FILEMAN DELPHI COMPONENTS (FMDC)

GETTING STARTED GUIDE

Version 1.0

March 1998

Revised December 2004

Department of Veterans Affairs
VistA Health Systems Design & Development (HSD&D)
Development and Infrastructure Support (DaIS)

Revision History

Documentation Revisions

The following table displays the revision history for this document. Revisions to the documentation are based on patches and new versions released to the field.

Date	Revision	Description	Author
03/1998	1.0	Initial FileMan Delphi Components (FMDC) V. 1.0 software documentation creation.	FMDC Development Team, Oakland, CA OIFO
12/14/04	2.0	Reformatted manual to follow ISS Style Guide. Also, updated outdated information (e.g., Web site references). In addition, recreated PDF document to meet 508 compliance requirements.	REDACTED

Table i: Documentation revision history

Patch Revisions

For a complete list of patches related to this software, please refer to the Patch Module on FORUM.

Contents

Figures and Tables

Orientation

How to Use this Manual

Throughout this manual, advice and instructions are offered regarding the use of FileMan Delphi Components (FMDC) software and the functionality it provides for Veterans Health Information Systems and Technology Architecture (VistA) software products.

This manual uses several methods to highlight different aspects of the material:

- Various symbols are used throughout the documentation to alert the reader to special information. The following table gives a description of each of these symbols:

Symbol	Description
ℹ	Used to inform the reader of general information including references to additional reading material.
⚠	Used to caution the reader to take special notice of critical information.

Table ii: Documentation symbol descriptions

- Descriptive text is presented in a proportional font (as represented by this font).
- HL7 messages, "snapshots" of computer online displays (i.e., roll-and-scroll screen captures/dialogues) and computer source code, if any, are shown in a *non*-proportional font and enclosed within a box.
  - User's responses to online prompts will be boldface type. The following example is a screen capture of computer dialogue, and indicates that the user should enter two question marks:

```
Select Primary Menu option: ??
```

  - The "**<Enter>**" found within these snapshots indicate that the user should press the Enter key on their keyboard. Other special keys are represented within < > angle brackets. For example, pressing the PF1 key can be represented as pressing **<PF1>**.
  - Author's comments, if any, are displayed in italics or as "callout" boxes.

ℹ Callout boxes refer to labels or descriptions usually enclosed within a box, which point to specific areas of a displayed image.

- All uppercase is reserved for the representation of M code, variable names, or the formal name of options, field and file names, and security keys (e.g., the XUPROGMODE key).

How to Obtain Technical Information Online

Exported file, routine, and global documentation can be generated through the use of Kernel, MailMan, and VA FileMan utilities.

Methods of obtaining specific technical information online will be indicated where applicable under the appropriate topic. Please refer to the *FileMan Delphi Components (FMDC) Technical Manual* for further information.

Help at Prompts

VistA software provides online help and commonly used system default prompts. Users are encouraged to enter question marks at any response prompt. At the end of the help display, the user is immediately returned to the point from which he/she started. This is an easy way to learn about any aspect of VistA software.

To retrieve online documentation in the form of Help in any VistA character-based product:

- Enter a single question mark ("**?**") at a field/prompt to obtain a brief description. If a field is a pointer, entering one question mark ("**?**") displays the HELP PROMPT field contents and a list of choices, if the list is short. If the list is long, the user will be asked if the entire list should be displayed. A **YES** response will invoke the display. The display can be given a starting point by prefacing the starting point with an up-arrow ("^") as a response. For example, **^M** would start an alphabetic listing at the letter M instead of the letter A while **^127** would start any listing at the 127th entry.
- Enter two question marks ("**??**") at a field/prompt for a more detailed description. Also, if a field is a pointer, entering two question marks displays the HELP PROMPT field contents and the list of choices.
- Enter three question marks ("**???**") at a field/prompt to invoke any additional Help text stored in Help Frames.

Obtaining Data Dictionary Listings

Technical information about files and the fields in files is stored in data dictionaries. You can use the List File Attributes option on the Data Dictionary Utilities submenu in VA FileMan to print formatted data dictionaries.

For details about obtaining data dictionaries and about the formats available, please refer to the "List File Attributes" chapter in the "File Management" section of the *VA FileMan Advanced User Manual*.

Assumptions About the Reader

This manual is written with the assumption that the reader is familiar with the following:

- VistA computing environment
- VA FileMan data structures and terminology
- Microsoft Windows
- M programming language

It provides an overall explanation of configuring the FileMan Delphi Components (FMDC) software. However, no attempt is made to explain how the overall VistA programming system is integrated and maintained. Such methods and procedures are documented elsewhere. We suggest you look at the various VA home pages on the World Wide Web (WWW) for a general orientation to VistA. For example, go to the Veterans Health Administration (VHA) Office of Information (OI) Health Systems Design & Development (HSD&D) Home Page at the following Web address:

http://vista.med.va.gov/

Reference Materials

Readers who wish to learn more about the FileMan Delphi Components (FMDC) software should consult the following:

- *FileMan Delphi Components Installation Guide*
- *FileMan Delphi Components Technical Manual and Security Guide*
- *FileMan Delphi Components Getting Started Guide* (this Manual)
- *FileMan Delphi Components Help File*—Provides complete information on the FileMan Delphi Components (FMDC), including full listings of each component's methods and properties.

  For more information, please refer to the "FMDC.HLP Help File" chapter in this manual.

- The FileMan Delphi Components (FMDC) Home Page at the following Web address:

  http://vista.med.va.gov/fmdc/index.asp

  The FileMan Delphi Components (FMDC) Web site provides up-to-date information including Frequently Asked Questions (FAQs), troubleshooting tips, and any code or documentation links/updates.

- *VA FileMan Programmer Manual*—Provides complete information on the VA FileMan Database Server (DBS) API. The FMDC data access components are wrappers around calls in the VA FileMan Database Server (DBS) API. Thus, having a DBS reference can be handy when you're working with data access components. An online version of this documentation is available at the following Web address:

  http://vista.med.va.gov/fileman/docs/pm/index.asp

VistA documentation is made available online in Microsoft Word format and Adobe Acrobat Portable Document Format (PDF). The PDF documents *must* be read using the Adobe Acrobat Reader (i.e., ACROREAD.EXE), which is freely distributed by Adobe Systems Incorporated at the following Web address:

http://www.adobe.com/

VistA documentation can be downloaded from the Enterprise VistA Support (EVS) anonymous directories or from the Health Systems Design and Development (HSD&D) VistA Documentation Library (VDL) Web site:

http://www.va.gov/vdl/

For more information on the use of the Adobe Acrobat Reader, please refer to the *Adobe Acrobat Quick Guide* at the following Web address:

http://vista.med.va.gov/iss/acrobat/index.asp

DISCLAIMER: The appearance of any external hyperlink references in this manual does not constitute endorsement by the Department of Veterans Affairs (VA) of this Web site or the information, products, or services contained therein. The VA does not exercise any editorial control over the information you may find at these locations. Such links are provided and are consistent with the stated purpose of this VA Intranet Service.

