

InfoPower™

Developer's Guide

Copyright ©2017 Woll2Woll Software, all rights reserved.

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from Woll2Woll Software.

InfoPower is a trademark of Woll2Woll Software. Delphi is a trademark of Inprise Corporation. Other brand and product names are trademarks or registered trademarks of their respective holders.

Woll2Woll Software
3150 Reed Ave.
Livermore, CA 94550 U.S.A.
Voice:(925) 371-1663
<http://www.woll2woll.com>
sales@woll2woll.net

InfoPower VCL License Agreement

By using the software product (“InfoPower VCL”) contained in this package, you agree to the terms and conditions of this license agreement.

Permission is given to the licensee (“you”) of this product to use the development version of this software under Delphi or C++ Builder on one computer at a time, and to make one backup copy. Similarly, if the InfoPower source code is purchased, permission is given to the licensee to use the source code under Delphi or C++ Builder on one computer at a time. You may utilize and/or modify this product for use in your compiled applications. You may distribute and sell any product, which results from using this product in your applications, except a product of similar nature. You may NOT redistribute any source code that may be included with this product.

This product is sold “as is”, without warranty, implied or expressed. While every effort is made to insure that this product and its documentation are free of defects, Woll2Woll Software shall not be held responsible for any loss of profit or any other commercial damage, including, but not limited to special, incidental, consequential or other damages occasioned by the use of this product.

Additional Source Code Restrictions:

If you purchased the optional InfoPower source code...

You **may** use InfoPower components and the related source code to create new components for use within your company or to create a Windows program (executable file created by Delphi). The resulting .EXE file, and .bpl run-time packages may be distributed via freeware, shareware or any commercial means of sale or distribution, but you must **not** include any other InfoPower file with your distribution media.

You may **not** create new components for distribution outside of your company, via freeware, shareware or any commercial product offering, based on any InfoPower component, unless those using your new component also have purchased an InfoPower license.

Woll2Woll Software reserves the right to modify or remove any function, procedure or property, that is not documented in this InfoPower Developer’s Guide, in future releases of the InfoPower component library. This includes modifying the number and/or type of parameters passed to un-documented functions or procedures.

Woll2Woll Software is not responsible for, nor can we provide technical support for, your use of any un-documented InfoPower function, procedure or property. You assume full responsibility for supporting your resulting code and component(s) as well as the results of your using any undocumented function, procedure or property.

Technical Support Options

Before contacting us for technical support, please take some time to carefully search the manual and on-line help for the information, including the troubleshooting section. Make sure that you are asking a specific question about InfoPower instead of a general Delphi question. Also be sure to check the useful sites at <http://www.codenewsfast.com/> and <http://www.mers.com/searchsite.html>, as they contain a database of InfoPower newsgroup threads as well as all other Delphi related newsgroups.

When you need to contact us, please post your questions into our newsgroup. Also review the messages already asked on the forum to see if your question has been asked before. On the Internet, you can find our newsgroup by clicking on the MessageBoard link located at <https://groups.google.com/forum/?fromgroups#!forum/woll2wollinfopower>.

In some cases it may be necessary to email us a simple project that shows us the problem you are having. If you need to do this then please follow these recommendations:

1. Make your project as simple as possible so that we are not debugging your code but instead are helping you with the proper way to use the components. In general try to get your project down to one form, and remove all the extraneous objects and code.
2. When packaging your files for email delivery, use pkzip to compress your files into one .ZIP file.
3. Email to techsupport@woll2woll.com

Our newsgroups are the fastest way to obtain technical support as it allows us to efficiently obtain all the necessary information to solve your problems.

If you need to call technical support, *you will need to supply us your InfoPower registration number.*

Internet WWW Site: <http://www.woll2woll.com>

Newsgroup: Click on link located at <http://www.woll2woll.com>

Internet Technical Support e-mail address: support@woll2woll.com

Contents

Introducing InfoPower	3
Before You Begin	3
What's Included in the Developer's Guide?.....	4
What is InfoPower?.....	5
Installing InfoPower	7
InfoPower Requirements.....	7
Installation Steps.....	8
Uninstalling InfoPower	12
Compatibility issues between InfoPower and previous versions of InfoPower.....	12
Distributing applications which use the InfoPower components.	12
Building packages that use the InfoPower components.	14
InfoPower Component Overview	15
InfoPower Sample Projects.....	15
Complete InfoPower Component Hierarchy	15
Programming with InfoPower	19
Overview.....	19
Getting Help	20
Using the Optional InfoPower Source Code	20
What's new in InfoPower	21
InfoPower Database Architecture.....	21
Adding Custom Framing & Transparency.....	21
Using the Select Fields Dialog Box.....	26
Using InfoPower's Picture Masks.....	32
Determining the object names of the controls contained in an InfoPower dialog	41
Using XP Themes with InfoPower	42
InfoPower Component Reference	43
Description of Reference	43
TwwCheckBox	44
TwwClientDataset	49
TwwController	51
TwwDataInspector	52
TwwInspectorCollection.....	72

TwwInspectorItem	74
TwwDataSource.....	81
TwwDBComboBox	82
TwwDBComboDlg.....	93
TwwDBDateTimePicker.....	96
TwwDBEdit	101
TwwDBGrid	105
TwwDBLookupCombo	157
TwwDBLookupComboDlg.....	168
TwwDBMonthCalendar	175
TwwDBNavigator	181
TwwDBRichEdit, TwwDBRichEditMSWord.....	187
TwwDBSpinEdit.....	206
TwwExpandButton.....	209
TwwFilterDialog.....	214
TwwIncrementalSearch	232
TwwIntl	235
TwwKeyCombo	239
TwwLocateDialog	241
TwwLookupDialog.....	246
TwwMemoDialog.....	253
TwwQBE	258
TwwQuery	264
TwwRadioButton.....	266
TwwRadioGroup	268
TwwRecordViewDialog	272
TwwRecordViewPanel	286
TwwSearchDialog.....	293
TwwStoredProc	301
TwwTable	303
Troubleshooting	311
Index.....	316

Introducing InfoPower

With the assistance of this InfoPower *Developer's Guide*, you will learn what InfoPower is, how to install the InfoPower components into your Delphi/C++ Builder development environments, how to access InfoPower's demonstration forms, what each of the InfoPower components is and most importantly, how to use these powerful components in your Windows applications.

Before You Begin

This guide was written with several assumptions in mind: First, that you understand how to use the Microsoft Windows environment. For help with Windows, please refer to your printed Windows documentation and on-line help files. Next, that you have a basic understanding of relational databases, database design and data management, along with a working knowledge of the specific Database Management System (DBMS) your application will be accessing, such as Paradox, dBase, Oracle, Sybase, InterBase and others. For information related to creating a client/server application with Delphi, please refer to Delphi's documentation.

Last, that you have a basic understanding of Delphi terminology and the application development techniques covered in your Delphi manuals. The Delphi-specific topics you should be familiar with include:

- ◆ Creating and managing projects.
- ◆ Creating new forms (data entry/edit windows) and managing units (source code files).
- ◆ Working with data-aware components and their associated properties and events.
- ◆ Writing simple Object Pascal source code.
















What's Included in the Developer's Guide?


















The InfoPower *Developer's Guide* is comprised of the following six main chapters:

1. **Introduction** Description of InfoPower, its requirements and how you and your end-users benefit when InfoPower components are included in your Delphi or C++ Builder based Windows applications.
2. **Installation** Complete installation instructions. Compatibility issues between previous versions of InfoPower. Building and distributing packages that use InfoPower.
3. **Overview** Text and graphic charts showing the architecture of all InfoPower components. Reference to demonstration programs.
4. **Programming** Programming tips and overviews of InfoPower topics.
5. **Reference** Implementation instructions for each InfoPower component, which includes complete descriptions of new properties and events added to each InfoPower component; descriptions of modified events; how-to section; tips section; and Object Pascal source code examples where necessary to help you implement the InfoPower components in your applications.
6. **Troubleshooting** Alphabetical listing of components with descriptions of possible problems and their solutions.

What is InfoPower?

InfoPower is a library of very powerful, data-aware components that are automatically installed into the component palette in its integrated development environment (IDE). The InfoPower components include the following, listed alphabetically:

	TwwCheckBox
	TwwClientDataSet
	TwwController
	TwwDataSource
	TwwDataInspector
	TwwDBComboBox
	TwwDBComboDlg
	TwwDBDateTimePicker
	TwwDBEdit
	TwwDBGrid
	TwwDBLookupCombo
	TwwDBLookupComboDlg
	TwwDBMonthCalendar
	TwwDBNavigator
	TwwDBRichEdit

	TwwDBSpinEdit
	TwwExpandButton
	TwwFilterDialog
	TwwIncrementalSearch
	TwwIntl
	TwwKeyCombo
	TwwLocateDialog
	TwwLookupDialog
	TwwMemoDialog
	TwwRadioGroup
	TwwQBE
	TwwQuery
	TwwRecordViewDialog
	TwwRecordViewPanel
	TwwSearchDialog
	TwwStoredProc
	TwwTable

Installing InfoPower

We've automated the installation of InfoPower as much as possible, but a few manual steps are still required to complete the process *before* you can access the InfoPower components and sample applications provided with InfoPower. Complete instructions for both installing and un-installing InfoPower are provided in this chapter.

InfoPower Requirements

To install the InfoPower component library, your system should already contain a fully functional version of the Delphi 5.0, Delphi 6.0, Delphi 7.0, or C++ Builder 5.0 or 6.0 development environment, contain about 15 MB of free hard disk space. InfoPower does not have any CPU or memory requirements above or beyond those necessary to run Delphi or C++ Builder. However, if you are creating a complex form that contains many components, you may need to increase the stack size of your project. ***We deem the 16K (04000 Hex) default to be inadequate in most cases and strongly recommend that you raise this value to 24K (6000 Hex),*** or up to whatever size is necessary to stop any compiler or runtime errors you might be receiving.

Options | Project | Linker | Min Stack Size 0x00006000

Installation Steps

Installing is accomplished with the following steps:

1. Running the Setup.exe program for your version of Delphi or C++ Builder (C++ support in InfoPower Professional version only)
2. Installing the components or packages into the IDE environment
3. Installing the help files into the IDE environment

1 - Running the SETUP.EXE program for your version of Delphi or C++ Builder

1. Insert the InfoPower CD-ROM into your computer, and then using the Windows Program Manager, or your favorite method of running a Windows program, run the SETUP.EXE program located in the `\Delphi6` directory (for Delphi 7), the `\Delphi6` directory (for Delphi 6), the `\Delphi5` directory (for Delphi 5), or the `\Builder5` directory (for C++ Builder 5), the `\Builder6` directory (for C++ Builder 6) of your CD-ROM
2. Carefully read each screen, including the license agreement, and click *Next* to proceed. When you encounter the *Information* dialog box that is shown in Figure 2.1, enter your name, organization, and registration number. Click the *Next* button to proceed further.

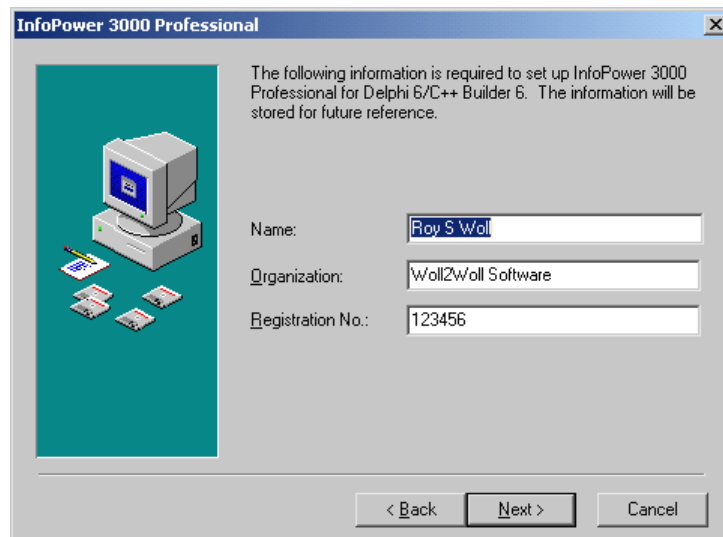


Figure 2.1 - InfoPower's Information dialog box.

Select a directory to place the InfoPower files.

If you want to change the installation directory, then type a new name or click the *Browse* button to select an existing folder.

When you are ready to continue, click on the *Next* button to start the installation process, or click the *Back* button to return to the main installation dialog box. The installation will *automatically* check for available space, and create all the necessary directories and sub-directories, de-compress and copy all requested files from the installation diskette to your hard drive, and then display some additional installation instructions for your viewing.

2 - Installing the components or packages into the IDE environment

Packages are special dynamic-link libraries used by Delphi or C++ Builder applications. They allow code sharing among applications, reducing executable size and conserving system resources. InfoPower supports both design time and runtime packaging options. The following are the steps to install these packages into Delphi and/or C++ Builder.

