViews.VfarmViewName.SetFocus()
        RETURN
    ENDIF
    IF EMPTY(cSql)
        =MESSAGEBOX("Complete Required Fields ?", 16)
        ThisForm.VfarmCntnrViews.VfarmSqlStmnt.SetFocus()
        RETURN
    ENDIF

    IF ThisForm.DupCheck(cViewName) > 1
        nAnswer =MESSAGEBOX("Duplicate View Name, Please Re-Enter", 21)
        IF nAnswer = 4      &&Retry
            ThisForm.VfarmCntnrViews.VfarmViewName.Value = "
            ThisForm.VfarmCntnrViews.VfarmViewName.SetFocus()
            RETURN
        ELSE && nAnswer = 2      Cancel
            SELECT curDBViews
            GO TOP
            RETURN
        ENDIF
    ENDIF
    WAIT 'Creating View ' + cViewName WINDOW AT 20,30 NOCLEAR NOWAIT
    OPEN DATABASE (ThisForm.cCreateTo) EXCLUSIVE
    SET DATABASE TO (ThisForm.cCreateTo)

    cConnection = 'FishConn'

    CREATE SQL VIEW (cViewName) CONNECTION (cConnection) ;
```

```

        AS &cSql.

    ON ERROR

    WAIT CLEAR
    **      Thisform.DBCOpen = .T.

    **      SELECT curDBViews
    **      GO TOP

    **      INSERT INTO curDBViews VALUES (cViewName, cSql, .F., .F., .F., .T., RGB(255,255,255))
    **
    **      SELECT curDBViews
    **      GO TOP

ENDIF

WAIT CLEAR
Thisform.OpenDBC()

ThisForm.VfarmCntrViews.Visible = .F.
ThisForm.VCntrDBC.Visible = .T.
ThisForm.VCntrDBC.VGridDBC.SetFocus()
ThisForm.VCBSync.Enabled = .T.
ThisForm.Refresh()
ENDPROC

```

#### **VfarmViewName**

```

PROCEDURE InteractiveChange
    IF !EMPTY(This.Value) AND !EMPTY(ThisForm.VfarmCntrViews.VfarmSqlStmnt.Value)
        ThisForm.VfarmCntrViews.VfarmcmdSave.Enabled = .T.
    ELSE
        ThisForm.VfarmCntrViews.VfarmcmdSave.Enabled = .F.
    ENDIF
ENDPROC

```

#### **VfarmSqlStmnt**

```

PROCEDURE InteractiveChange
    IF !EMPTY(This.Value) AND !EMPTY(ThisForm.VfarmCntrViews.VfarmViewName.Value)
        ThisForm.VfarmCntrViews.VfarmcmdSave.Enabled = .T.
    ELSE
        ThisForm.VfarmCntrViews.VfarmcmdSave.Enabled = .F.
    ENDIF
ENDPROC

```

#### **VfarmIblViewName**

### VfarmIblSqlStmnt

#### VfarmcmdPrev

```
PROCEDURE Click
  IF Thisform.VCntnrDBC.DisplayBy.Value <> 2      && View by Applications
    cCursorname = 'curDBViews'
  ENDIF
  Thisform.lockscreen = .T.
  SELECT (cCursorname)

  ThisForm.VfarmCntnrViews.VFarmCmdNext.Enabled = .T.
  ThisForm.VfarmCntnrViews.VFarmCmdLast.Enabled = .T.