1. Introduction

This Getting Started Guide introduces developers to the FileMan Delphi Components (FMDC) Version 1.0. It aims to quickly get you started building Delphi applications that access VA FileMan data.

The FileMan Delphi Components make it easy for developers to work with VA FileMan data in Delphi applications. The components encapsulate the details of retrieving, validating and updating VA FileMan data within a Delphi application. This saves you from creating your own custom remote procedure calls (RPCs) when you need to access VA FileMan data.

The FileMan Delphi Components also include special enhanced features such as complete server-side error checking and data dictionary help.

If you're already familiar with Delphi, the time needed to develop an application to edit a set of VA FileMan fields using the FileMan Delphi Components is comparable to the time needed to create the same application using VA FileMan's roll-and-scroll ScreenMan interface.

The FileMan Delphi Components provide three types of components:

- **Data Access Components** are invisible to the user, but contain the functionality for calling the server to find, retrieve, validate, and file data. Each of the data access components encapsulates the functionality of one or more VA FileMan Database Server (DBS) calls.
- **Custom Dialogues** are like mini-applications you can include in your own application. The TFMLookUp custom dialogue makes it easy to perform lookups in files with large numbers of records.
- **Data Controls** are visual controls users can interact with to change data values. For example, a TFMCheckBox control is good for editing "Boolean" two-value set of codes fields; a TFMMemo control is good for editing word-processing fields; and a TFMEdit control is good for editing free text fields. Data controls are directly populated by the data access components. Values are directly validated and filed from the controls by the data access components.

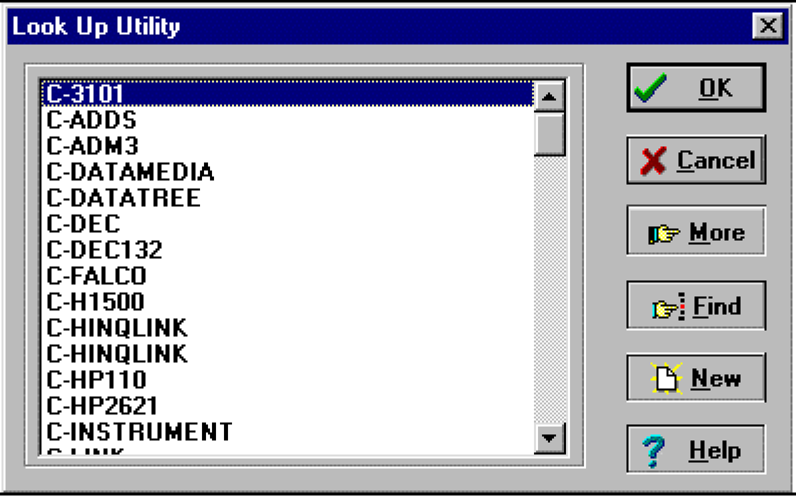
FMDC Data Access Components

Component	Icon	Function
TFMGets	⬇	Encapsulates the Data Retriever (GETS^DIQ) DBS call. Retrieves a record from the server. Populates any associated data controls with field values of the retrieved record.
TFMValidator	☑	Encapsulates the Validator (VAL^DIE) DBS call. Validates a data value against the corresponding VA FileMan field on the server. If any associated data control's coValOnExit flag is True, the TFMValidator automatically validates values entered in the control.
TFMFiler	⬆	Encapsulates the Filer (FILE^DIE) and Updater (UPDATE^DIE) DBS calls. Given a list of data controls with values to file, the TFMFiler collects those values from the controls and files them.

Component	Icon	Function
TFMLister		Encapsulates the Lister (LIST^DIC) DBS call. Retrieves a list of records from the server. Optionally populates listbox-type data controls with the retrieved list.
TFMHelp		Encapsulates the Helper (HELP^DIE) DBS call. Retrieves field-based help from the data dictionary on the server. Can automatically display help for a data control's field in a panel, whenever a user sets focus on a data control.
TFMFinder		Encapsulates the Finder (FIND^DIC) DBS call. Finds one or more records in a file that match a lookup value. Also used for lookups by TFMComboBoxLookUp and the TFMLookup custom dialogue.
TFMFindOne		Encapsulates the Single Record Finder ($$FIND1^DIC) DBS call. Finds a unique record in a file based on a lookup value; not linked with data controls.

Table 1-1: FMDC data access components

FMDC Custom Dialogues

TFMLookUp The TFMLookUp custom dialogue makes it easier to do lookups in files with large numbers of records.	

Look Up Utility

```
C-3101
C-ADDS
C-ADM3
C-DATAMEDIA
C-DATATREE
C-DEC
C-DEC132
C-FALCO
C-H1500
C-HINQLINK
C-HINQLINK
C-HP110
C-HP2621
C-INSTRUMENT
```

Table 1-2: FMDC custom dialogues

FMDC Data Control Components

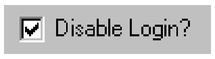
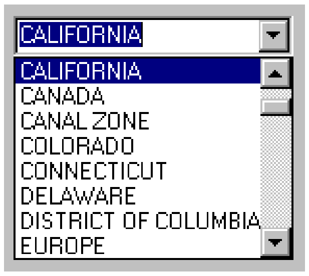
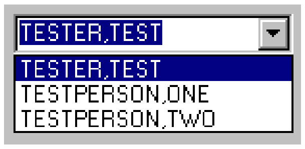
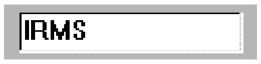
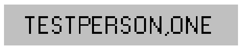
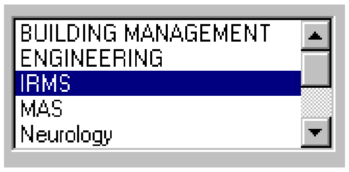
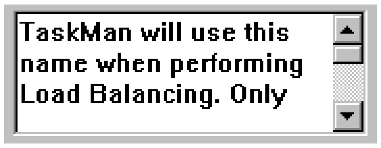
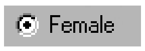
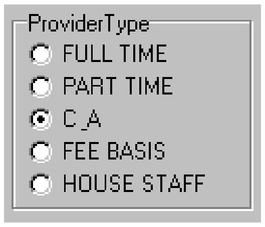
Component	Icon	Example	Use For
TFMCheckBox		 ☑ Disable Login?	"Boolean" two-value yes/no set of codes fields.
TFMComboBox		 CALIFORNIA CALIFORNIA CANADA CANAL ZONE COLORADO CONNECTICUT DELAWARE DISTRICT OF COLUMBIA EUROPE	Pointer fields, record lookups.
TFMComboBoxLookUp		 TESTER,TEST TESTER,TEST TESTPERSON,ONE TESTPERSON,TWO	Pointer fields, record lookups. Does on-the-fly lookups of what the user types in; good for longer lists.
TFMEdit		 IRMS	Free Text, Numeric, Date and MUMPS fields.
TFMLabel		 TESTPERSON,ONE	Computed fields, Read-only field values.
TFMListBox		 BUILDING MANAGEMENT ENGINEERING IRMS MAS Neurology	Pointer fields, record lookups.
TFMMemo		 TaskMan will use this name when performing Load Balancing. Only	Word-processing fields.
TFMRadioButton		 ● Female	Set of codes fields.
TFMRadioGroup		 ProviderType ○ FULL TIME ○ PART TIME ● C A ○ FEE BASIS ○ HOUSE STAFF	Set of codes fields.