1. If Delphi/C++ Builder is not currently running, start it now. If Delphi/C++ Builder is currently running, save and close your open project and all related files *before* you proceed.
2. If using Delphi, update the Delphi search path to point to the InfoPower DCU files. If using C++ Builder, skip this step.
 - A. Click on *Tools | Environment Options | Library*.
 - B. Edit the *Directories | Library Path* edit box and add the InfoPower DCU library path. For instance if you installed to c:\ip4000vcl7, you would add c:\ip4000vcl7\lib to the *Library* path edit box. If you wish to debug into the InfoPower source code, then instead add the \ip4000vcl7\source directory path to your Library Path.
3. Installing the design time package - The install program will automatically install the IP4000DCL7.BPL (for Delphi 7), IP4000DCL6.BPL (for Delphi 6, Builder 6), IP4000DCL5.BPL (for Delphi 5, Builder 5), design time packages for you. If for any reason you fail to see the InfoPower components appear in your component palette, then perform the following steps:
 - A. Click on *Project | Options | Packages*
 - B. Click on the *Design Packages | Add* button to add IP4000DCL7.BPL (for Delphi 7, Builder 7), IP3000DCL6.BPL (for Delphi 6, Builder 6) or IP3000DCL5.BPL (for Delphi 5, Builder 5) to your design time package for your project. These files can be found in your \IP4000\package subdirectory.

If you wish to use the InfoPower richedit component integrated with Microsoft's spell checker (TwwDBRichEditMSWord), then you will also need to add one of the following InfoPower packages (see steps above).

IP4000WORDXPVCL7	Delphi 7 with Office Automation XP
IP4000WORD2000VCL7	Delphi 7 with Office Automation 2000
IP4000WORD2000VCL6	Delphi 6 with Office Automation 2000
IP4000WORD2000VCL5	Delphi 5 with Office Automation 2000
IP4000WORDVCL7	Delphi 7 with Office Automation 97
IP4000WORDVCL6	Delphi 6 with Office Automation 97
IP4000WORDVCL5	Delphi 5 with Office Automation 97

If you have previously used the `TwwClientDataSet` component, then you will also need to add the package `IP4000CLIENTVCL7` (Delphi 7), `IP4000CLIENTVCL6` (Delphi 6), or `IP4000CLIENTVCL5` (Delphi 5) to your list of runtime packages. Note that `TwwClientDataSet` is for backwards compatibility with older applications. Newer applications can simply use the `TClientDataSet` component which removes the dependency upon this InfoPower package.

4. Optional and recommended - installing the run time package into Delphi/C++ Builder. This step is required if your applications are using the `IP4000V7` (for Delphi 7, Builder 7), `IP4000V6` (for Delphi 6, Builder 6), or `IP4000V5` (for Delphi 5, Builder 5) run-time packages.
 - A. Click on (Project | Options | Packages).
 - B. Click on the (Runtime Packages | Add button) to add `IP4000V6.DCP` (for Delphi 6, Builder 7), `IP4000V6.DCP` (for Delphi 6, Builder 6), `IP4000V5.DCP` (for Delphi 5, Builder 5) found in your DELPHI or C++ Builder LIB directory, to your runtime time package list for your project.
 - C. Click on the default button in order to make the InfoPower package available to all your projects.

3 - Installing the InfoPower On-line Help Files

To install the InfoPower online help, you will need to run the `OpenHelp` utility by clicking on *Help | Customize* from the Delphi or Builder IDE. From there, click on *Edit | Add Files* to add the `ip4000d7.hlp` (Delphi 7), `ip4000d6.hlp` (Delphi 6), or `ip4000d5.hlp` (Delphi 5) file to the list of help files in the index and link tab pages. After adding the help file, click on *File | Save Project* to save your changes.

Installation Tip

If desired, you can move either the *IP Access*, *IP Controls*, or *IP Dialogs* component palette tabs to a different position from their default installation location via Delphi's Environment Options dialog box.

1. Open the Environment Options dialog box using the following:
Tools | Environment Options
2. Click the Palette tab to display the Pages and Components lists.

3. Click and drag the *IP Access*, *IP Controls*, or the *IP Dialogs* entry displayed in the Pages list to the desired location within the list.
4. Click the OK button to close the dialog box.

Uninstalling InfoPower

Uninstalling InfoPower from the Delphi/C++ Builder can be accomplished by the following:

- A. Close Delphi/C++ Builder if either is open.
- B. Start the Control Panel application from Windows.
- C. Click on the icon labeled Add/Remove Programs
- D. Select InfoPower and click on the add/remove button. If this fails for any reason, then go to the InfoPower directory and run the UNWISE.EXE program found there. Only the files that were installed with the Setup program will be removed

Compatibility issues between InfoPower and previous versions of InfoPower

InfoPower 4000 applications should not require any modifications if previously built with InfoPower 3000. If you are upgrading from a version previous to InfoPower 3000, then note the following compatibility issues:

- TwwDBGrid – InfoPower includes a property PadColumnStyle which defaults to pcsPadHeaderAndData. This removes the whitespace at the bottom and right of the grid if there are not enough records or columns to fill the grid. If you wish to return to the old behavior of InfoPower 2000, then you will need to set the grid's PadColumnStyle to pcsPlain.
- TwwDBComboBox – When the ShowMatchText property is False, InfoPower now adheres to the Windows combobox search behavior where the entered character is used to find a match starting with that one character. Set ShowMatchText to true if you desire continuous incremental searching where all entered characters are used to search the list.

Distributing applications which use the InfoPower components.

If you use the InfoPower runtime packages IP4000V7, IP4000V6, IP4000V5, in your applications, then you will also need to distribute the corresponding files to your customer's computer. We recommend you place these files in your customer's \windows\system directory.

If you are not using these runtime package when building your applications, but instead only the IP4000DCL7, IP4000DCL6, IP4000DCL5, design time package, then you will have no additional distribution requirements beyond what Delphi already requires.

Development system	InfoPower Distributables
Delphi 5 C++ Builder 5	<p>IP4000V5.BPL</p> <p>IP4000CLIENTVCL5.BPL (If using TwwClientDataSet)</p> <p>If you are using the TwwDBRichEditMSWord component with Delphi 5, then please read the following paragraph:</p> <p>We recommend omitting IP4000WORDVCL5 and IP4000WORD2000VCL5 from your list of runtime packages for your project, as this removes the requirement to distribute both the corresponding ip4000word* package as well as the office automation package Dclaxserver50.bpl or Dcloffice2k50.bpl (which are quite large). Omitting the IP4000WORD* runtime packages from your project will add only about 10K to your executable size, and allow you to omit distribution of these related packages. If you do not omit these packages from your runtime package list, then you will need to distribute these files to your end-user systems.</p>
Delphi 6	<p>IP4000V6.BPL</p> <p>IP4000CLIENTVCL6.BPL (If using TwwClientDataSet)</p> <p>If you are using the TwwDBRichEditMSWord component, then please read the following paragraph:</p> <p>We recommend omitting IP4000WORDVCL6 and IP4000WORD2000VCL6 from your list of runtime packages for your project, as this removes the requirement to distribute both the corresponding ip4000word* package as well as the office automation package Dclaxserver60.bpl or Dcloffice2k60.bpl (which are quite large). Omitting the IP4000WORD* runtime packages from your project will add only about 10K to your executable size, and allow you to omit distribution of these related packages. If you do not omit these packages from your runtime package list, then you will need to distribute these files to your end-user systems.</p>
Delphi 7	<p>IP4000V7.BPL</p> <p>IP4000CLIENTVCL7.BPL (If using TwwClientDataSet)</p> <p>If you are using the TwwDBRichEditMSWord component, then please read the following paragraph:</p> <p>We recommend omitting IP4000WORDVCL7, IP4000WORD2000VCL7, and IP4000WORDXPVCL7 from your list of runtime packages for your project, as this removes the requirement to distribute both the corresponding ip4000word* package as well as the office automation package</p>

	Dclaxserver70.bpl, Dcloffice2k70.bpl or Dclofficeexp (which are quite large). Omitting the IP4000WORD* runtime packages from your project will add only about 10K to your executable size, and allow you to omit distribution of these related packages. If you do not omit these packages from your runtime package list, then you will need to distribute these files to your end-user systems.
--	---

Building packages that use the InfoPower components.

If you wish to build your own custom Delphi packages which require the InfoPower component library, then you will need to add the IP4000V5 (Delphi 5 or Builder 5), IP4000V6 (Delphi 6 or Builder 6), or IP4000V7 (Delphi 7 or Builder 7) package to the required section of your package.

InfoPower Component Overview

When possible, each InfoPower component was modeled after one of Delphi's built-in *Data Access* or *Data Control* components by inheriting either the actual Delphi component itself or one of its ancestors. The InfoPower component library was created in this manner for two main reasons: First, to insure that each InfoPower component contains as much of the basic functionality provided by its Delphi ancestor as possible. Second, to keep your InfoPower learning curve as short as possible. In other words, if you are familiar with Delphi's Data Access and Data Control components, you already know the basics of how to add and implement most of the InfoPower components!

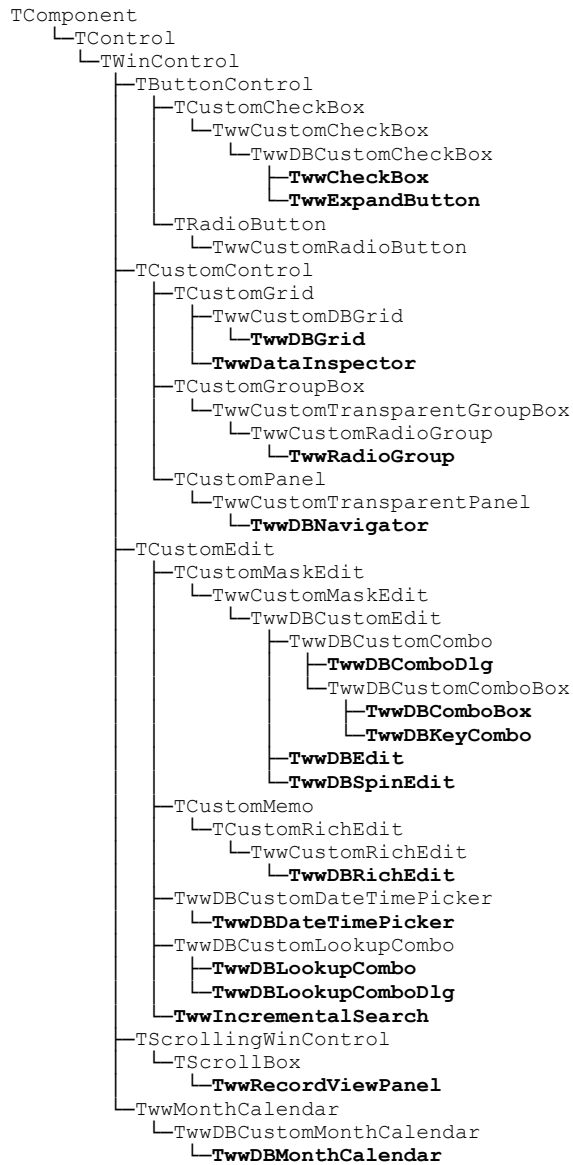
InfoPower Sample Projects

Included with InfoPower are several small Delphi sample units that demonstrate the features and functionality of the InfoPower components. During installation, a subdirectory named DEMOS was automatically created within the InfoPower directory. We recommend you build and run the main demonstration program as it includes all of the InfoPower demos in one project. The main demonstration program is located in your InfoPower sub-directory at `ip3000\demos\maindemo\prjdemo.dpr`.

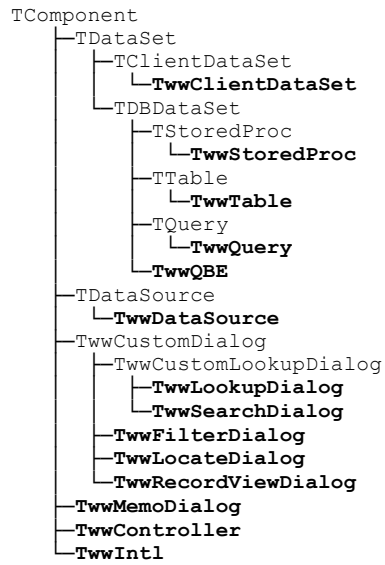
Complete InfoPower Component Hierarchy

The next several pages contain both text-based and graphical hierarchies of the complete InfoPower component library with all Delphi ancestors being shown for each InfoPower component. These hierarchies provide you with a clear guide to all ancestor components so you can obtain information about inherited methods, properties and other component data and behavior. This becomes very important, and a great time saver, if you decide to create some of your own in-house components by inheriting an InfoPower component.