  IF RECNO('&cCursorname') > 1
    SKIP -1
    IF RECNO('&cCursorname') <= 1
      ThisForm.VfarmCntnrViews.VFarmCmdFirst.Enabled = .F.
      ThisForm.VfarmCntnrViews.VFarmCmdPrev.Enabled = .F.
    ENDIF
    ThisForm.VfarmCntnrViews.Refresh()
    *!* ThisForm.Refresh()
  ENDIF

  Thisform.lockscreen = .F.
ENDPROC
```

#### VfarmcmdLast

```
PROCEDURE Click
  IF Thisform.VCntnrDBC.DisplayBy.Value <> 2      && View by Applications
    cCursorname = 'curDBViews'
  ENDIF
  Thisform.lockscreen = .T.
  SELECT (cCursorname)
  GO BOTTOM

  ThisForm.VfarmCntnrViews.VFarmCmdFirst.Enabled = .T.
  ThisForm.VfarmCntnrViews.VFarmCmdPrev.Enabled = .T.
  ThisForm.VfarmCntnrViews.VFarmCmdNext.Enabled = .F.
  ThisForm.VfarmCntnrViews.VFarmCmdLast.Enabled = .F.

  ThisForm.VfarmCntnrViews.Refresh()
  *!* ThisForm.Refresh()
  Thisform.lockscreen = .F.
ENDPROC
```

#### VfarmcmdNext

```

PROCEDURE Click
    IF ThisForm.VCntrDBC.DisplayBy.Value <> 2      && View by Applications
        cCursorname = 'curDBViews'
    ENDIF
    ThisForm.lockscreen = .T.
    SELECT (cCursorname)

    ThisForm.VfarmCntrViews.VFarmCmdFirst.Enabled = .T.
    ThisForm.VfarmCntrViews.VFarmCmdPrev.Enabled = .T.

    IF RECNO('&cCursorname') <= RECCOUNT('&cCursorname')
        SKIP 1
        IF RECNO('&cCursorname') >= RECCOUNT('&cCursorname')
            ThisForm.VfarmCntrViews.VFarmCmdNext.Enabled = .F.
            ThisForm.VfarmCntrViews.VFarmCmdLast.Enabled = .F.
        ENDIF
        ThisForm.VfarmCntrViews.Refresh()
    *!*      ThisForm.Refresh()
    ENDIF
    ThisForm.lockscreen = .F.
ENDPROC

```

#### VfarmCmdDone

```

PROCEDURE Click
    *!* IF ThisForm.lMode = 'A'
    *!* *!*      DELETE FROM curDBViews WHERE ALLTRIM(ViewName) =
    ALLTRIM(ThisForm.VfarmCntrViews.VfarmViewName.Value)
    *!*      SELECT curDBViews
    *!*      GO TOP
    *!* ENDIF
    ThisForm.OpenDBC()

    ThisForm.VCBSync.Enabled = .T.
    ThisForm.VfarmCntrViews.Visible = .F.
    ThisForm.VCntrDBC.Visible = .T.
    ThisForm.VCntrDBC.VGridDBC.SetFocus()
ENDPROC

```

#### VCntrDBC

##### Shape1

##### Shape4

##### Shape2

#### VCBDelete

```

PROCEDURE Click
    LOCAL cViewName, nAnswer

```



```

cViewName = ALLTRIM(UPPER(curDBViews.ViewName))
SELECT ViewAppl_By_ViewName
=REQUERY()
GO TOP

IF RECCOUNT('ViewAppl_By_ViewName') > 1
    =MESSAGEBOX("Cannot delete a view that is used by more than one applications")
    RETURN
ENDIF

SET DELETE ON

cViewName = ALLTRIM(curDBViews.ViewName)
nAnswer = MESSAGEBOX('Do you want to delete the view ' + ALLTRIM(cViewName) +
'?',36,Thisform.Caption)
IF nAnswer == 6                && yes

CLOSE DATABASES
*! OPEN DATABASE (HOME(2) + 'Data\testdata')

*! CREATE SQL VIEW myview
*! CLEAR
*! DISPLAY DATABASE

OPEN DATABASE (ThisForm.cCreateTo) EXCLUSIVE
SET DATABASE TO (ThisForm.cCreateTo)

cConnection = 'FishConn'

DELETE VIEW (cViewName)
*! CREATE SQL VIEW (cViewName) CONNECTION (cConnection) ;
*! AS &cSql.