Table 1-3: FMDC data control components

FMDC Object Hierarchy

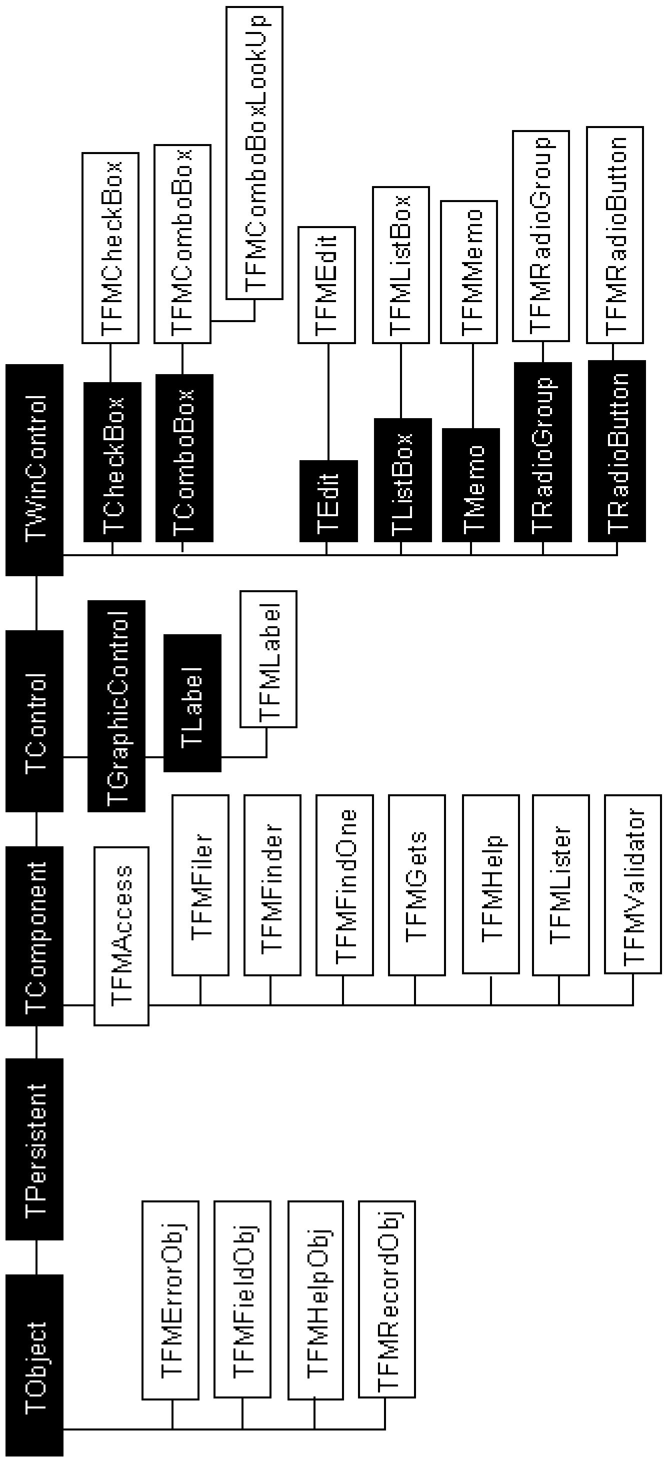


Figure 1-1: FMDC object hierarchy

2. Quick Start Guide

This Quick Start Guide demonstrates the basic approach to editing data with the FileMan Delphi Components. This approach only uses a subset of the properties, methods and components in the FileMan Delphi Components.

There are 17 FileMan Delphi Components, and each one has a variety of methods and properties that allow you to fine-tune its behavior. However, a basic approach to using them, involving only a subset of their properties and methods, is appropriate for most data editing situations.

To follow this basic approach: First, determine the set of fields you want to edit in your Delphi application. Then follow this Quick Start Guide to provide that access in your Delphi application.

To edit records in a given VA FileMan file with the FileMan Delphi Components:

1. Establish an RPC Broker connection:
    a. Add a **TRPCBroker** component to your form.
    b. Set its properties and invoke its methods as necessary to connect to a server system.
2. Add a TFMGets component to retrieve data:
    a. Add a **TFMGets** component to your form.
    b. Set its **RPCBroker** property to point to your form's TRPCBroker component.
    c. Set its **FileNumber** property to the file containing records to retrieve.
3. Add a TFMFiler component to file changes:
    a. Add a **TFMFiler** component to your form.
    b. Set its **RPCBroker** property to point to your form's TRPCBroker component.
4. Add a TFMValidator component to provide validation services:
    a. Add a **TFMValidator** component to your form:
    b. Set its **RPCBroker** property to point to your form's TRPCBroker component.

5. Add VA FileMan data controls for each field:
   a. For each field to edit, add data control(s) and supporting data access components to your form as follows:

For this Field Type:	Add to Your Form:
Free Text, Numeric, Date	1 TFMEdit
"Boolean" Set of Codes	1 TFMCheckBox
Set of Codes	1 TFMRadioGroup, or
	1 TPanel or TGroupBox, plus 1 TFMRadioButton per code
Word-processing	1 TFMMemo
Pointer	1 TFMLister and 1 TFMListBox, or
	1 TFMLister and 1 TFMComboBox, or
	1 TFMLister and 1 TFMComboBoxLookUp
Computed	1 TFMLabel

Table 2-1: Field types and data controls on a form

   b. For each field that you add component(s) for, set the field-type-specific properties of the components according to the guidelines (listed by field type) in the "How To: By VA FileMan Field Type" chapter in this manual.
6. Select and retrieve a record:
   a. To select a record, follow the procedure in the "Selecting a Record" section below in the "How To: By Task" chapter. You'll add a TFMLookUp and TFMLister component to your form, and add a button that calls TFMLookup's Execute method to perform the lookup.
   b. TFMLookUp.Execute returns a record number. Using it you can retrieve the record and populate your data controls with the record's field values. To retrieve the record, follow the procedure in the "Retrieving a Record" section below in the "How To: By Task" chapter. You'll call TFMGets' GetAndFill method to retrieve the record and populate data controls.

c. The OnClick event handler for the button that executes the TFMLookup.Execute method (Step #6a) can also perform the retrieval (Step #6b). Code to do this would look like:

```
procedure TForm1.Button1Click(Sender: TObject);
var AddRecord:Boolean;
begin
  if FMLookup1.Execute(AddRecord) then begin
    FMGets1.IENS:=FMLookUp1.RecordNumber+',';
    // Call any TFMListBox/TFMComboBox GetList methods
    // here, before calling GetAndFill.
    FMGets1.GetAndFill;
  end
  else
    ShowMessage('No record chosen.');
end;
```

Figure 2-1: TFMLookUp—Sample OnClick even handler code

Compile your application. You can now retrieve records.