Complete InfoPower Component Hierarchy



Complete InfoPower Component Hierarchy (Continued)



Programming with InfoPower

Overview

This chapter of the InfoPower *Developer's Guide* discusses the details on how to program with InfoPower.

Before jumping into the InfoPower component library, please take a few minutes to familiarize yourself with the following topics, which are presented in this chapter:

- Getting Help
- Using the optional InfoPower source code
- InfoPower Database Architecture
- Adding Custom Framing and transparency to InfoPower's edit controls
- The Select Fields dialog box because it's available from many InfoPower component
- InfoPower's Picture Masks
- Determining the object names of the controls contained in an InfoPower dialog

Getting Help

Windows On-line Help

Accessing on-line help for an InfoPower component or one of its properties is exactly the same as within Delphi—select the component or property you want help with and press F1.

How-To and Tips Sections

Most of the InfoPower component descriptions in this chapter also include *How to* and *Tips* sections. These sections provide very valuable information that could save you many hours of design, creation and debugging headaches, so take advantage of them whenever you can.

Implementation and Coding Examples

When you want a source code example of how to implement one or more InfoPower components, look in this guide's Index under the name of the component you are working with. Then turn to the page number given for the *sample application* entry.

Troubleshooting Section

When you run into problems implementing an InfoPower component, please browse our newsgroups located at <http://www.woll2woll.com>, as well as the troubleshooting chapter at the end of this guide, **before** calling our technical support department. Also be sure to check the useful sites at <http://www.tamaracka.com/search.htm> and <http://www.mers.com/searchsite.html>, as they contain a database of InfoPower newsgroup threads as well as all other Delphi related newsgroups.

The information provided in our newsgroups and Troubleshooting chapter are there to save you time, money and frustration. Please use it wisely.

Exhaustive Index

We put a lot of extra effort into creating the Index section at the back of this guide and hope that most topics you might need to search for are listed there. Please take a moment to browse through the Index to get an idea of it's layout and how it can help you, before you really need it.

Using the Optional InfoPower Source Code

If you purchased the optional InfoPower component library source code, your use of this code is limited by the terms and conditions specified in the InfoPower **License Agreement** which is located at the beginning of this manual. As stated in this agreement, by using this product, you automatically agree to the terms and conditions specified therein.

Your educational benefit of the source code depends upon your interest and knowledge of the Delphi language. However the source code is invaluable if you run into a problem and need to trace into the InfoPower source to determine the cause.

From time to time, you may be tempted to modify one of the existing InfoPower components to meet some specific need you have. However, resist this temptation with all your might because we cannot provide technical support to you if you have modified the InfoPower component source code in any way. In addition, you would not be able to install any InfoPower maintenance or upgrade releases from us since your modified source code would be *overwritten* with these new releases.

Rather, if you need to create a new component for use within your organization that is based on one of the InfoPower components, we suggest that you do one of the following:

1. Inherit the InfoPower component in your program and modify it as necessary.
2. If substantial internal code changes are necessary, create your own *new* component: Copy all of the necessary source code files to new file names in a new directory, rename the component internally, rewrite the registration section accordingly and then finally modify the component code to meet your specific needs.

Another need for acquiring the InfoPower source code is to make InfoPower compatible with other third-party components or database drivers. Since both InfoPower and these other third-party components are being constantly enhanced, it's a good idea for you to contact both Woll2Woll Software *and* the creator of your other third-party product to find out exactly what source code changes to either product may be necessary for them to work together.

What's new in InfoPower

See the WhatsNew4000.htm file in your \ip4000 directory for information on the new features of InfoPower.

InfoPower Database Architecture

Prior to InfoPower, you could not directly use the native datasets provided by Delphi, nor could you directly use 3rd party datasets. You were forced to use descendents of these. InfoPower now allows you directly use these Delphi datasets as well as any TDataSet descendent. For instance to use a Delphi TADOTable, you simply drop a TDataSource and TADOTable into your form, and the InfoPower components can connect directly to it. In the same way, you use the Delphi InterBase objects, such as TIBTable or TIBQuery.

You still may wish to use the InfoPower TDataSet descendents as they do provide some additional conveniences and functionality. For instance, they allow you to store your picture masks in the dataset, instead of the visual control. See the documentation on TwwTable, TwwQuery, TwwStoredProc, TwwClientDataSet, and TwwQBE for details on the InfoPower datasets. However you are no longer required to use these.

Adding Custom Framing & Transparency

InfoPower gives you the means to create elegant forms that look just like the real hardcopy form they are based on. Each control's transparent and custom framing effects can even

display underline controls that are transparent. However the custom framing goes far beyond simple underline controls as you can display the borders in many different frame styles. You can also set different frame styles for when the control has focus and when it doesn't. You can additionally disable any edge from being displayed. See the demo in the \ip4000\demos\framing directory.

Paul Coffey
2222 E. 55th Street
San Jose, CA 93116

1001

DATE 08/02/1999

PAY TO THE ORDER OF John A. Williams

22.55

Twenty Two Dollars and Fifty Five Cents DOLLARS

FOR: Stereo Cables
Includes \$10 Rebate

:145681 :146 :1001 Signature

Form displayed as a check, using InfoPower's transparent edit controls, custom framing, and custom button effects.

Components that support custom framing and transparency

The following InfoPower components support picture, custom framing, transparency, and button effects (where applicable): TwwCheckBox, TwwDBEdit, TwwDBComboBox, TwwDBComboDlg, TwwDBSpinEdit, TwwIncrementalSearch, TwwKeyCombo, TwwDBDateTimePicker, TwwDBLookupCombo, TwwDBLookupComboDlg, TwwDBRichEdit, TwwRadioGroup, TwwRecordViewDialog, and TwwRecordViewPanel.

New in InfoPower 4000 - Controller to centralize your framing properties

InfoPower 4000 introduces a new controller to centralize your framing settings for your InfoPower or 1stClass edit controls. By using a controller, you can modify the framing properties of the controller and have all the edit controls attached to this controller reflect its property settings.

To use this new controller, drop a TwwController component into your application, and set the controller property for each edit control. If you decide to modify the controller properties during program execution, then you will need to call the ApplyFrame method in order for the edit controls to reflect the changes.

Key properties and events for custom framing support

The following properties are new in InfoPower to support the custom framing, transparency, and special button effects. Frame is a property available to all the InfoPower edit controls. In

the RecordView controls, the name of the property is EditFrame. The following details each sub-property of Frame.

Frame or EditFrame

Enabled

Set to True to enable the custom frame or transparency effects. If this property is false, then the other properties below will not function.

AutoSizeHeightAdjust

When an edit control's AutoSize is set to True, InfoPower computes what it deems the most appropriate height for an edit control. You can set this property to adjust the resulting height of the control. For instance a value of 1 will cause the control to be 1 pixel larger than the value that InfoPower computes.

FocusBorders

Selects which borders are displayed when the control has focus.

NonFocusBorders

Selects which borders are displayed when the control does not have focus.

FocusStyle

Select the frame style when the control has focus.

NonFocusStyle

Selects the frame style when the control does not have focus

NonFocusTextOffsetX, NonFocusTextOffsetY

Use these properties to customize the painting of the text when the control does not have focus. You should only override these properties if you do not like the default placement of the painted text

NonFocusColor

Set this property to change the background color of the control when it does not have focus. Use the *Color* property if you wish to change the color of the control when it has the focus. If this property is set to clNone, then the color property is used to paint the background when it does not have the focus.

NonFocusFontColor

Set this property to change the text color of the control when it does not have focus. You may wish to set this property so that the text of the control stands out when it does not have focus. This property is particularly useful when you have enabled transparency, and the control's font color is not legible with the background. By assigning the font to a color that is contrasted well with the background will enable your user's to clearly see the text when it does not have the focus.

NonFocusTransparentFontColor

This property is maintained for backwards compatibility. For newer applications, instead use the *NonFocusFontColor* property.

Set this property to change the text color of the control when it does not have focus. You may wish to set this property so that the text of the control stands out when it does not have focus. If you instead set the control's *font.color* property, the text color will be the same whether or not the control has focus. This could cause your text to disappear when your control receives focus as the control paints the background instead of being transparent. Thus you should set this property instead when using transparent controls.

MouseEnterSameAsFocus

Now in InfoPower, you can set this property to true to enable the control's borders to paint as if they had focus when the mouse is moved over the control. This gives a pleasing visual effect similar to Microsoft Office controls. Set *FocusStyle* to *efsFrameSunken* with all borders and set *NonFocusStyle* to *efsFrameBox* with no Borders to achieve this effect. For optimal display may also wish to set the *NonFocusTextOffsetX* to 2 and the *NonFocusTextOffsetY* to 1.

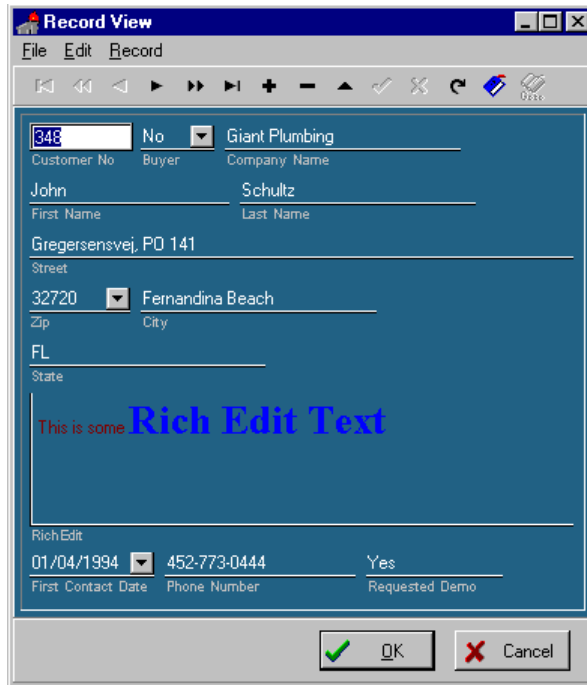
Transparent

This property causes the control to display itself transparently when it does not have the focus. The net effect is that you will see the background painted behind the control. Set this property to *True* if you wish to see the background when the control does not have the focus.

Restrictions: The background must be painted by a non-windows control (not derived from *TWinControl*), such as a Delphi *TImage* or the *TfcImager* (from Woll2Woll's 1stClass product). There may be some painting side effects when using a *TWinControl* to paint the background. You should only set this to *True* if you have a background painted by a *TControl*, not a *TWinControl*.

RecordViewDialog and RecordViewPanel custom framing

The InfoPower *RecordViewDialog* and *RecordViewPanel* components also support custom framing. You can set the *EditFrame* property of these controls to determine the default framing style for its contained edit controls. You may also wish to set the property *Options | tvoLabelsBeneathControls* to *True* when your custom frame appears as an underline control, as this provides for a more natural look.



Use the *OnSetControlEffects* event to override the RecordView's *EditFrame* settings for an individual or selected control. For instance the following code in this event will place a left-border when the edit control is tied to a TBlobField.

```

procedure TForm1.wwRecordViewDialog1SetControlEffects(
  Form: TwwRecordViewForm; curField: TField; Control: TControl;
  Frame: TwwEditFrame; ButtonEffects: TwwButtonEffects);
begin
  if curfield is TBlobfield then
  begin
    Frame.NonFocusBorders:=
      Frame.NonFocusBorders + [efLeftBorder];
  end
end;

```

Key properties for enabling custom button effects in the edit controls.

The following properties are new in InfoPower to support the custom button effects in controls that display a button next to the edit control. These include the following controls: TwwDBComboBox, TwwDBComboDlg, TwwDBDateTimePicker, TwwDBLookupCombo, TwwDBLookupCombo, TwwDBLookupComboDlg, TwwDBSpinEdit. You can also enable these effects in the recordview controls by using the *OnSetControlEffects* event.

ButtonEffects

Transparent

Set to True to enable the button to be displayed transparently so that the control's background is used as the button's background.

Flat

Set to True to enable the button to be normally painted without the borders. The borders are painted when the mouse moves over the button.

Using the Select Fields Dialog Box

The Select Fields dialog box, as shown in Figure 4.1, is available within most of the InfoPower components. This dialog box allows you to select the fields you want displayed in the associated visual interface components, as well as how you want the field to be displayed and edited. The dialog box can be opened by double-clicking a data-aware InfoPower component, or by clicking the “...” button of an InfoPower component's *ControlType*, *PictureMasks*, or *Selected* property.