DELETE FROM curDBViews WHERE ALLTRIM(ViewName) = cViewName

SELECT curDBViews
GO TOP

ELSE
ENDIF
ThisForm.Refresh()
ENDPROC

```

## VGridAppl

```

PROCEDURE AfterRowColChange
LPARAMETERS nColIndex

*SET STEP ON
IF Thisform.VCntrDBC.DisplayBy.Value == 2        && View by Applicatio
    Thisform.LockScreen = .T.

```

```

ThisForm.VCntnrDBC.VGridDBC.RecordSource = 'CurDummyDBC'

cAppName = ALLTRIM(ViewAppl_Distinct.ApplName)
IF USED('CurViewsByAppl')
    SELECT CurViewsByAppl
    USE
ENDIF

CREATE CURSOR CurViewsByAppl (ViewName C(32) NULL, cSQLStatement C(250) NULL, ISelected L,
ICurrentRec L, linAppl L NULL, linDbc L NULL, BackColor I NULL)

SELECT ViewAppl_By_ApplName
=REQUERY()
GO TOP
SCAN
    cViewName = ViewAppl_By_ApplName.ViewName
    SELECT curDBViews
    SCAN FOR ALLTRIM(UPPER(ViewName)) = ALLTRIM(UPPER(cViewName))
    INSERT INTO CurViewsByAppl ;
        VALUES (curDBViews.ViewName, curDBViews.cSQLStatement, ;
            curDBViews.ISelected, curDBViews.ICurrentRec, ;
            curDBViews.linAppl, curDBViews.linDbc, curDBViews.BackColor)
    ENDSCAN
ENDSCAN

SELECT CurViewsByAppl
GO TOP
ThisForm.VCntnrDBC.VGridDBC.RecordSource = 'CurViewsByAppl'
This.Column1.DynamicBackColor = "CurViewsByAppl.BackColor"

ThisForm.LockScreen = .F.
ENDIF
ENDPROC

```

## Header1

## VtxtAppl

```

PROCEDURE GotFocus
    *!* Vfarmtextbox::GotFocus()

    *!* SELECT ViewName_Distinct
    *!* ThisForm.CurrentApplRow = RECNO('ViewName_Distinct')
    DODEFAULT()
ENDPROC

```

## DisplayBy

PROCEDURE Click

```
ThisForm.LockScreen = .T.  
ThisForm.VCntrDBC.VGridAppl.RecordSource = 'CurDummyAppl'  
ThisForm.VCntrDBC.VGridAppl.Column1.ControlSource = 'ApplName'  
ThisForm.VCntrDBC.VGridDBC.RecordSource = 'CurDummyDBC'
```

DO CASE

CASE This.Value == 1

```
SELECT curDBViews  
GO TOP  
SELECT ViewAppl_Distinct  
=REQUERY()  
GO TOP
```

```
ThisForm.VCntrDBC.VGridDBC.RecordSource = 'curDBViews'  
ThisForm.VCntrDBC.VGridAppl.RecordSource = 'ViewAppl_Distinct'  
ThisForm.VCntrDBC.VGridAppl.Column1.ControlSource = 'ApplName'  
ThisForm.VCBSync.Enabled = .T.  
ThisForm.VCntrDBC.VCBAdd.Enabled = .T.  
ThisForm.VCntrDBC.VCBEdit.Enabled = .T.  
ThisForm.VCntrDBC.VCBDelete.Enabled = .T.
```

CASE This.Value == 2

```
SELECT ViewAppl_Distinct  
=REQUERY()  
GO TOP  
i = 0  
DO WHILE !EOF('ViewAppl_Distinct') AND i < 1  
    cApplName = ALLTRIM(ViewAppl_Distinct.applname)  
    i = i + 1  
ENDDO  
IF USED('CurViewsByAppl')  
    SELECT CurViewsByAppl  
    USE  
ENDIF
```