7. Set Up Automated OnExit Processing:

   a. Your data controls should already be linked to a TFMFiler and a TFMValidator component, from following the "Data Control Property Settings for All Field Types" guidelines in the "How To: By VA FileMan Field Type" chapter in this manual.

   b. Set every data control's **coValOnExit** value to True, in each control's **FMCtrlOptions** property.

About Automated OnExit Processing

This feature automatically validates a field value in a data control when a user changes it (this is the control's OnExit event). If the changed value is valid, automated OnExit processing adds the control to the associated TFMFiler's component's list of controls to file.

Compile your application. All changes to fields in VA FileMan data controls will be validated, and only accepted by the data controls if valid.

8. Provide an event to save changes:

   a. To save changes the user makes to the record, follow the procedure in the "Saving a Record" section below in the "How To: By Task" chapter. You'll add a button whose caption is something like "Save Changes". You'll add code for this button's OnClick event handler that calls your TFMFiler's Update method to file changes.

Compile your application. You can now file changes to the record

9. (optional) Provide context-sensitive field help.

   There are several ways you can provide context-sensitive help for the VA FileMan fields being edited:

   - The first line of the DD help for a field can automatically be displayed in a display panel, whenever a user sets focus to a control.
   - All DD help for a field can be displayed when, with a particular control selected, the user presses F1.
   - Standalone help: you can retrieve DD help for any field and display it using your own methods.

   For more information on providing context-sensitive help, please refer to the "FMDC.HLP Help File" chapter in this manual.

10. Register your application:

    a. Create a "B"-type option in the Option file for your application.

    b. In the option's RPC multiple, include every Remote Procedure Call (RPC) your application calls.

       You need to include all RPCs invoked by methods of the FileMan Delphi Components called by your application, as well as RPCs you invoke yourself. The FMDC.HLP online help file details which FMDC RPCs are called by FMDC component methods.

    c. In your application's OnCreate event, register the option name using the broker's CreateContext method. If registration fails, your application should probably terminate. For example:

```
if not RPCBroker1.CreateContext('A6A APP1')
then Application.Terminate;
```

    d. Users must have the registered "B"-type option assigned to them in order to use your client application.

Bypass Security During Development.

Possessing the XUPROGMODE key allows you as a developer to bypass RPC Broker security.

Once you're ready to deploy your application to non-developer users, your application will need to register itself appropriately.

For more information on RPC Broker security, please refer to the RPC Broker documentation.

Compile your application. Users without the XUPROGMODE key should now be able to run your application.

The FMDC data access components are wrappers around calls in the VA FileMan Database Server (DBS) API. So having a DBS reference can be handy when you're working with data access components. For a complete DBS reference, please refer to the *VA FileMan Programmer Manual*. Online versions of this documentation are available on the VDL:

http://www.va.gov/vdl/Infrastructure.asp?appID=5

3. How To: By VA FileMan Field Type

This chapter shows which VA FileMan data controls are compatible with which VA FileMan field types, and how to set the properties of the data controls.

For each VA FileMan data control you use on your form to edit a particular VA FileMan field type, follow the corresponding procedure in this chapter to set the control's properties.

VA FileMan Field Types and Compatible Controls

Field Type	Compatible Controls
Computed	TFMEdit, TFMLabel
Date	TFMEdit, TFMLabel
Free Text	TFMEdit, TFMLabel
MUMPS	TFMEdit, TFMLabel
Numeric	TFMEdit, TFMLabel
Pointer	TFMComboBox, TFMComboBoxLookUp, TFMListBox, TFMEdit, TFMLookUp custom dialogue
Set of Codes	TFMCheckBox, TFMRadioButton, TFMRadioGroup, TFMEdit
Variable Pointer	(must be done manually)
Word-processing	TFMMemo

Table 3-1: VA FileMan field types and compatible controls

Data Control Property Settings for All Field Types

Set these properties for every VA FileMan data control you use.

1. Set the **FMField** and **FMFile** properties to the field and file that the control is to edit.
2. Set the **FMGets** property to the TFMGets to use to retrieve values.
3. Set the **FMValidator** property to the TFMValidator to use to validate (except for TFMMemo controls, which do not require validation).
4. Set the **FMFiler** property to the TFMFiler to use to file changes.
5. (optional) Set the **FMCtrlOptions** coValOnExit flag to True to enable automated OnExit processing. For more information, please refer to the "Providing Automated OnExit Processing for Controls" topic.

Hint

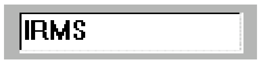
In Delphi, to set a property for a set of controls to the same value, select all of the components simultaneously (Hold down the Ctrl key and select each control).

Then, in Delphi's Object Inspector, in a single edit of that property you set the property value for all selected controls.

Free Text, Numeric, Date, MUMPS: TFMEdit

1. Add a **TFMEdit** component to your form.
2. Set its properties as described in the "Data Control Property Settings for All Field Types" topic above.

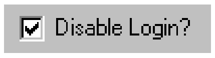
TFMEdit:



"Boolean" Set of Codes (2 Yes/No Values): TFMCheckBox

1. Add a **TFMCheckBox** component to your form.
2. Set its properties as described in the "Data Control Property Settings for All Field Types" topic above.
3. Set its **FMValueChecked** and **FMValueUnchecked** properties to the two internal codes (from the VA FileMan data dictionary) that should be represented by the checked and unchecked states of the control.
4. Set the **Caption** property to the label to display to the end-user (typically the external value represented by the "True" internal code).

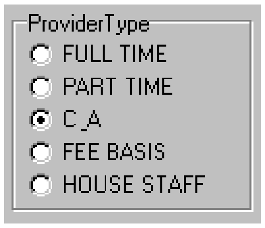
TFMCheckBox:



Multi-value Set of Codes: TFMRadioGroup

1. Add a **TFMRadioGroup** component to your form.
2. Set its properties as described in the "Data Control Property Settings for All Field Types" topic above.
3. Populate its **Items** property (an array of TStrings) with the values to display to the user for each code.
4. Populate its **FMInternalCodes** property with the **internal** values (from the VA FileMan data dictionary) of the codes in the set of codes field. Populate it in the same sequence, line-by-line and code-by-code, with which you populated the Items property.