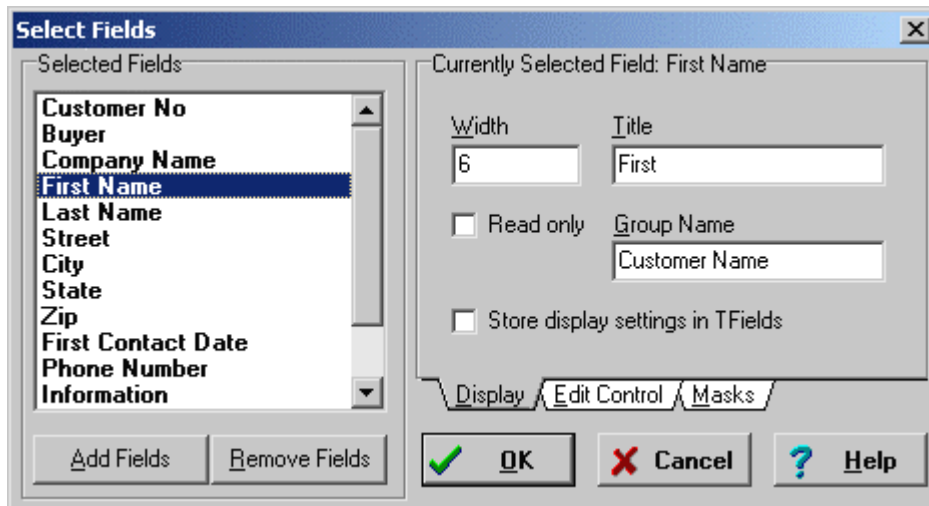


Figure 4.1 - InfoPower's Select Fields dialog box.

The Select Fields dialog box contains a scrollable list of *Selected Fields* along with the tab-controlled *Currently Selected Field* section. The *Selected Fields* list box displays a list of fields that are currently selected for display in the associated component.

Adding Fields

To select additional fields for display, click on the *Add Fields* button. This brings up the Add Fields Dialog box.



Figure 4.2 - InfoPower's Add Fields dialog box.

The Available Fields list box displays a list of fields available from the associated table. The contents of this list is determined by which fields were selected for retrieval from the associated table via the Delphi Fields editor window (opened by double-clicking a table or query component). The default display sort order for the Available Fields list is by Field Name. To change the display sort order of the Available Fields list to that of the field's position within the table, check the Sort By Table Field Order check box.

To add one or more fields for display, you can use one of the following methods.

- ◆ To add several fields, not necessarily in sequence, click the first field name selection within the Available fields list box that you want to add and then use Ctrl+click (hold the Ctrl key down and click the left mouse button) to make additional selections. Then click the *OK* button.
- ◆ To select an entire set of fields in sequence for adding, click the first selection and then use Shift+click (hold the Shift key down and click the left mouse button) on the last sequential field you want added. All fields in between are automatically selected for adding. Then click the *OK* button.

Removing Fields

To remove one or more fields for display, you can use one of the following methods:

- ◆ To remove several fields, not necessarily in sequence, click the first field name selection within the list box that you want removed and then use Ctrl+click (hold the Ctrl key down and click the left mouse button) to make additional selections. Then click the *Remove Fields* button.
- ◆ To select an entire set of fields in sequence for removal, click the first selection and then use Shift+click (hold the Shift key down and click the left mouse button) on the last sequential field you want removed. All fields in between are automatically selected for removal. Then click the *Remove Fields* button.

The order in which fields are displayed as columns in an InfoPower grid or record-view is based on their order in the Selected Fields list. The first field in the list is displayed in column position one, the second field in the list is displayed in column position two, and so forth. To change the column in which a field is displayed in the component, change its position in the Selected Fields list by clicking and dragging the field name to the desired position. The field name is inserted immediately above the field name you release the mouse button over.

The Currently Selected Field section of the Select Fields dialog box contains four tabbed *pages*: Display Attributes, Edit Control, Masks and Links. The Links tab only is visible when the currently selected field is a calculated field. However the Links tab is only retained for backwards compatibility, as you should use Delphi's lookupfields for new projects. To switch between the pages, click on the appropriate tab. Each of these pages is described in the following paragraphs.

Display Attributes

The Display Attributes page allows you to modify the number of characters to be displayed in the field via the Width edit box and the title to be displayed as the column heading via the Title edit box (see Figure 4.3). The default value for Width is the length of the field, as defined in the table, or the number of characters in the Title, whichever is greater. The default value for Title is the name of the field as defined in the table. When either the Field Width or Field Title edit boxes has focus (the edit box value is highlighted or the I-beam cursor is located in the edit box), you can press the up or down cursor arrow key on your keyboard to move to the previous or next Selected Field.

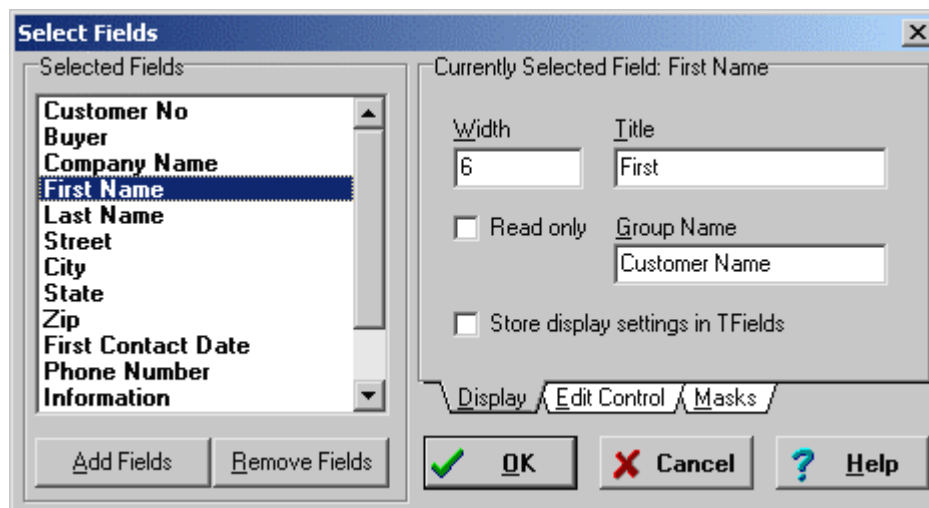


Figure 4.3 - The Currently Selected Field section of the Select Fields dialog box with the Display Attributes tab activated.

When you need to change field-specific attributes, such as data display alignment within a grid cell, display format, edit mask, and others, you must use the Object Inspector. If the field is not currently listed in the Object Inspector, it must first be added via Delphi's Fields editor window, which is opened by double-clicking the associated table or query component. Please refer to your Delphi manuals or on-line help for more information about using the Fields editor.

Read only: Set this to True if you wish for the column to be read-only

Group Name: Assign a group name property if you wish for the grid to hierarchically represent a field title. For instance, if you have two fields "First Name" and "Last Name", and wish to group them together, then assign their GroupName property to "Name". **Note:** You must set UseTFields to false in order to assign the GroupName property and make certain that your fields that you are grouping together are right next to each other.

Store display settings in TFields: When you want the field display properties to be stored into the DataSet's TField properties, then click on the "Store display settings in TFields" checkbox. If you do not want this behavior, but instead want the field display properties to be stored with the component, then uncheck this checkbox.

You might want to uncheck this when you are simultaneously displaying more than one grid attached to the same TwwTable. This allows each grid to display different fields. This property correlates to the UseTFields property of the TwwDBGrid.

Note: In general we recommend setting UseTFields to false, as when this property is true certain grid functionality is disabled (i.e. ProportionalColWidths). The default is set to True to maintain backwards compatibility.

Edit Control

The Edit Control page allows you to modify the type of control, or component, used to display the field's data (see Figure 4.4). Your options include Field, Bitmap, CheckBox, ImageIndex, RichEdit, or CustomEdit. The default value for Edit Control is Field, which is a normal edit control. The CheckBox, Bitmap, ImageIndex, RichEdit, URL-Link, and CustomEdit are very useful when used from within an InfoPower TwwDBGrid or TwwRecordViewDialog components, as discussed in the TwwDBGrid section later in this chapter.

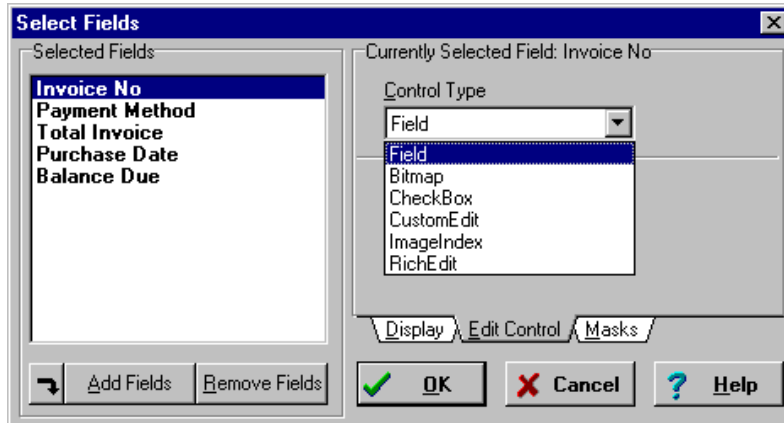


Figure 4.4 - The Currently Selected Field section of the Select Fields dialog box with the Edit Control tab activated.

When you select *CheckBox*, you are asked to supply two additional values, defaulted to Yes and No respectively: “Value when checked” and “Value when unchecked”. If you are attaching a checkbox to a Logical field type, then you should put in values of “True” and “False”. If you have enabled the grid's *EditControlOptions.ecoCheckboxSingleClick*, then only a single click is required to toggle the checkbox.

InfoPower defaults to checkmark style checkboxes in the grid. If you wish to use an Xmark style checkbox, then see the *TwwIntl* property *CheckboxInGridStyle*.

When you select *CustomEdit*, you are asked to supply the “Control name”, which must be an existing component selected from the drop-down list. Any TwwDBGrid or TwwRecordViewDialog displaying this field uses your specified Control. If your custom control does not display as plain text and you are using it in the grid, then you may want to enable the *Control Always Paints* checkbox. In this way the grid asks the control to take care of painting the grid cell instead of the grid just painting the control as plain text. Not all controls will support use of *Control Always Paints*, but InfoPower's TwwCheckBox, TwwRadioGroup, TwwDBRichEdit do support this.

When you select URL-Link, the field is displayed as a URL-Link. The format in the database is <URL Display String>#<URL Link Address>. Alternatively you can omit the <URL Display String># and the grid will display the raw address.

When you select *ImageIndex*, the field is used as an index to the *ImageList* specified by your grid's *ImageList* property. If "Shrink To Fit" is checked, then InfoPower will shrink the image to fit into the cell.

When you select *RichEdit*, you can specify the name of a *TwwDBRichEdit* control to associate with the field. Any *TwwDBGrid* or *TwwRecordViewDialog* displaying this field uses your specified *TwwDBRichEdit*. The grid uses the rich-edit by allowing the end-user to pop-up an editing window simply by pressing F2 in the field.

When you select *Bitmap*, you are asked to supply the bitmap drawing characteristics.

<i>Bitmap Scaling</i>	Determines how the bitmap is drawn in the cell. You are given the following choices for this setting.
<i>Original Size</i>	Bitmap is copied with no scaling. If necessary, the bitmap will be clipped so it fits in the cell.
<i>Stretch to Fit</i>	Bitmap is stretched to the size of the cell.
<i>Fit Width</i>	Bitmap is stretched horizontally so that it fits the width of the cell.
<i>Fit Height</i>	Bitmap is stretched vertically so that it fits the height of the cell.
<i>Raster Operation</i>	Determines the raster operation used to copy the bitmap to the cell.

Note: Instead of using a control type of *Bitmap*, you instead may wish to embed a *TDBImage* as a custom control (setting its *Control Always Paint* checkbox to true). This will provide increased flexibility in managing the bitmap.

Masks

See the following section "Using InfoPower's Picture Masks", which details the InfoPower picture mask functionality.

Links

InfoPower's linked fields perform the equivalent of Delphi's lookup fields. This property is maintained for *backward compatibility* with earlier versions of InfoPower. See the documentation on Delphi's lookup fields if you wish to display information from a related table in the grid.

Using InfoPower's Picture Masks

InfoPower gives Delphi programmers the power to define a data entry template, or mask, for the values that can be entered into a field displayed on the screen. This functionality duplicates the Picture function that's been available in Borland's Paradox relational database product for quite some time now, so most Paradox programmers will recognize this component and its functionality almost immediately.

Picture masks force end-users to enter only those characters, digits and special characters that you allow them to, and only in those specific positions within a field that you pre-define. This can be very important for fields such as multi-sectioned account and part numbers; various USA and international Zip code formats; a short list of pre-defined words such as Red, Blue or Green; or even for automatic capitalization of only the first word or all words in a field. The uses of Picture Masks are almost limitless, and go far beyond what Delphi's edit masks can do

Components that support picture masks

The majority of InfoPower components support picture masks. These include the following: TwwDBEdit, TwwDBComboBox, TwwDBComboDlg, TwwDBSpinEdit, TwwDBGrid, TwwDataInspector, TwwRecordViewDialog, TwwRecordViewPanel, TwwIncrementalSearch, and TwwFilterDialog. The TwwDBLookupCombo and TwwDBLookupComboDlg do *not* support picture masks.