```
CREATE CURSOR CurViewsByAppl (ViewName C(32) NULL, cSQLStatement C(250) NULL, ISelected L,  
ICurrentRec L, linAppl L NULL, linDbc L NULL, BackColor I NULL)
```

```
SELECT ViewAppl_By_ApplName  
=REQUERY()  
GO TOP  
SCAN  
    cViewName = ViewAppl_By_ApplName.ViewName  
    SELECT curDBViews  
    SCAN FOR ALLTRIM(UPPER(ViewName)) = ALLTRIM(UPPER(cViewName))  
    INSERT INTO CurViewsByAppl ;  
        VALUES (curDBViews.ViewName, curDBViews.cSQLStatement, ;  
            curDBViews.ISelected, curDBViews.ICurrentRec, ;
```

```

                                curDBViews.linAppl, curDBViews.linDbc, RGB(255,255,255))
        ENDSCAN
    ENDSCAN

```

```

SELECT ViewAppl_Distinct
=REQUERY()
GO TOP
ThisForm.VCntrDBC.VGridDBC.RecordSource = 'CurViewsByAppl'
ThisForm.VCntrDBC.VGridAppl.RecordSource = 'ViewAppl_Distinct'
ThisForm.VCntrDBC.VGridAppl.Column1.ControlSource = 'ApplName'
ThisForm.VCBSync.Enabled = .F.
ThisForm.VCntrDBC.VCBAAdd.Enabled = .F.
ThisForm.VCntrDBC.VCBEdit.Enabled = .F.
ThisForm.VCntrDBC.VCBDelete.Enabled = .F.

```

CASE This.Value == 3

```

SELECT curDBViews
GO TOP
i = 0
DO WHILE !EOF('curDBViews') AND i < 1
    cViewName = ALLTRIM(UPPER(curDBViews.ViewName))
    i = i + 1
ENDDO
IF USED('CurApplByViews')
    SELECT CurApplByViews
    USE
ENDIF

```

```

CREATE CURSOR CurApplByViews(ApplName C(32), ViewName C(32), ISelected L, ICurrentRec

```

L)

```

SELECT ViewAppl_By_ViewName
=REQUERY()
GO TOP
SCAN
    INSERT INTO CurApplByViews ;
        VALUES (ViewAppl_By_ViewName.ApplName, cViewName, .F., .F.)
ENDSCAN

```

```

SELECT curDBViews
GO TOP

```

```

SELECT CurApplByViews
GO TOP
ThisForm.VCntrDBC.VGridDBC.RecordSource = 'curDBViews'
ThisForm.VCntrDBC.VGridAppl.RecordSource = 'CurApplByViews'
ThisForm.VCntrDBC.VGridAppl.Column1.ControlSource = 'ApplName'
ThisForm.VCBSync.Enabled = .F.
ThisForm.VCntrDBC.VCBAAdd.Enabled = .F.
ThisForm.VCntrDBC.VCBEdit.Enabled = .F.
ThisForm.VCntrDBC.VCBDelete.Enabled = .F.

```

```

ENDCASE
Thisform.Refresh()
Thisform.LockScreen = .F.

```

```

ENDPROC

```

## VGridDBC

```

PROCEDURE Init
  VFarmgrid::Init()

  *This.Column1.DynamicForeColor = "CurDBViews.forecolor"
  *This.Column1.DynamicBackColor = "CurDBViews.BackColor"

  *! * This.SetAll("Sparse", .F.)

  This.SetAll("FontBold", .F.)

  This.SetAll("BackColor", RGB(255,255,255))
  This.SetAll("ForeColor", RGB(0,0,0))

  This.SetAll("BackColor", RGB(0,0,128), "Header")
  This.SetAll("ForeColor", RGB(192,192,192), "Header")

  This.SetAll("ReadOnly", .T.)
  This.SetAll("Resizable", .F.)
  This.SetAll("Movable", .F.)
  This.SetAll("SelectOnEntry", .F.)
ENDPROC

```

```

PROCEDURE AfterRowColChange
  LPARAMETERS nColIndex

```

```

*SET STEP ON
IF Thisform.VCntrDBC.DisplayBy.Value == 3      && View by Remote View
  Thisform.LockScreen = .T.