TFMRadioGroup:



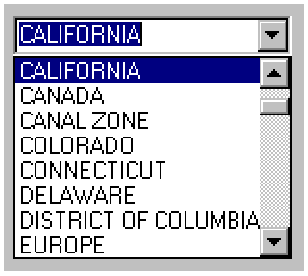
Multi-value Set of Codes: TFMRadioButton

TFMRadioButton:

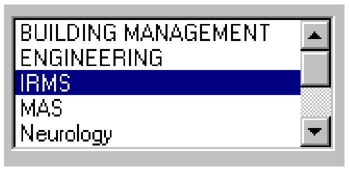
1. Place a **TPanel** or **TGroupBox** panel component on your form.
2. Place one **TFMRadioButton** per code directly from the component palette onto the TPanel or TGroupBox. For example, for a 3-value set of codes field, place 3 TFMRadioButtons directly from the component palette onto the TPanel or TGroupBox.
3. Set each TFMRadioButton's properties as described in the "Data Control Property Settings for All Field Types" topic above.
4. Set each TFMRadioButton's **FMValueChecked** property to the internal code (from the VA FileMan data dictionary) that should be represented by the checked state of the control.
5. Set each TFMRadioButton's **Caption** property to the label to display to the end-user (typically the external value represented by each code).

Pointer: TFMComboBox, TFMListBox

TFMComboBox:



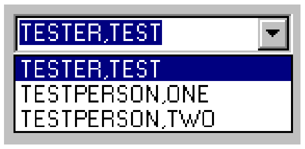
TFMListBox:



1. Add a **TFMComboBox** or **TFMListBox** to your form. These work best when the set of records is not huge.
2. Set its properties as described in the "Data Control Property Settings for All Field Types" topic above.
3. Add a **TFMLister** component to your form. It must be dedicated entirely to servicing the pointer field.
4. Set the TFMLister's **FileNumber** property to that of the **pointed-to** file. Make sure its **FieldNumbers** property contains at least ".01". Set its **RPCBroker** property to your form's TRPCBroker.
5. Associate the TFMListBox or TFMComboBox with the TFMLister through the **FMLister** property.
6. To retrieve the current value of the pointer field: **First** use the control's GetList method to populate its Items property with pointed-to file's list of records. **Then** use the GetAndFill method of the associated TFMGets component to select the current pointer value from the list of possible values.

Pointer: TFMComboBoxLookUp

TFMComboBoxLookUp:



1. Add a **TFMComboBoxLookUp** to your form. This control performs lookups "on-the-fly" (the entire list of possible records does not have to be retrieved into the control).
2. Set its properties as described in the "Data Control Property Settings for All Field Types" topic above.
3. Add a **TFMLister** component to your form. It must be dedicated entirely to servicing the pointer field.
4. Set the TFMLister's **FileNumber** property to that of the **pointed-to** file. Make sure its **FieldNumbers** property contains at least ".01". Set its **RPCBroker** property to the TRPCBroker component on your form.
5. Optionally set the TFMLister's **Number** property to restrict the number of records returned in any one lookup when the user enters "**?**" or leaves the edit box null and presses the down-arrow. This enables a "<<< More >>>" item at the end of the Items list so the user can request more records if needed.
6. Associate the TFMComboBoxLookUp with the TFMLister through its **FMLister** property.
7. You don't need to populate the TFMComboBoxLookUp's Items property prior to calling GetAndFill, because lookups are performed on the fly.
8. Because TFMComboBoxLookUp performs server-side calls automatically you application will need to register the DDR LISTER and DDR FINDER RPCs.

For more information on registering RPCs, please refer to the RPC Broker documentation on the VDL:

http://www.va.gov/vdl/Infrastructure.asp?appID=23

Pointer: TFMLookUp

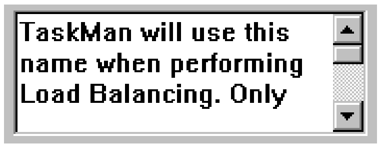
You can also edit pointer fields with the TFMLookUp custom dialogue. This can be useful if the pointer field points to a file with a huge number of records; the TFMLookUp custom dialogue simplifies lookups in large files.

Use a VA FileMan data control such as **TFMLabel** or **TFMEdit** to display the current value of the pointer field. If you use TFMEdit you should set its ReadOnly property set to True. Then a button you place next to the control could invoke the **Execute** method of **TFMLookUp** to edit the value of the pointer field.

For a complete list of steps and a code sample, please refer to the online FMDC.HLP help file.

Word-processing: TFMMemo

TFMMemo:


TaskMan will use this name when performing Load Balancing. Only

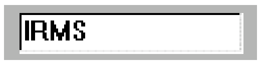
1. Add a **TFMMemo** component to your form.
2. Set its properties as described in the "Data Control Property Settings for All Field Types" topic above.
3. Make sure you set its **FMFile** property to the file number of the file containing the word-processing field, and **FMField** to the word-processing field number. Don't use the subfile number of the word-processing field.

Computed: TFMLabel or TFMEdit

TFMLabel:

TESTPERSON,ONE

TFMEdit:


IRMS

1. Add a **TFMLabel** or **TFMEdit** component to your form.
2. Set its properties as described in the "Data Control Property Settings for All Field Types" topic above.
3. Unlike other field types, no automated OnExit processing or filing is needed for computed fields since they are read-only.
4. If you are using a TFMEdit control, set its **ReadOnly** property to True, so users don't think that they can edit the field (and so that filing is never attempted).

4. How To: By Task

This chapter demonstrates how to accomplish the most common VA FileMan record-editing tasks.

Selecting a Record

The procedure below describes how to select a record with the TFMLookUp custom dialogue.

For additional ways to select records, please refer to the FMDC.HLP help file.

To select a record with a TFMLookUp component:

1. Add a **TFMLister** component to your form.
2. Set the TFMLister's **RPCBroker** property to point to your form's TRPCBroker component.
3. Set the TFMLister's **FileNumber** property to the file to do the lookup in.
4. Optionally set the TFMLister's **Number** property to restrict the number of records returned when the TFMLookUp window is first opened (particularly for files with large numbers of records). Setting this property enables the TFMLookUp More button so that the user can request more records if needed.
5. Add a **TFMLookUp** component to your form.
6. Set the TFMLookUp's **FMLister** property to the TFMLister.
7. Add a button to your form so the user can do a lookup. Set the button's **Caption** to something the user will recognize, e.g. "Choose User" if you're doing a lookup in the NEW PERSON file (#200).