How are picture masks used

InfoPower uses picture masks in the following ways.

1. For bound (*datasource* and *datafield* properties both assigned) and unbound controls, pictures are used as an edit mask. When editing with one of the picture-mask supported visual components, the user's entry is continuously checked against the mask to prevent illegal entries as well as to assist in the auto-filling of values.

Note: Picture masks only check the validity of the input when the user is typing at the end of the text. Its not possible to verify in the middle because quite often the user needs to temporarily make the text invalid, but it would become valid if they were allowed to continue. For instance consider the picture mask {Red, Blue, Green, Gn}, and the current text contains 'Gn'. The end-user wishes to insert 'ren' between 'G' and 'n', so that the end-result would be 'Green'. If picture masks were doing validation in the middle of the text it would not allow them to type in 'r', as 'Grn', would be invalid. Your data though is still protected as validation is again performed when the end-user tries to exit the control. As a result, they still cannot enter anything invalid into the database.

2. When using picture masks with an InfoPower TwwTable or TwwQuery, they are additionally used to verify the validity of the fields in the record before the record is posted. In this way even invalid assignments to a database field using code will be detected. InfoPower generates a DatabaseError exception when encountering an invalid

user value. If you wish to provide your end-users a different error message, you can use the TwwTable, TwwQuery, or TwwQBE event *OnInvalidValue*.

- When using an InfoPower TwwTable component against a Paradox table, the picture masks stored in the physical table are automatically read and used by the component.

Note: Do not confuse InfoPower’s picture masks with how a field is displayed to the end-user. InfoPower’s masks are only used for editing and validation. They do not control how a field is displayed in a grid or an edit control. To control display formatting, see the Delphi TField DisplayFormat property, as well as the formatting options available for date/time such as ShortDateFormat, etc.

Defining a PictureMask property string:

To define a *PictureMask* property string, use the following special characters for the definition of required characters, digits, repeats, literals, letter case conversion, and optional entries, along with any other character as a constant:

Character	Description
#	Any digit (0-9)
?	Any letter (a-z or A-Z)
&	Any letter (a-z or A-Z – automatically converted to uppercase)
~	Any letter (a-z or A-Z – automatically converted to lowercase)
@	Any character
!	Any character (letters automatically converted to uppercase)
;	Next character is to be used literally and not used as a picture mask character.
*	Repeat the following character any number of times. For instance *& means convert any number of characters to uppercase. To specify a specific number of times, follow the * with a number. For instance to specify 5 numbers in a row, you would use *5{#}
[abc]	Optional sequence of characters abc that do not need to be entered by the end-user
{a,b,c}	Grouping operator. Set of a, b, or c. The end-user must choose either a, b, or c. For instance {Red,Green,Blue} means the end-user must choose either Red, Green, or Blue. Similarly {R,G,B} means the end-user must choose between the characters R, G, or B. Note: The picture mask must be carefully constructed to avoid situations where one element completely contains another element in the group. For instance if you have 3 valid choices of Auto, Automobile, and Car, do NOT set the picture mask to {Auto,Automobile,Car}. Instead set the

	picture mask to {Auto[mobile],Car}.
--	-------------------------------------

If you enter any other character in a picture mask, InfoPower treats the character as a constant. When the end-user is entering a value in a field with a picture mask that contains constant values, and they come to the point where the constant value is specified, InfoPower automatically fills-in (enters or types) the constant for the end-user. This is known as Auto-Fill and requires the *AutoFill* property setting to be True, otherwise the constant characters will *not* be filled-in automatically.

Simple Examples:

- #####[-#####] Standard U.S.A. 5-digit postal code with optional 4-digit suffix (i.e. 12345 or 12345-6789). Optionally, this picture mask can be specified as *5{#}[-*4#]. This “shorthand” method allows you to specify very long picture masks in just a few characters.
- #&#&#& Standard Canadian postal code (i.e. 1A2B3C).
- *! Any number or any character with all *letters* automatically converted to uppercase (i.e. 123abc becomes 123ABC).
- {Yes,No} Either "Yes", "No", or blank (since the braces indicate that an entry in this field is *optional*). If the *AutoFill* property is set to True, all the user has to type is either “y” or “n” and the respective value is automatically entered into the field. If *AutoFill* is set to False, the user must enter the entire word, either “Yes” or “No”.
- &*? Letters, with first letter capitalized.
- #[#][#] Numeric of 3 digits or less

Complex AutoFill Example:

{Red,Gr{ay,een},B{l{ack,ue},rown},White,Yellow}

This picture mask allows the user to enter only the colors Red, Gray, Green, Black, Blue, Brown, White or Yellow. If the *AutoFill* property is set to True, as the user enters characters that define the specific color (non-repeating characters), InfoPower will automatically fill in the remaining characters. For example, when the user enters the letter “r”, as the first character in the field, InfoPower will automatically fill the field with the text “Red”. When the user enters a “b” as the first character in the field, InfoPower doesn’t know whether to enter Black, Blue or Brown into the field. So, nothing happens until the user adds the characters “la” (for Black), “lu” (for Blue), or “r” (for Brown), at which point InfoPower automatically fills in the corresponding value into the field.

How to edit the picture masks using the InfoPower design tools

You can edit picture masks via one of the following methods.

1. Invoke the *Select Picture Mask* Dialog by clicking on the *Picture* property through one of the picture-mask supported InfoPower components. See the following pages for a description of the *Select Picture Mask* dialog.
2. Invoke the *Select Fields* Dialog by double clicking on an InfoPower TwwDBGrid.

There exists at most one picture mask for a given database field, so using any one of the above 3 techniques is equivalent. All InfoPower components tied to the same field will share the same picture mask.

Note: If using Paradox tables that define a picture mask at the table level (Database Desktop), these will automatically be used by the InfoPower TwwTable when it posts a record. However in these cases you cannot change the picture mask for the field within Delphi.

Key properties and events of InfoPower's picture mask support

Unless otherwise indicated the following properties and events are for the TwwDBEdit, TwwDBComboBox, TwwDBComboDlg, TwwDBSpinEdit, TwwDataInspector, TwwRecordViewDialog, TwwRecordViewPanel, and TwwDBGrid. Note: the TwwDBGrid, TwwRecordViewDialog, and TwwRecordViewPanel do not have properties with the following names, but instead has design-time dialogs for manipulating these properties.

Picture | PictureMask

This property allows you to enter picture mask characters manually, via the Object Inspector. The default value is blank. Please refer to the Defining a Picture Mask section earlier in this chapter for a description of defining a picture mask, using the Select Picture Mask dialog box to select a previously saved picture mask, along with creating and saving a new picture mask.

Data Type: String

Valid Values: Blank or a valid picture mask

Picture | AutoFill

When you enter a picture mask that consists of auto-fill characters, setting this property to True will activate the auto-fill capabilities of the picture mask when the application is run. When False, all auto-fill capabilities of the picture mask defined for this field are deactivated. The default value is True.

Data Type: Boolean

Picture | AllowInvalidExit

When True, the end-user may move off of a field that contains a picture mask even though the data they entered might be invalid. When False, the user must enter valid data, according to the picture mask, into this field. The default value is False. Normally you will probably want to leave this property set to False. However in some situations you may want to allow user to move to another control (i.e. button) before completing the editing.

This property only applies to unbound InfoPower edit controls. This property can not be set to True for any InfoPower edit control that is bound (*datasource* and *datafield* properties assigned). Bound controls do not allow the edit control to lose focus if it has a value that does not pass the picture mask test. This ensures the integrity of the data input.

Data Type: Boolean

Picture | PictureMaskFromDataSet

This property is ignored unless you are using a control bound to a TwwTable, TwwQuery, or TwwStoredProc. In other cases the picture mask information is always stored in the edit control. If the related edit control is bound to a datasource, then the picture mask information can be stored in the dataset. Set this property to false to instead store the picture mask information in the edit control.

UsePictureMask

When True, picture masks are used by the InfoPower controls during editing. When False, picture masks are not used during editing. In either case when using a TwwTable or TwwQuery, picture masks are still used to verify the validity of the fields in the record before the record is posted.

OnCheckValue

This event allows you to perform some custom action based on any change to the edit component's text. This event is only fired if you have a picture mask assigned for the field. For instance you may want to put the edit control in yellow when it does not satisfy the picture mask requirements.

Parameters

Sender : TObject Edit Component that is being checked. Sender is the component this event is attached to, except when in a grid. If used in a TwwDBGrid, Sender is a TwwInplaceEdit component. See the property TwwDBGrid *InplaceEditor* for more information on the TwwInplaceEdit type.

PassesPictureTest : Boolean True if edited text passes the picture mask constraints.

Example 1 (Coloring of a TwwDBEdit control during editing): The following example will give a TwwDBEdit component a yellow background whenever the edited text does not satisfy the picture mask constraints.

```
procedure TMainDemo.wwDBEdit1CheckValue(Sender: TObject;
  PassesPictureTest: Boolean);
begin
  if (not PassesPictureTest) then wwDBEdit1.Color := clYellow
  else wwDBEdit1.Color := clWhite;
end;
```

Example 2 (Coloring of a TwwDBGrid cell during editing): The following example will give cells being edited with the default editor a yellow background whenever the edited text does not satisfy the picture mask constraints. If you are also attaching your own custom editors within the grid (i.e. TwwDBComboBox, TwwDBEdit) you need to use the code in the prior example.


```

procedure TForm1.wwDBGrid1CheckValue(Sender: TObject;
  PassesPictureTest: Boolean);
begin
  if (not PassesPictureTest) then
    (Sender as TwwInPlaceEdit).Color := clYellow
  else
    (Sender as TwwInPlaceEdit).Color := clRed;
end;

```

PictureMaskFromDataSet (TwwDBGrid, TwwRecordViewPanel, TwwRecordViewDialog, TwwDataInspector)

This property determines where the design-time picture mask settings are stored. If this property is True, then the picture mask information is stored in the dataset. This allows you to configure the picture masks once for a field in a centralized location. If this property is false, the picture mask definition is stored in the related edit component.

This property is ignored unless you are using a control bound to a TwwTable, TwwQuery, or TwwStoredProc. In other cases the picture mask information is always stored in the visual control

OnValidationErrorUsingMask (TwwIntl)

When exiting an InfoPower edit control with a picture mask assigned, you can customize the error handling when the input does not satisfy the picture mask constraints. Be sure to set your *TwwIntl* connected property to *True*, if you want the InfoPower controls to call this event. If the *OnInvalidValue* of the related dataset is assigned, then this event is not called as it then uses the *OnInvalidValue* event. The parameters for this event are described below.

Parameters

<i>Sender</i> : TObject	Edit control or grid associated with the picture mask validation error.
<i>Field</i> : TField	TField that has an invalid value
var <i>Msg</i>	Assign <i>Msg</i> to override the default error message.
var <i>DoDefault</i>	Set to False if you want to completely handle the display of the error message as well as throwing the exception.

Example: The following code changes the picture mask error message.

```

procedure TMainDemo.wwIntl1ValidationErrorUsingMask(Sender: TObject;
  Field: TField; var Msg: String; var DoDefault: Boolean);
begin
  Msg:= 'Bad value: ' + field.fieldname;
end;

```

OnInvalidValue (TwwTable, TwwQuery, TwwQBE, TwwStoredProc)

Before posting a record, InfoPower will test the defined picture masks for each field. If it finds a field value that does not pass the picture mask test, then InfoPower generates a DatabaseError exception. If you wish to provide your end-users a different error message, you can use the TwwTable, TwwQuery, TwwStoredProc, or TwwQBE event *OnInvalidValue*.

When using this method, be sure to use either a procedure that throws an exception (such as

DatabaseError - See Delphi on-line help for information on this procedure), or throw your own exception. If you do not, the user will still see the default message.

Parameters

DataSet: TDataSet Table or Query attempting a post

Field: TField TField that has an invalid value

Example: The following example changes the error message on an invalid value to “Value is not valid : ”, followed by the Field’s display name.

```
procedure TForm1.wvTable1InvalidValue(DataSet: TDataSet;
  Field: TField);
begin
  DatabaseError('Value is not valid : ' +
    Field.DisplayLabel);
end;
```


Using the Select Picture Mask dialog

The *Select Picture Mask* Dialog gives you a convenient way of assigning a picture mask to an InfoPower edit component. This dialog is invoked by clicking on the *Picture* property through one of the picture-mask supported InfoPower components (TwwDBEdit, TwwDBComboBox, TwwDBComboDlg, and TwwDBSpinEdit, TwwIncrementalSearch)



Figure 4.5 - Select Picture Mask Dialog

The following defines the controls in the dialog:

Picture Mask - Enter your picture mask into the *Picture Mask* edit control. Note: If you want to select an existing picture mask, click on the  icon.

Auto Fill - Enable this checkbox to enable AutoFill. See property *AutoFill*

Allow Invalid Exit - Enable this checkbox to enable AllowInvalidExit. See property *AllowInvalidExit*. This checkbox is disabled if you are modifying a bound edit control.