  ThisForm.VCntrDBC.VGridAppl.RecordSource = 'CurDummyAppl'
  ThisForm.VCntrDBC.VGridAppl.Column1.ControlSource = 'ApplName'
  cViewName = ALLTRIM(UPPER(curDBViews.ViewName))
  IF USED('CurApplByViews')
    SELECT CurApplByViews
    USE
  ENDIF

  CREATE CURSOR CurApplByViews(ApplName C(32), ViewName C(32), ISelected L, ICurrentRec L)

  SELECT ViewAppl_By_ViewName
  =REQUERY()

```

```

GO TOP
SCAN
    INSERT INTO CurApplByViews ;
        VALUES (ViewAppl_By_ViewName.ApplName, cViewName, .F., .F.)
ENDSCAN

SELECT CurApplByViews
GO TOP

ThisForm.VCntrDBC.VGridAppl.RecordSource = 'CurApplByViews'
ThisForm.VCntrDBC.VGridAppl.Column1.ControlSource = 'ApplName'
ThisForm.LockScreen = .F.
ENDIF
ENDPROC

```

## HdrViews

### VtxtViews

```

PROCEDURE Click
    Vfarmtextbox::Click()
ENDPROC

PROCEDURE DblClick
    Vfarmtextbox::DblClick()
    This.RightClick()
ENDPROC

PROCEDURE GotFocus
    Vfarmtextbox::GotFocus()
ENDPROC

PROCEDURE RightClick
    Vfarmtextbox::RightClick()
    LOCAL nRecNo
    IF ThisForm.VCntrDBC.DisplayBy.Value <> 1      && View by Applications
        =MESSAGEBOX("Feature available only when Displaying All")
        RETURN
    ELSE
        cCursormame = 'curDBViews'
        WITH ThisForm.VfarmCntrViews
            .VfarmViewName.ControlSource = 'ViewName'
            .VfarmSqlStmnt.ControlSource = 'cSQLStatement'
        ENDWITH
    ENDIF

    ThisForm.VfarmCntrViews.VfarmcmdDone.Caption = 'Done'

    SELECT curDBViews
    nRecNo = RECNO('curDBViews')

```

GOTO nRecNO

```
ThisForm.IMode = 'V'
ThisForm.LockScreen = .T.
ThisForm.VCBSync.Enabled = .F.
ThisForm.VfarmCntnrViews.VfarmViewName.Enabled = .F.
ThisForm.VfarmCntnrViews.VfarmSqlStmnt.Enabled = .F.
ThisForm.VfarmCntnrViews.VfarmcmdFirst.Enabled = .T.
ThisForm.VfarmCntnrViews.VfarmcmdNext.Enabled = .T.
ThisForm.VfarmCntnrViews.VfarmcmdPrev.Enabled = .T.
ThisForm.VfarmCntnrViews.VfarmcmdLast.Enabled = .T.
ThisForm.VfarmCntnrViews.VfarmcmdSave.Enabled = .F.
ThisForm.VCntnrDBC.Visible = .F.
ThisForm.VfarmCntnrViews.Visible = .T.
ThisForm.LockScreen = .F.
ThisForm.VfarmCntnrViews.VfarmCmdDone.SetFocus()
ThisForm.Refresh()
```

ENDPROC

#### **VCBEdit**

PROCEDURE Click

```
IF ThisForm.VCntnrDBC.DisplayBy.Value == 2      && View by Applications
    =MESSAGEBOX("Feature available only when viewing by Applications")
    RETURN
```

ELSE