   The button's OnClick event should call the **Execute** method of the TFMLookUp component to do the lookup. It can make the retrieved record number, if any, available to a TFMGets component so that the TFMGets is ready to retrieve the record (see the "Retrieving a Record" topic). For example:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  AddRecord :Boolean;
begin
  FMLookUp1.AllowNew:=False; // disallow adding records
  if FMLookUp1.Execute(AddRecord) then
    TFMGets1.IENS:=FMLookUp1.RecordNumber+','
  else
    ShowMessage('You did not select a record.');
end;
```

Figure 4-1: Sample of code from the TFMLookUp Execute method

Retrieving a Record

To retrieve a record from the server, use the TFMGets component. Its GetAndFill method retrieves fields from a record on the server and populates a set of associated data controls with the retrieved values.

The following should be set prior to calling the code example below:

- TFMGets.FileNumber should be set to the file from which to retrieve the record.
- All data controls should be associated with the TFMGets component (through their FMGets properties).
- All data controls' FMFile and FMField properties should be set to reflect the file and field number that they should be editing.

The following code retrieves a record and populates the associated controls with appropriate field values:

```
// Set TFMGets.IENS to IENS (#+',' for top-level records)
// of the record to retrieve
FMGets.IENS:=TFMLookup1.RecordNumber+',';
// if you have TFMListBox or TFMComboBox controls to get lists for,
// get the lists here before calling GetAndFill.
FMListBox1.Getlist;
// Now you can call TFMGets.GetAndFill.
FMGets1.GetAndFill;
// Check for errors after the server call.
if FMGets1.ErrorList.Count>0 then FMGets1.DisplayErrors;
```

Figure 4-2: Sample code to retrieve a record and populate controls

The Items property of TFMListBox and TFMComboBox controls must be populated **before** calling TFMGets.GetAndFill. Do this by calling the GetList method of the data control before calling GetAndFill. Then GetAndFill can retrieve the current field value, and select the corresponding value in the control's Items property as the control's current value.

About Error Checking

Error checking is provided for each of the data access components. Whenever the data access components make a server call (invoking an M routine on the server) there is the potential for an error to occur on the server side, due to a variety of conditions in the server environment.

You should check a data access component's ErrorList.Count property after making a server call to see if a server-side error occurred. If any did, you can call the component's DisplayErrors method to display those errors to the user (typically the error number and text from a DBS error is what is displayed), and then branch accordingly to gracefully handle the error condition.

Providing Automated OnExit Processing for Controls

Each time a user enters a data value in a data control, there is a set of tasks you would typically perform:

1. Check if the user changed the data value.
2. If the control's FMRequired property is True, and the user deleted the data value, warn the user that the value is required, and reset the control's value to its previous state.
3. Otherwise, validate the changed value (usually needed only for TFMEdit controls):
    a. If the value is invalid, return the control's value to the original and provide the user help on why the value was invalid.
    b. If the value is valid, update the control's display value from what user typed to validated external form of field value, if necessary.
4. If the value is valid, update the control's FMCtrlInternal and FMCtrlExternal properties to reflect the internal and external VA FileMan forms of the control's value (except for TFMMemo controls).
5. Unless the changed value is invalid, tell the associated TFMFiler that the control has data to file (with the TFMFiler's AddChgdControl method).

A good place to perform these actions is in a control's OnExit event handler. Coding these actions by hand for each of your data controls would be laborious. Fortunately, the VA FileMan data controls have an Automated OnExit processing option you can turn on to perform each of these tasks automatically.

To turn on Automated OnExit Processing:

1. Make sure each of your VA FileMan data controls is linked to a TFMFiler component through their **FMFiler** properties.
2. Make sure each of your VA FileMan data controls is linked to a TFMValidator component through their **FMValidator** properties (except TFMMemo, which does not need validation).
3. Set every data control's **coValOnExit** value in the **FMCtrlOptions** property to True.

When Control Values are changed by Code

If a control's value is changed by code (rather than through user interaction) the OnExit event of the control is not triggered. Automated OnExit processing is not triggered either. In this case, your code will need to perform any actions that you would ordinarily expect to be performed by automated OnExit processing.

Saving a Record

To save changes to the database, use the TFMFiler component. You can add filing requests with its AddFDA and AddChgdControl methods, and file all pending requests with its Update method.

To request filing:

- Call a TFMFiler's AddFDA method to request filing a standalone field value (not attached to a control).
- Call a TFMFiler's AddChgdControl method to request filing a field value in a VA FileMan data control.

Hint

If you are using automated OnExit processing, the AddChgdControl method is automatically called when a user changes the value in a VA FileMan data control to a valid new value.

To file all accumulated filing requests to the database, call the TFMFiler's Update method.

To save accumulated filing requests:

1. Provide some event that the user can trigger to save changes. The **OnClick** event of a "Save Changes" button is one good place.
2. Provide code for this event that will save the changes to the server. This code should:
   a. (optional) Check if there are any data updates waiting to be filed (TFMFiler's **AnythingToFile** method). Because the Update method also calls AnythingToFile, your code only needs to call AnythingToFile if that code needs to know if changes are waiting to be filed.

   b. If you designated any controls as required with the FMRequired property, call the TFMFiler's **DataProblemCheck** method to check that all required controls have values.

   c. If no problems are found, call the TFMFiler's **Update** method to file the accumulated changes.

   d. If Update fails, call the TFMFiler's **DisplayErrors** method to display error information to the end-user.

   For example:

```
if FMFiler1.AnythingToFile then
  if FMFiler1.DataProblemCheck then
    FMFiler1.ProcessDataProblemList
  else
    if FMFiler1.Update then
      ShowMessage('Changes filed!')
    else
      FMFiler1.DisplayErrors
else
  ShowMessage('No changes to file!');
```

Figure 4-3: TFMFiler—Sample code calling DisplayErrors method

Editing Records from Several Files Simultaneously

Previous chapters describe how to edit a set of fields from a single VA FileMan file. Often, however, you may need to edit sets of fields from several related records in different VA FileMan files simultaneously. To do this:

Data Controls	Set each data control's FMFile and FMField property to reflect the file and field to edit.
TFMGets	Use one TFMGets component for each file you need to retrieve records for. Set the FMGets property of each data control to the TFMGets component you're using to retrieve records for that file.
TFMValidator	Use one TFMValidator component for all data controls, regardless of file and field being edited.
TFMFiler	If the set of fields from several files is to be filed simultaneously, and there are no dependencies between records, use a single TFMFiler to file all fields for one or more files. Otherwise, use a separate TFMFiler for each file you need to save records to. Set the FMFiler of each data control to the TFMFiler component you're using to save records for that file.

Other "How To" Tasks

Some other tasks within the scope of editing VA FileMan data in Delphi applications are not addressed in this Getting Started Guide. For help on these and other issues, please refer to the online FMDC.HLP help file:

- Selecting records with TFMListBox, TFMComboBox, TFMComboBoxLookUp, or TFMFindOne.
- Retrieving records with TFMFinder.
- Retrieving and filing data that is in multiples.
- Providing manual OnExit processing for VA FileMan data controls.
- Providing context-sensitive DD field help for data controls.
- Adding new records using VA FileMan controls or TFMLookUp.
- Deleting Records.

5. Using Data Access Components Directly

Previous chapters in this guide demonstrate using the VA FileMan data access components in conjunction with the VA FileMan data controls.

You can also use the VA FileMan data access components directly to retrieve, validate and file VA FileMan data, without involving any VA FileMan data controls.

TFMGets: Retrieving a Record

You can call the TFMGets.GetData method to retrieve one or more field values from a specified record. The field values are returned in the TFMGets.Results property as follows:

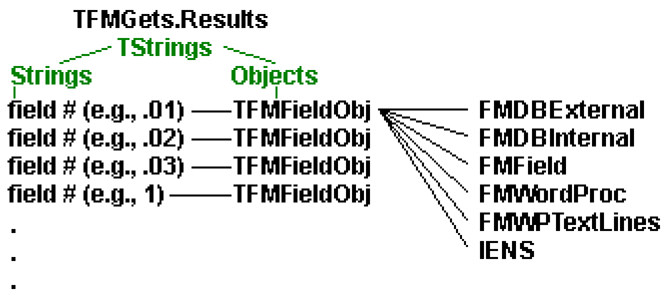


Figure 5-1: TFMGets sample returned data

Each field is retrieved into a TFMFieldObj object, whose structure is:

Object	Description
FldObj.IENS	string (IENS of record).
FldObj.FMField	string (field number)
FldObj.FMDBExternal	string (external field value if requested)
FldObj.FMDBInternal	string (internal field value if requested)
FldObj.FMWordProc	boolean (if True, field is a word-processing field)
FldObj.FMWPTextLine	TStrings (Word-processing field lines)

Table 5-1: TFMFieldObj—Structure description

To retrieve a record without populating VA FileMan data controls:

1. Add a **TFMGets** component to your form.
2. Set its **RPCBroker** property to your form's TRPCBroker component.
3. Set its **FileNumber** property to the file containing records to retrieve.

4. Set its **FieldNumbers** property to contain all fields to retrieve. You can set the property directly or use the **AddField** method.
5. Set its **IENS** property to the IENS of the record to retrieve.
6. Call the **GetData** method to retrieve the record into the Results property.
7. Access each field's retrieved TFMFieldObj object with the **GetField** method. For example:

```
MyFldObj:=FMGets1.GetField('.01');
```

TFMLister: Retrieving a List of Records

You can call the TFMLister.GetList method directly to retrieve a set of records. The set of records is returned in the TFMLister's Results property, which is a TStrings object. The format of returned data is as follows:

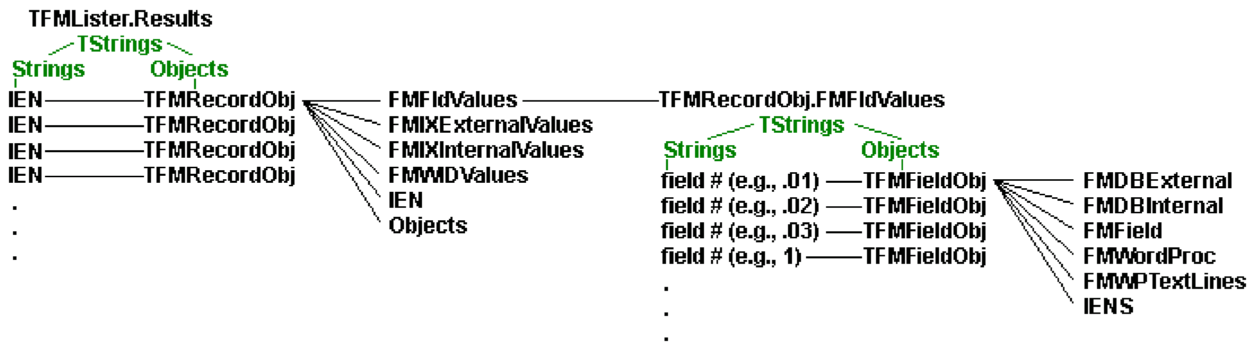


Figure 5-2: TFMLister—Sample returned data

The structure of each returned TFMRecordObj is:

Object	Description
RecObj.IEN	IEN of the record.
RecObj.FMFldValues	TFMFieldObj objects for fields requested in FieldNumbers.
RecObj.FMIXExternalValues	external form of index value(s) (TFMLister only) if requested.
RecObj.FMIXInternalValues	internal form of index value(s) (TFMLister only) if requested.
RecObj.FMWIDValues	Write Identifier of a file and the Identifier parameter result.
RecObj.Objects	Place to associate your own object.

Table 5-2: TFMRecordObj—Structure description

To retrieve a list of records with TFMLister:

1. Add a **TFMLister** component to your form.
2. Set its **RPCBroker** property to the TRPCBroker component your form is using.
3. Set the TFMLister's **FileNumber** property to the file or subfile number to retrieve records from. If retrieving from a subfile, use the **IENS** property to indicate the full path to the subfile.
4. Set the **FieldNumbers** property to the fields to retrieve with each record.
5. To use any of the optional parameters for the LIST^DIC call that affect how records are retrieved, set the corresponding property in the TFMLister component (ListerFlags, FMIndex, Identifier, PartList, and Screen).
6. Set any of the options controlling the TFMLister component with the **ListerOptions** property.
7. You can limit the number of records returned in any one call with the **Number** property. If you need to retrieve more records, you'll need to use the TFMLister's **GetMore** method.
8. Invoke the TFMLister component's **GetList** method to retrieve records. The list of records returned in the **Results** property of the TFMLister. If you pass a TStrings object as a parameter to the GetList method, that list is also populated.
9. You can access the TFMRecordObj object and TFMFieldObj objects returned for a particular record (based on IEN) with the GetRecord and GetField functions as follows:

```
MyRecordObj:=TFMLister1.GetRecord(IEN);
MyFieldObj:=MyRecordObj.GetField('.01');
MyFieldExternal:=MyFieldObj.FMDBExternal;
```

TFMValidator: Validating a Standalone Value

You can use the TFMValidator directly to validate any given value in the context of a given file, field number and IENS.

To validate a standalone value:

1. Add a **TFMValidator** component to your form.
2. Set its **RPCBroker** property to the TRPCBroker component your form is using.
3. Set the TFMValidator component's **FieldNumber**, **FileNumber**, **IENS**, and **Value** properties directly to reflect the value to validate, and the context in which to validate it. **Value** should be in the form as would be input by a user (not internal form!). Alternatively you can call the TFMValidator's **Setup** method to set these properties.
4. Call the **Validate** method of the TFMValidator. Results of the validation are returned in TFMValidator's **Results** property, in the format:

Results[0]:	Internal VA FileMan form of value if input Value was valid, otherwise "^".
Results[1]:	External VA FileMan form of value if input Value is valid.

TFMFiler: Filing Standalone Values

You can use the TFMFiler directly to file any given value to a specified VA FileMan file, field and record.

To file standalone values:

1. If you want to pre-validate the value to file, follow the steps in the "Validating a Standalone Value" section above.
2. Use the TFMFiler's **AddFDA** method to request filing for a given value. You pass as parameters to this method the values that would comprise a VA FileMan Database Server (DBS) FDA: FileNumber, IENS, FieldNumber and Value (internal). For example:

```
FMFiler1.AddFDA('49','+1,','.01',NewName);
```

Figure 5-3: TFMFiler—AddFDA method

Call the **Update** method of the TFMFiler to file the value. This method returns True if filing was successful, or False if one or more errors were encountered on the server. You can use the return value as follows in code to branch to an error handler:

```
if FMFiler1.Update then
  ShowMessage('Changes filed!')
else
  FMFiler1.DisplayErrors;
```

Figure 5-4: TFMFiler—Sample code branching to error handler

TFMFiler: Adding a Record

There are several ways you can add a new record to a file. This section describes how to add a record using the TFMFiler.AddFDA method.

For information on other ways to add records, please refer to the online FMDC.HLP help file.

To add a new record to a file using TFMFiler.AddFDA:

1. Get values from the user for the record's .01 field, and for any required identifier fields.
2. Pre-validate each field value. To do this, set the **FieldNumber**, **FileNumber**, and **Value** properties of a TFMValidator component for each field directly (or use its **Setup** method). For validation, **Value** should be in the form it would be input as a user (not internal form!) Set the **IENS** property to the IENS including placeholder (e.g.,'+1,') for the new record. Call the TFMValidator 's **Validate** method. This gives you the external and internal VA FileMan forms of each field value. Ask for the value again if it fails validation.
3. Use a TFMFiler component's **AddFDA** method to add VA FileMan Data Arrays (FDAs) (File, record number, field and value) for these fields to the TFMFiler component's list of fields to be updated. **Value** should be in internal VA FileMan format (obtained from the validation step above). For the record number for each field, use an identical placeholder (e.g. '+1,').
4. When you're ready to file the new record, call the TFMFiler component's **Update** method. This adds the new record.
5. To find the actual IEN assigned to the new record, call the TFMFiler component's **FindIEN** method using the new record's placeholder. If the return value is null, the record was not created (and an error would have been returned by the Update call).

The following example adds a new record and displays the IEN of the added record:

```
procedure TForm1.AddRecord(Sender: TObject);
  var NewName, IEN: String;
begin
  NewName:= InputBox('Add a New Service', 'Service Name: ', '');
  FMValidator1.Setup('49','+1,','.01',NewName);
  If not FMValidator1.Validate then
    FMValidator1.DisplayErrors
  else begin {file}
    FMFiler1.AddFDA('49','+1,','.01',NewName);
    if not FMFiler1.Update then FMFiler1.DisplayErrors;
    //get IEN of new record}
    IEN:=FMFiler1.FindIEN('+1,');
    if IEN <> '' then FMGets1.IENS:=IEN+',';
  end; {file}
end;
```

Figure 5-5: Adding a new record

Once you have the IEN for the new record you've created, you could use a TFMGets component to populate a set of VA FileMan data controls with the new record's values.

6. FMDC.HLP Help File

For complete information on the FileMan Delphi Components, including full listings of each component's methods and properties, please refer to the FMDC.HLP help file provided with the FileMan Delphi Components.

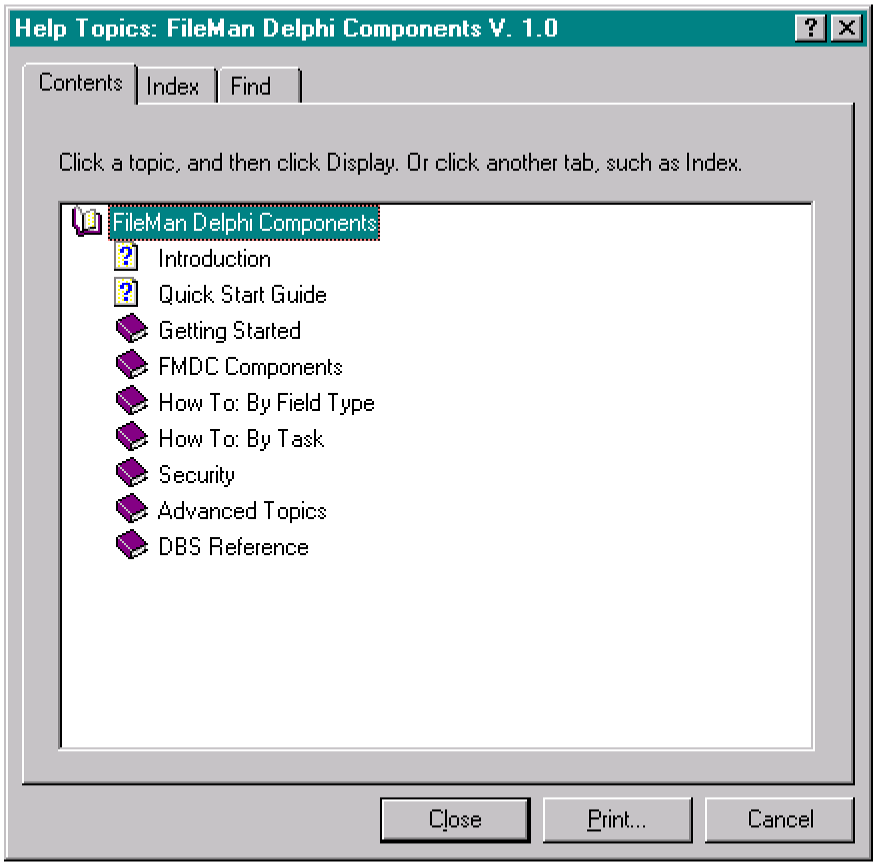


Figure 6-1: FMDC online help file directory tree

FMDC.HLP Help File as Context-Sensitive Help

You can integrate the FMDC.HLP help file with Delphi's help. This means that you can select a FileMan Delphi Component on a Delphi form, or a FileMan Delphi Component property in Delphi's Object Browser, press F1, and automatically access the corresponding help file topic from FMDC.HLP.

For more information integrating the FMDC help file with Delphi, please refer to the *FileMan Delphi Components Installation Guide*.

Accessing the FMDC.HLP Help File Directly

You can invoke the FMDC.HLP file directly by making a Windows shortcut or desktop icon for it. Your shortcut or icon should load the copy of the FMDC.HLP file that you place in your DELPHI\HELP directory during installation.

Index