Use Mask in Edit Field in Control - Enable this checkbox to use the mask when the user edits the field in the grid. Otherwise the mask is only used prior to posting of the record.

This property is identical to the *UsePictureMask* property of the InfoPower edit control, and is only placed in this dialog for convenience.

Design Mask - Click this button to create and test a new picture mask

Using the Design Picture Mask dialog

The *Design Picture Mask* dialog allows you to design a new picture mask that you can save to the picture mask database. This dialog can be accessed by either clicking on the *Design Mask* button in either the *Select Picture Mask* dialog, or from the *Select Fields Dialog | Masks* Tab Page.

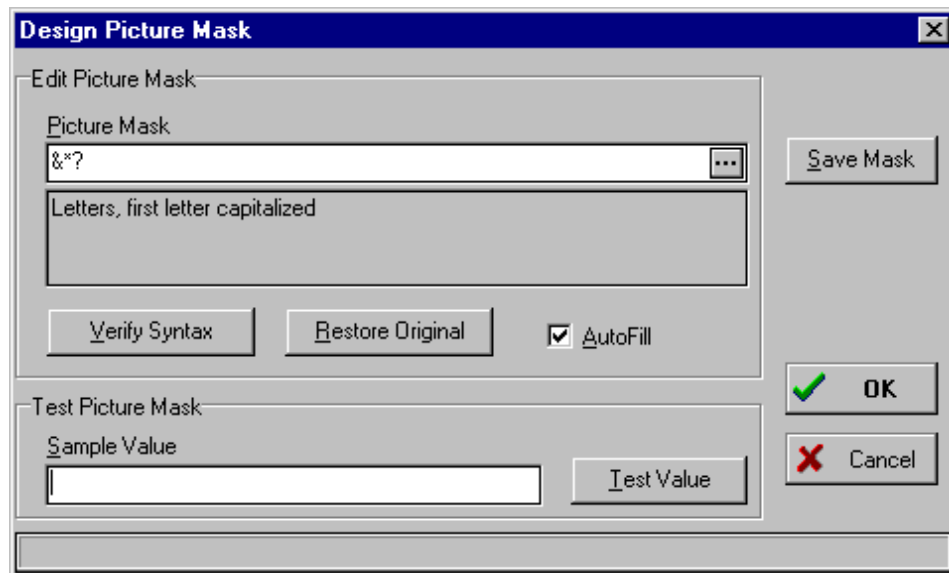


Figure 4.6 - Design Picture Mask Dialog

The following defines the controls in this dialog:

Picture Mask - Picture mask to be tested

Test Value - This button tests the currently entered Sample Value and checks if it is valid with the picture mask. Since the Sample Value is continuously checked as you type into it, this button may seem useless. However it is useful if you change the picture mask and want to test the current Sample Value with the new Picture Mask.

Save Mask - Click this button to save your new picture mask to the InfoPower mask database.

Verify Syntax - Click this button to ensure that the syntax of your picture mask is correct.


Restore Original - Click this button if you want to restore the picture mask to the value before you made any changes.

AutoFill - Enable this checkbox if you want to automatically fill during editing

Sample Value - Enter text into here to see your new picture mask in action.


Create and save a new picture mask:

To design a new picture mask you can follow these steps.

1. Enter your picture mask into the *Picture Mask* edit control. Note: If you want to start from an existing picture mask, click on the  icon.
2. Click on the *Verify Syntax* button to check that your picture mask syntax is correct.
3. Select your desired setting for the default *AutoFill* capabilities.
4. Now test your new picture mask by editing into the *Sample Value* edit control.
5. When satisfied with your mask, you can click on *Save Mask* button to save your mask to the picture mask database.
6. Click on the *OK* button to use this picture mask in your InfoPower components. Your picture mask is saved to a file named InfoPowerMasks.ini. You can configure this location by editing the registry variable \Software\Woll2Woll\InfoPower\Masks IniFile.

InfoPower change : InfoPower's picture mask design tools now use an INI file to store your custom picture masks. In the past a Paradox table was used to store these settings in your IDE environment. However to remove the necessity of installing the BDE in your design time environment, InfoPower no longer uses this Paradox table.

Remove an existing picture mask from the picture mask table:

From the Design Mask Dialog, click on the  icon to enter the *Lookup Picture Mask* dialog. Then select the mask you wish to delete, and then click on the *Delete Mask* button.

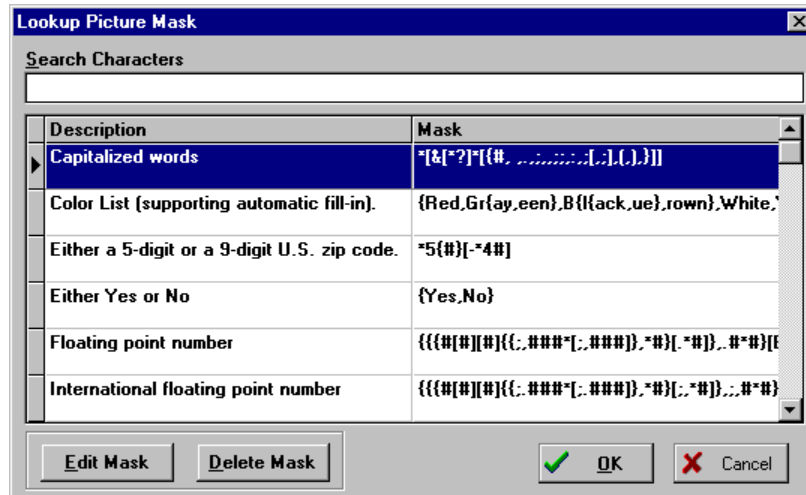


Figure 4.7 - Lookup Picture Mask Dialog

Tips

If your picture mask contains auto-fill capabilities, remember to set the *AutoFill* property to True.

Determining the object names of the controls contained in an InfoPower dialog

Many of the InfoPower components have an associated form or dialog. Within these forms are many components which you may want to change the properties of. In order to accomplish this, you can use the *OnInitDialog* event of the component and reference the objects contained within the form. To find out the object names, you can open the associated .pas file that contains the dialog. The names of these .pas files are documented in the *OnInitDialog* event, or at the beginning of the event documentation for the component.

If you do not have the source code version of InfoPower, then perform the following steps to generate a temporary form that contains all the components in the dialog.

1. Open the associated .dfm file instead of the .pas file.
2. Perform *Edit | Select All* from the Delphi IDE menu to select all the contained components.
3. Perform *Edit | Copy* from the Delphi IDE menu to copy the components to the clipboard.
4. Create a new form by select *File | New Form* from the Delphi IDE Menu.
5. Copy all the components to your newly created form with the menu selection *Edit | Paste*.

After performing the above steps you can view your newly created form to determine the object names for every component in the InfoPower dialog.

Using XP Themes with InfoPower

InfoPower 4000 supports windows themes throughout its controls. To enable themes, you must be using Delphi 7 or later and drop in a TXPManifest component into your application. TXPManifest can be found in the Win32 component palette.

If you wish to disable themes for certain InfoPower components, you can set the *DisableThemes* property to false for the intended component.

InfoPower Component Reference

Description of Reference

This chapter of the InfoPower *Developer's Guide* discusses the details of each InfoPower component, the new properties and events added to it, properties and events that have been modified or removed, along with how-to and tips sections for each component. It does **not** discuss the properties or events that are available as part of the ancestor Delphi /C++ components, unless changes were made to them. If you are not familiar with Delphi's built-in data-aware components, their properties or events, please read through the *Delphi User's Guide* and *Database Application Developer's Guide* **before** you begin working with the InfoPower component library.

The InfoPower library contains native VCL components, many of which are based on Delphi's built-in data-aware components. Thus, when you are familiar with Delphi's built-in components, properties and events, you already know how to use most of the InfoPower components—with no further training.

Several of the InfoPower components appear to use the same visual bitmap as their Delphi counterparts, such as `TwwDataSource`, `TwwTable` and others. Upon closer inspection, you'll notice a + symbol in the upper right-hand corner of the InfoPower bitmaps. This is how you can tell whether you are working with an InfoPower or Delphi/C++ Builder component.

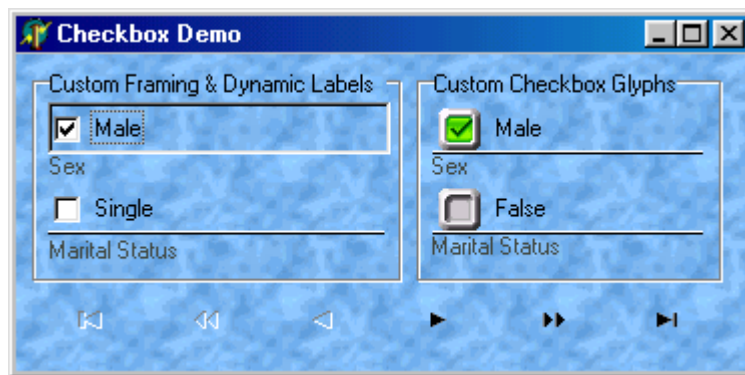
This chapter is dedicated to describing each InfoPower component, along with their ancestry; required supporting components; added properties, events and methods; required property assignments; modified properties, events and methods; how-to examples; and tips.

TwvCheckBox



InfoPower integrates a versatile new checkbox control into its suite. Some of its features include the following:

- ◆ Support for your own custom bitmaps for the checkbox glyphs
- ◆ Support for 3 states including integration with the InfoPower grid, record-view components, and data inspector.
- ◆ Dynamic caption support – The text of the checkbox can change to reflect the underlying value or mapped value.
- ◆ Custom framing and transparency support for a consistent look with other InfoPower edit controls



Ancestor

```
TCustomCheckBox
├─ TwvCustomCheckBox
│   └─ TwvDBCCustomCheckBox
```

Added Properties

Alignment

Assign this property to change the location of the text within the control. If Alignment is set to `taRightJustify`, then the text is aligned on the right-hand side of the control. If Alignment is set to `taLeftJustify`, then the text is aligned on the left-hand side of the control.

AllowGrayed

If AllowGrayed is set to `True`, the check box has three possible states: checked, unchecked, and grayed. If AllowGrayed is set to `False`, the check box has only two possible states: checked and unchecked.

AlwaysTransparent

Set this to true if you wish for the checkbox to be transparent even when it has the focus. Normally when `Frame.Enabled` and `Frame.Transparent` are both true, the control is only transparent when it does not have the focus. Note: This property has no effect unless `Frame.Enabled` and `Frame.Transparent` are both true.

Caption

Assign a string to this property to assign the label that appears next to the checkbox. Note: Assigning this property has no effect if `DynamicCaption` is set to true.

Checked

Use `Checked` to determine whether the check box is in the checked state.

Note: If the `AllowGrayed` property is `True`, you may find it more useful to use the `State` property.

DisableThemes

If your project has enabled XP themes but you do not wish for this control to be theme-enabled, then set this property to `False`.

DisplayValueChecked

This is the value that the checkbox displays when the checkbox is checked. Note: `DynamicCaption` must be true in order for the checkbox to display this value. If this value is unassigned, then it displays the string assigned to `ValueChecked`.

DisplayValueUnchecked

This is the value that the checkbox displays when the checkbox is unchecked. Note: `DynamicCaption` must be true in order for the checkbox to display this value. If this value is unassigned, then it displays the string assigned to `ValueUnchecked`.

DynamicCaption

Set this to true if you wish for the checkbox to display the field value as its checkbox label, instead of the static label indicated by the `Caption` property.

Frame

See the topic “Key properties and events for custom framing” in chapter 4 for more information on this property.

Data Type: `Tw>EditFrame`

Images

Assign this property if you wish to change the icons displayed by the checkbox. The first image in the `imagelist` is used as the unchecked icon, and the second image is used as the checked icon, and the third image is used as the grayed icon.

Data Type: `TImageList`

Indents

Use Indents to change the relative placement of the icons and the text.

- ButtonX** Assign this property to specify the number of pixels to move the checkbox icon to the left (negative value) or right (positive value).
- ButtonY** Assign this property to specify the number of pixels to move the checkbox icon upward (negative value) or downward (positive value).
- TextX** Assign this property to specify the number of pixels to move the text to the left (negative value) or right (positive value).
- TextY** Assign this property to specify the number of pixels to move the text upward (negative value) or downward (positive value).

When the database field that the checkbox is bound to is null, blank, or doesn't match the ValueChecked or ValueUnchecked properties, then the checkbox displays its state according to the NullAndBlankState property. See the Delphi documentation under TCheckboxState for a description of its possible values.

NullAndBlankState

When the database field that the checkbox is bound to is null, blank, or doesn't match the ValueChecked or ValueUnchecked properties, then the checkbox displays its state according to the NullAndBlankState property. See the Delphi documentation under TCheckboxState for a description of its possible values.