```
    cCursorname = 'curDBViews'
    WITH ThisForm.VfarmCntnrViews
        .VfarmViewName.ControlSource = 'ViewName'
        .VfarmSqlStmnt.ControlSource = 'cSQLStatement'
```

ENDWITH

ENDIF

```
cViewName = ALLTRIM(UPPER(curDBViews.ViewName))
SELECT ViewAppl_By_ViewName
=REQUERY()
GO TOP
```

```
IF RECCOUNT("ViewAppl_By_ViewName") > 1
    =MESSAGEBOX("Cannot edit a view that is used by more than one applications")
    RETURN
```

ENDIF

ThisForm.VfarmCntnrViews.VfarmcmdDone.Caption = 'Done'

```
SELECT curDBViews
nRecNo = RECNO()
GOTO nRecNO
```

```
ThisForm.IMode = 'A'
ThisForm.LockScreen = .T.
ThisForm.VCBSync.Enabled = .F.
```

```

        ThisForm.VfarmCntnrViews.VfarmViewName.Enabled = .T.
        ThisForm.VfarmCntnrViews.VfarmSqlStmnt.Enabled = .T.
        ThisForm.VfarmCntnrViews.VfarmcmdFirst.Enabled = .F.
        ThisForm.VfarmCntnrViews.VfarmcmdNext.Enabled = .F.
        ThisForm.VfarmCntnrViews.VfarmcmdPrev.Enabled = .F.
        ThisForm.VfarmCntnrViews.VfarmcmdLast.Enabled = .F.
        ThisForm.VfarmCntnrViews.VfarmcmdDone.Caption = 'Cancel'
        ThisForm.VfarmCntnrViews.VfarmcmdSave.Enabled = .F.
        ThisForm.VCntnrDBC.Visible = .F.
        ThisForm.VfarmCntnrViews.Visible = .T.
        ThisForm.LockScreen = .F.
        ThisForm.VfarmCntnrViews.VfarmViewName.SetFocus()
        ThisForm.Refresh()
    ENDPROC

```

### **VCBAdd**

```

PROCEDURE Click
    ThisForm.LockScreen = .T.
    SELECT curDBViews
    *!* APPEND BLANK

    ThisForm.IMode = 'A'
    ThisForm.VCBSync.Enabled = .F.
    ThisForm.VfarmCntnrViews.VfarmViewName.Value = ""
    ThisForm.VfarmCntnrViews.VfarmSqlStmnt.Value = ""
    ThisForm.VfarmCntnrViews.VfarmViewName.Enabled = .T.
    ThisForm.VfarmCntnrViews.VfarmSqlStmnt.Enabled = .T.
    ThisForm.VfarmCntnrViews.VfarmcmdFirst.Enabled = .F.
    ThisForm.VfarmCntnrViews.VfarmcmdNext.Enabled = .F.
    ThisForm.VfarmCntnrViews.VfarmcmdPrev.Enabled = .F.
    ThisForm.VfarmCntnrViews.VfarmcmdLast.Enabled = .F.
    ThisForm.VfarmCntnrViews.VfarmcmdDone.Caption = 'Cancel'
    ThisForm.VfarmCntnrViews.VfarmcmdSave.Enabled = .F.
    ThisForm.VCntnrDBC.Visible = .F.
    ThisForm.VfarmCntnrViews.Visible = .T.
    ThisForm.LockScreen = .F.
    ThisForm.VfarmCntnrViews.VfarmViewName.SetFocus()
    ThisForm.Refresh()
ENDPROC

```

### **Shape3**

### **Vfarmlabel2**

### **Vfarmlabel1**

### **VCBSync**

```

PROCEDURE Click

```



```

ThisForm.LockScreen = .T.
ThisForm.VCntrDBC.VGridAppl.RecordSource = 'CurDummyAppl'
ThisForm.VCntrDBC.VGridAppl.Column1.ControlSource = 'ApplName'
ThisForm.VCntrDBC.VGridDBC.RecordSource = 'CurDummyDBC'

```