Data Type: TCheckBoxState

ReadOnly

Set this property to true to prevent the user from toggling the checkbox.

ShowFocusRect

When true, a focus rectangle is drawn around the text. You may wish to set this property to false when using custom framing, as this can already give the end-user a visual cue to when the checkbox has the focus.

ShowText

Set this property to false to hide the text of the checkbox. This may be useful to you if your checkbox is embedded in the grid.

State

Indicates whether the check box is selected, deselected, or grayed.

Valid Values:

- cbUnchecked* The check box has no check mark.
- cbChecked* The check box has a check mark in it.
- cbGrayed* The check box has a check mark in it, but it is grayed.

ValueChecked

This is the value that the checkbox stores into the database when the checkbox is checked.

ValueUnchecked

This is the value that the checkbox stores into the database when the checkbox is unchecked.

Added Events

OnMouseEnter

Occurs when the mouse cursor passes from outside the control to inside the control.

OnMouseLeave

Occurs when the mouse cursor passes from inside the control to outside the control.

How-to

Dynamic Captions

Often the value stored in the database for a given field is a Boolean value that could be represented as a '1' or a '0', 'True' or 'False', 'Yes' or 'No', etc. In these situations it is often desirable for the checkbox caption associated with these values to be more descriptive, or to reflect the actual value that is stored in the database.

For example: If you wished to display Male or Female instead of the value stored in the database, then all you need is to do the following:

- 1) Set *DynamicCaption* to True.
- 2) Set the values of *DisplayValueChecked* and *DisplayValueUnchecked* to 'Male' and 'Female' respectively (noting to match them to the values that are stored in the database).

Note: *ValueChecked* and *ValueUnchecked* should match the stored values in the database.

Hot-Track the CheckBox's Caption

Use the *OnMouseEnter* and *OnMouseLeave* events to create a hot-tracking effect on the label of the caption. To do this put the following code in the *OnMouseEnter* and *OnMouseLeave* events of your *TwwCheckBox*.

```
procedure TForm1.wwCheckBox1MouseEnter(Sender: TObject);
begin
  (Sender as TwwCheckBox).Font.Color := clBlue;
  (Sender as TwwCheckBox).Font.Style := [fsUnderline];
end;

procedure TForm1.wwCheckBox1MouseLeave(Sender: TObject);
begin
  (Sender as TwwCheckBox).Font.Color := clWindowText;
end;
```

```
(Sender as TwwCheckBox).Font.Style := [];  
end;
```

TwwClientDataset



The non-visual TwwClientDataset component allows you to define a database-independent, distributed dataset that supplies data to one or more of the other InfoPower visual interface components placed on your form.

Note: This component is maintained for backwards compatibility. InfoPower allows you to directly use TClientDataSet without having to use the TwwClientDataSet. You may still wish to use the TwwClientDataSet for backwards compatibility, or if you want to store the design-time picture mask definitions into the dataset instead of the visual control.

Ancestor

TClientDataset.

Required supporting components

None.

Added Properties

ControlType

This property holds information about the type of control used to display a field if the field is contained within a grid component. The default value is Field. (See *Using the Select Fields Dialog Box* at the beginning of Chapter 4.) To change this property at runtime, see the *SetControlType* method of the TwwDBGrid component.

Data Type: (Internal to InfoPower)

PictureMasks

This property holds information about a field's picture mask. See *Using InfoPower's Picture Masks* in Chapter 4 for more details.

Data Type: TStrings

Valid Values: (Internal to InfoPower)

ValidateWithMask

See the documentation for *ValidateWithMask* under the TwwTable component

Data Type: Boolean

Modified properties

None.

Added Events

OnFilter

This function is equivalent to *OnFilterRecord*. For consistency with other InfoPower datasets we have included this event. See the documentation for *OnFilter* under the *TwWTable* component.

OnInvalidValue

See *Using InfoPower's Picture Masks* in chapter 4.

Added Methods

wwFilterField

See the documentation for *wwFilterField* under the *TwWTable* component

How-to

The *TwWClientDataset* component is inherited from Delphi's *TClientDataset*, so please refer to your Delphi manual for more information about this component. Since InfoPower's *TwWClientDataset* component is inherited from Delphi's *TClientDataset* component, you are provided with 100% backward compatibility. Thus, you can safely replace your use of *TClientDataset* with *TwWClientDataset* at any time.

TwwController



The non-visual TwwController component allows you to centralize your framing properties for an application into a single component. Assign the controller property of each InfoPower or 1stClass component that you want to use the controller. If you modify the framing in the controller during program execution, you can call the *ApplyFrame* method to have the controller apply the new framing properties to each edit control attached to the controller.

Ancestor

TComponent

Required supporting components

None.

Added Properties

Frame

This property contains the framing properties for the controller. Each control that has its controller property set to this component will use these framing properties instead of the properties of its own Frame. See Chapter 4 for more information on customizing the button effects.

Data Type: TwwEditFrame

ButtonEffects

This property contains the button effect properties for the controller. Each control that has a button that can be clicked on has specific display attributes that can be customized. See Chapter 4 for more information on customizing the button effects.

Data Type: TwwButtonEffects

Added Methods

ApplyFrame

If you modify the framing in the controller during program execution, you can call the ApplyFrame method to have the controller apply the new framing properties to each edit control attached to the controller.

TwWDataInspector

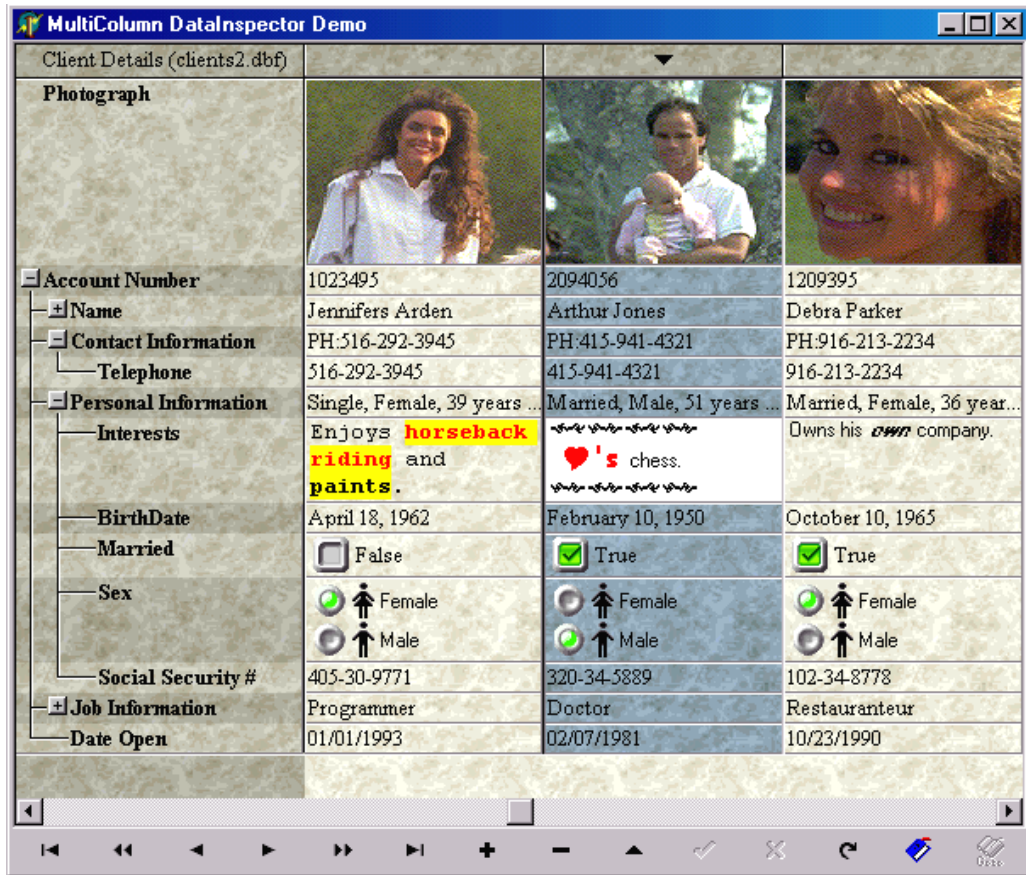


InfoPower provides a robust component that allows you to hierarchically edit one or more records, consisting of one or more datasets. This component allows you to display records, similar to the way you edit/view an object in the Delphi object inspector. It has substantial advantages over more traditional ways of editing, including the ability to group related fields together (even from different datasets), provide a hierarchical view, conserve screen real-estate, embed custom controls, display a background image or tile, and much more. See the demo in the \ip4000\demos\inspector directory

This component can also be used without a datasource, which significantly increases the way this component can be used. InfoPower 4000 also adds an *Enabled* property for each inspector item.

InfoPower's versatile data inspector includes the following capabilities:

- ◆ **Support for multi-record display:** InfoPower's versatile data inspector can now display *multiple* records vertically, bringing your end-users a smart alternative to the left-right editing of a traditional grid. To support multiple records, you will need to set the *DataColumns* property. You may also want to set the *IndicatorRow.Enabled* property.
- ◆ **Improved custom control integration and flexibility :** InfoPower allows you to embed a wider variety of controls in the inspector. You can now even embed non-InfoPower controls, such as the TDBImage. The inspector also allows each custom control to do their own painting in the grid so that even graphics, richedits, etc. will be displayed for every row in the grid without any code on your part.
- ◆ **Background texture tiling:** InfoPower allows your applications to further impress by adding support for background texture tiling. The component takes care of blending your tile with the color of the inspector region, giving you a truly impressive and professional display. To enable background texture tiling see the *PaintOptions* property.
- ◆ **Alternating colors for rows:** New property to automatically alternate the row colors and to highlight the active column See the *PaintOptions.AlternatingRowColor* and *AlternatingRowRegions* properties.
- ◆ **Enhanced hierarchical display:** When items are expanded., the inspector now paints tree lines supporting an elegant tree display. To enable the tree-lines set the *Options | ovShowTreeLines* property to True.



Ancestor

TCustomGrid

Required property assignments

None

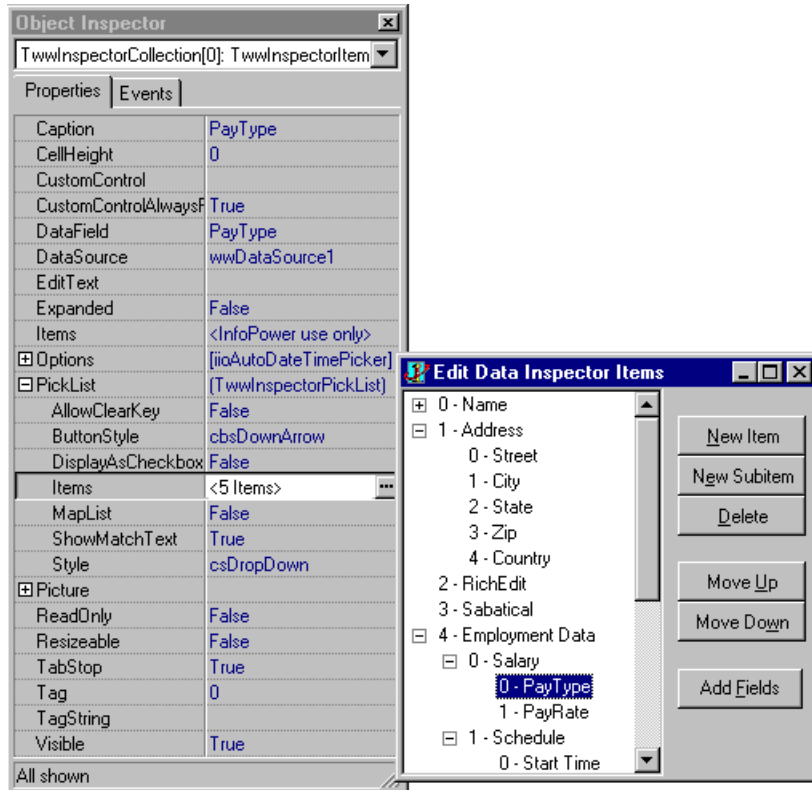
Overview of TwwDataInspector architecture

The Data Inspector's display consists of one caption column and one or more data columns. The first column defines the labels, and the remaining column display the data. The Data Inspector uses a TCollection to define the attributes for each row of the data inspector, as a TCollectionItem defines each row in the data inspector

The data type of the TCollection is TwwInspectorCollection, and the collection items are of type TwwInspectorItem.

Defining the data inspector's items

To customize each row of the data inspector, dbl-click the control at design time. The following designer form is then displayed.



The Data Inspector component contains a hierarchical collection of one or more inspector items. From here you can click on an item, and then by using the Delphi object inspector you can customize any of the collection item properties of the selected `TwwInspectorItem`.

Customizing the custom control: If you have previously assigned the `CustomControl` property, you can select this control by holding down the ALT key before you click on the inspector item. After doing so, the object inspector will show you the properties and events of the custom control.