```

IF ThisForm.BuildProjectList() == .T.
    ThisForm.BuildControlList()
    ThisForm.RemoveDuplicates()
ENDIF

```

```

ThisForm.OpenDBC()

```

```

SELECT curDBViews
GO TOP

```

```

SELECT viewname FROM ViewAppl INTO CURSOR CurTemp ;
    WHERE ALLTRIM(UPPER(viewname)) NOT IN ;
        (SELECT ALLTRIM(UPPER(ViewName)) FROM CurDBViews)

```

```

SELECT CurTemp
GO TOP
SCAN
    INSERT INTO curDBViews VALUES (CurTemp.viewname, ", .F., .F., .T., .F., RGB(255,0,0))
ENDSCAN

```

```

UPDATE curDBViews SET linAppl = .F., BackColor = RGB(255,255,0) ;
    WHERE ALLTRIM(UPPER(ViewName)) NOT IN ;
        (SELECT ALLTRIM(UPPER(ViewName)) FROM ViewAppl)

```

```

SELECT curDBViews
GO TOP

```

```

SELECT ViewAppl_Distinct
=REQUERY()
GO TOP

```

```

ThisForm.VCntrDBC.VGridDBC.RecordSource = 'curDBViews'
ThisForm.VCntrDBC.VGridAppl.RecordSource = 'ViewAppl_distinct'
*ThisForm.VCntrDBC.VGridAppl.Column1.ControlSource = 'ApplName'

```

```

ThisForm.Refresh()
ThisForm.LockScreen = .F.

```

```

ENDPROC

```

## VCBReport

```

PROCEDURE Click
LOCAL nRecNum, nAnswer

```

```

*!* DO CASE

```

```

** CASE ThisForm.VCntrDBC.DisplayBy.Value == 2
SELECT viewname_distinct
=REQUERY()
GO TOP

nRecNum = RECCOUNT('viewname_distinct')
IF nRecNum > 0
    nAnswer = MESSAGEBOX('Do you wish to view the Report?',36,Thisform.Caption)
    IF nAnswer == 6
        FOR i = 1 TO _Screen.FormCount
            _SCREEN.Forms(i).visible = .F.
        NEXT
        _SCREEN.Top = 0
        _SCREEN.WindowState = 2

        REPORT FORM rvmrpt1 PREVIEW

        _SCREEN.Top = -100
        _SCREEN.Height = 0
        FOR i = 1 TO _Screen.FormCount
            _SCREEN.Forms(i).visible = .T.
        NEXT
    ENDIF

    nAnswer = MESSAGEBOX('SendReport to printer?',36,Thisform.caption)
    IF nAnswer = 6
        REPORT FORM rvmrpt1 TO FILE NOCONSOLE
        REPORT FORM rvmrpt1 TO PRINTER NOCONSOLE
    ENDIF

    =MESSAGEBOX("Printing completed. ",64,Thisform.Caption)
    ENDIF

    Thisform.WindowState = 0
ENDIF
** CASE ThisForm.VCntrDBC.DisplayBy.Value == 3

** SELECT CurApplByViews
** * =REQUERY()
** GO TOP
** INDEX ON viewname TAG reptag

** nRecNum = RECCOUNT('CurApplByViews')
** IF nRecNum > 0
**     nAnswer = MESSAGEBOX('Do you wish to view the Report?',36,Thisform.Caption)
**     IF nAnswer == 6
**         FOR i = 1 TO _Screen.FormCount
**             _SCREEN.Forms(i).visible = .F.
**         NEXT
**         _SCREEN.Top = 0
**         _SCREEN.WindowState = 2
**     ENDIF
**     REPORT FORM rvmrpt2 PREVIEW

```

```

**          _SCREEN.Top = -100
**          _SCREEN.Height = 0
**          FOR i = 1 TO _Screen.FormCount
**              _SCREEN.Forms(i).visible = .T.
**          NEXT
**      ENDIF
**
**          nAnswer = MESSAGEBOX('SendReport to printer?',36,Thisform.caption)
**          IF nAnswer = 6
**              REPORT FORM rvmrpt2 TO PRINTER NOCONSOLE
**
**              =MESSAGEBOX("Printing completed. ",64,Thisform.Caption)
**          ENDIF
**
**          Thisform.WindowState = 0
**      ENDIF
**  ENDCASE
ENDPROC

```

#### VCBPrintTree

```

PROCEDURE Click
    LOCAL nRecNum, nAnswer
    SELECT vfarmctl
    GO TOP

    nRecNum = RECCOUNT('vfarmctl')
    IF nRecNum > 0
        nAnswer = MESSAGEBOX('Do you wish to view the Class Roster?',36,Thisform.Caption)
        IF nAnswer == 6
            FOR i = 1 TO _Screen.FormCount
                _SCREEN.Forms(i).visible = .F.
            NEXT
            _SCREEN.Top = 0
            _SCREEN.WindowState = 2

            REPORT FORM rvmsearch PREVIEW

            _SCREEN.Top = -100
            _SCREEN.Height = 0
            FOR i = 1 TO _Screen.FormCount
                _SCREEN.Forms(i).visible = .T.
            NEXT
        ENDIF

        nAnswer = MESSAGEBOX('Send Search Report to printer?',36,Thisform.caption)
        IF nAnswer = 6
            REPORT FORM rvmsearch TO PRINTER NOCONSOLE

            =MESSAGEBOX("Printing completed. ",64,Thisform.Caption)
        ENDIF
    ENDIF

```

```

        Thisform.WindowState = 0
    ENDIF

```

```

ENDPROC

```

#### VCBQuit

```

PROCEDURE Click
    Thisform.Release()
ENDPROC

```

#### vcboDBC

```

PROCEDURE InteractiveChange
    Thisform.OpenDBC()

    Thisform.VCntnrDBC.VGridDBC.SetFocus()
ENDPROC

```

#### VCbuttonDBC

```

PROCEDURE Click
    LOCAL cPath, cOldDir, nX

    cOldDir = "
    IF Thisform.vcboDBC.Listindex > 0
        cPath = ALLTRIM(Thisform.vcboDBC.List(Thisform.vcboDBC.Listindex))
        nX = RAT("\", cPath)
        IF nX > 0
            cPath = SUBSTR(cPath, 1, nX)
            cOldDir = SET('DEFAULT') + SYS(2003)
            SET DEFAULT TO (cPath)
        ENDIF
    ENDIF
    cPath = ALLTRIM(LOWER(GETFILE("dbc", "Open.DBC")))
    IF !EMPTY(cOldDir)
        SET DEFAULT TO (cOldDir)
    ENDIF

    IF !BLANK(cPath)
        RETURN
    ELSE
        IF ! Thisform.SetDBCPath(cPath, Thisform.vcboDBC)
            Thisform.AddDBCPath(cPath, Thisform.vcboDBC)
            Thisform.SetDBCPath(cPath, Thisform.vcboDBC)
        ENDIF

        Thisform.OpenDBC()
    ENDIF
ENDPROC

```

VIBIDBC

## REFERENCES

- [1] Booch, Grady. 1994. Object-Oriented Analysis and Design with Applications, Second Edition. Adison-Wesley Publishing Company.
- [2] Whil Hentzen, "Data-Driving Applications with DBCx", Visual FoxPro DevCon 97. On-Line. Dialog. 18 Sep. 1997
- [3] Long, Jeb, Visual FoxPro 3, Developers Guide, Third Edition, Sam Publishing.
- [4] Sanders, E., Brentnall, S. and Gunn, J. 1995. The Visual Guide to Visual FoxPro 3.0, First Edition. Ventana Communication Group.
- [5] Tama E. Granor and Ted Roche, Hacker's Guide to Visual FoxPro 3.0, First edition, Adison-Wesley Developers Press.