New Item: Click this button to add a new item to the data inspector. The new item is inserted as the last child of the currently selected item's parent. If the selected item is a root node, then an item is added to the end of the list.

New Subitem: Click this button to add a new child item to the currently selected item in the data inspector. The new item is inserted as the last child of the currently selected item.

Delete Item: Deletes the currently selected item and its children from the data inspector.

Move Up: Moves the currently selected item up one. You can also drag an item to another location by clicking the item with the mouse and then dragging the mouse to the location you wish to move to. If you hold the Shift key when you release the mouse, the item becomes a child of the destination node. Otherwise the item becomes the prior sibling of the destination node.

Move Down: Moves the currently selected item down one. You can also use drag and drop as described above.

Add Fields: Click this button to create items from the list of fields associated with the inspector's datasource. If you need to add fields from other datasources, then click the New Item button, followed by setting its datasource, datafield, and caption properties.

Added properties

ActiveEdit (Runtime only)

ActiveEdit returns the current editor active in the data inspector. This value will vary depending upon which row is active. If you have assigned a custom control to the active row, then it returns the handle to this custom control. If you are using a picklist, then it returns the *TwwDBComboBox* associated with displaying this picklist.

Data Type: *TWinControl*

ActiveItem (Runtime only)

This property returns the item associated with the currently active row in the inspector. You can also set this property to change the active row. See the methods *GetItemByCaption*, *GetItemByField*, *GetItemByTagString*, and *GetItemByRow* to help retrieve a handle to a *TwwInspectorItem*.

Data Type: *TwwInspectorItem*

ActiveRows (Runtime only)


This property returns the number of rows that are currently displayed by the inspector.

Data Type: Integer


ButtonOptions

This property defines the bitmaps used to display the expand and collapse buttons of the data inspector.

CollapseGlyph

Assign a *TBitmap* to this property if you wish to override the appearance of the collapse button. If this property is unassigned, then the button is displayed as .

ExpandGlyph

Assign a *TBitmap* to this property if you wish to override the appearance of the expand button. If this property is unassigned, then the button is displayed as .

CaptionColor

The *CaptionColor* property defines the background of cells that display the label for a data inspector item.

CaptionFont

Font used to display text in the caption column

Data Type: TFont

CaptionIndent

Number of pixels to indent when painting the text in the column associated with the captions. Increase this value to move the caption text more to the right of its default placement.

Data Type: Integer

CaptionWidth

Set this property to change the width of the caption column within the data inspector. You can also size the column by using the mouse to drag the line separating the columns.

Data Type: Integer

Canvas (Runtime only)

TCanvas used to paint the data inspector. You may wish to refer to this property when using the *OnDrawDataCell* or *OnDrawCaptionCell* event.

Data Type: TCanvas

DataColumns

Set *DataColumns* to a value greater than 1 to display multiple data columns of records in the inspector when the *DataSource* property is assigned.

Data Type: Integer

DataSource

Use *DataSource* to specify the data source component through which the data from a dataset component is provided to the *TwwDataInspector*

Data Type: TDataSource

DefaultRowHeight

Set *DefaultRowHeight* to change the default height of the rows in the data inspector. This property defaults to 0, which tells the control to compute the row height based on the height of the text given the control's font. If you assign an individual row's height through its *TwwInspectorItem.CellHeight* property, then the *DefaultRowHeight* property is ignored for that row.

Data Type: Integer

DisableThemes

If your project has enabled XP themes but you do not wish for this control to be theme-enabled, then set this property to *False*.

DottedLineColor

The *DottedLineColor* property defines the color of dotted lines within the data inspector. Dotted lines appear around each cell's borders when the inspector's *LineStyleCaption* or *LineStyleData* properties are set to *ovDottedLine*.

Data Type: TColor

IndicatorRow

IndicatorRow defines the attributes for the indicator row at the top of the inspector. This property is new in InfoPower, and is designed to indicate to the end-user which column is the active record.

Data Type: TwwInspectorIndicatorRow

Caption	Assign this property to display text in caption column of the indicator row. Data Type: String
Color	Sets the background color of the indicator row. See the <i>PaintOptions</i> property if you wish to blend this color with a tile. Data Type: TColor
Enabled	When true, the inspector display a row at the top to indicate which column is the active record. Data Type: Boolean
Height	Assign this property to change the height of the indicator row. This property defaults to 0, which means that the indicator row uses the default row height. Data Type: Integer
TextAlignment	Assign this property if you have assigned the caption property and want to align its text Data Type: TAlignment

Items

This property contains a collection of items assigned to the data inspector. Each collection item is of type *TwwInspectorItem*. Clicking on this property from the object inspector brings up InfoPower's data inspector collection editor. See the "Defining the Data Inspector's Items" topic discussed earlier in this section. See the documentation under *TwwInspectorCollection* and *TwwInspectorItem* for detailed documentation on these data types.

Data Type: TwwInspectorCollection

InplaceEditor (Runtime Only)

The default inplace editor used when editing cells. The default inplace editor not used if you have assigned a custom control, picklist, or checkbox to the field. Additionally is not used when a datetimepicker control is displayed in the cell. See the *ActiveEdit* property for a generic way of retrieving the editor for the active row.

Data Type: TwwDataInspectorEdit

LineStyleCaption

The style of the borders for each cell in the caption column can be set to any one of the following:

Data Type: TwwDataInspectorLineStyle

Valid Values: {ovNoLines,ovDottedLine,ovLight3DLine,ovDark3Dline,ovButtonLine}

<i>ovNoLines</i>	No lines are displayed around the cells
<i>ovDottedLine</i>	Dotted lines are drawn around the cells
<i>ovLight3Dline</i>	A light line is drawn around each cell.
<i>ovDark3Dline</i>	A dark line is drawn around each cell
<i>ovButtonLine</i>	A line is drawn around the cell so that the cell appears like a button.

LineStyleData

The style of the borders for each cell in the data column can be set to any one of the values, ovNonLines, ovDottedLine, ovLight3Dline, ovDark3Dline, ovButtonLine. See the LineStyleCaption property for a further description of these line styles.

Data Type: TwwDataInspectorLineStyle

Options

Options allow you to customize the appearance and certain behavior of the data inspector.

Data Type: TwwDataInspectorOptions

Valid Values: {ovColumnResize, ovRowResize, ovTabExits, ovEnterToTab, ovHighlightActiveRow, ovHideVertDataLines, ovCenterCaptionVert, ovTabToVisibleOnly, ovShowTreeLines, ovShowCaptionHints, ovShowCellHints, ovFillNonCellArea, ovActiveRecord3DLines, ovAllowInsert, ovHideCaptionColumn, ovHideVertFixedLines}

<i>ovColumnResize</i>	When true, the inspector allows the end-user to resize the columns.
<i>ovRowResize</i>	When true, the inspector allows the end-user to resize the rows whose resizable property is true. See also the TwwInspectorItem.Resizeable property.
<i>ovTabExits</i>	When false, the inspector allows the user to cycle through the rows by using tab and shift-tab. Focus is automatically moved to the next inspector item whose TabStop property is true. See also the ovTabToVisibleOnly property.
<i>ovEnterToTab</i>	When true, the enter key is converted to a tab.

<i>ovHighlightActiveRow</i>	When true, the caption column for the active row is painted with a recessed border to help signify which row has the focus. Defaults to True. You can also customize the way an active row is painted by using the <i>OnDrawCaption</i> event.
<i>ovHideVertDataLines</i>	When true, the vertical lines for a multiple column datainspector control will not be drawn. Default is False.
<i>ovCenterCaptionVert</i>	When True, the text in the caption column is centered vertically. When False, the text is positioned at the top of the cell.
<i>ovTabToVisibleOnly</i>	When True, the inspector will only tab to rows that are either a root item, or items that are part of a currently expanded branch. When <i>ovTabToVisibleOnly</i> is False, tabbing will advance to the next item whose <i>tabstop</i> property is true, and if necessary expand any items so that the item is visible. This property is ignored if <i>ovTabExits</i> is True.
<i>ovShowTreeLines</i>	When True, the inspector will display dotted lines between the inspector items. This provides a more elegant and tree-like display. See also the <i>TreeLineColor</i> property to customize the color of the tree lines.
<i>ovShowCaptionHints</i>	When True, the inspector will display the full caption as a tooltip when the mouse is over the caption. This allows the end-user to see the entire caption without having to enlarge the caption column.
<i>ovShowCellHints</i>	When True, the inspector will display the full text of the data cell as a tooltip when the mouse is over the caption. This allows the end-user to see the entire text of the cell without having to enlarge the column. Note: If you have assigned a customcontrol to the cell, then the tooltip is only displayed if <i>CustomControlAlwaysPaints</i> is set to false.
<i>ovFillNonCellArea</i>	When True, the inspector will fill the bottom non-cell area with the fixed color or <i>Column1</i> blended bitmap defined by the <i>PaintOptions</i> property.
<i>ovActiveRecord3DLines</i>	When this property is true, the active record is painted with 3D lines to help clarify its position with respect to the other records. You may also want to use this property in conjunction with the <i>PaintOptions ActiveRecordColor</i> property.
<i>ovAllowInsert</i>	When this property is true and there are no records in the <i>TwwDataInspector</i> this property will allow the enduser to edit/insert a new record. Currently this property is only

applicable when you have an empty inspector. In the future, this property may accommodate other cases.

ovHideCaptionColumn When this property is true, the Caption Column will be set to a width of zero effectively hiding it. When False, the first column which is the caption column will be visible.

ovHideVertFixedLines When this property is true, the vertical lines of the grid in the indicator row will be hidden.

PaintOptions

Use this property to really polish the look of your grid or inspector component. Note: If you are referencing this property with code, then add the unit `wwPaintOptions` to your form's uses clause.

Data Type: `TwwPaintOptions`

ActiveRecordColor Set this property to define the color that the inspector or grid use to paint the background of the data cells for the active record. This defaults to `clNone`, which means that the color of the control is used to paint the background.

AlternatingRowColor This defines the color that the inspector or grid use to paint the background for every other row. This property defaults to `clNone`, which means that the row colors are *not* alternated. See also the property `AlternatingRowRegions` to configure which section will paint with the alternating color.

AlternatingRowRegions Set this property to enable/disable the alternating color support within the grid or inspector.

arrFixedColumns Enable alternating colors in the fixed columns

arrDataColumns Enable alternating colors in the data cells

arrActiveDataColumn Enable alternating colors for the active record column. This property only applies when using a data inspector. It is ignored when using a grid.

BackgroundBitmap Assign this property to enable a background tile for the inspector or grid. You should make your tiles small so that your executables do not become large. We do not recommend non-tiled backgrounds as this may slow the performance of your grid's painting.

Note: When using non-tiled backgrounds, you may want to set `FastRecordScrolling` to false. For more information see the property `FastRecordScrolling`.

Data Type: `TPicture`

BackgroundDrawStyle	Set this property to change the way the background bitmap is drawn.
<i>bdsTile</i>	Paint the background bitmap as a tile
<i>bdsStretch</i>	Stretches the background bitmap into the control's client area.
<i>bdsTopLeft</i>	Paints the background bitmap starting at the top left of the control.
<i>bdsCenter</i>	Paints the background bitmap centered within the grid or inspector.
BackgroundOptions	Use this property to control how and where the background bitmap is painted. You may wish to enable the blending flags as they allow your grid or inspector to provide stunning and professional visual effects.
	<i>Note:</i> the background blending is automatically disabled when running on systems with less than 256 colors.
<i>coFillDataCells</i>	When true, the data area is filled with the background bitmap. Set this to false to prevent the background from being used in the data area.
<i>coBlendFixedRow</i>	When true, the fixed row is painted with a blended bitmap. The background bitmap is blended with the TitleColor (TwwDBGrid), or IndicatorRow.Color (TwwDataInspector).
<i>coBlendFixedColumn</i>	When true, the fixed column is painted with a blended bitmap. The background bitmap is blended with the TitleColor (TwwDBGrid), or CaptionColor (TwwDataInspector).
<i>coBlendActiveRecord</i>	When true, the active record is painted with a blended bitmap. The background bitmap is blended with the color defined by PaintOptions.ActiveRecordColor. This property is not currently supported for the TwwDBGrid.
<i>coBlendAlternatingRow</i>	When true, the alternating row color (PaintOptions.AlternatingRowColor) is blended with the background bitmap before it is painted into the grid or inspector.
FastRecordScrolling	Set this to true to force the grid or inspector to repaint its whole contents after any scroll operations take place. This will reduce the performance of your control's painting during scrolling operations, but will ensure that your grid or inspector's background do not shift position after the scrolling takes place.

For many tiled backgrounds, *FastRecordScrolling* can be left as *False* as the effect of tile being shifted does not harm the visual effect of the tile. If you are not using a tile, but instead have set *BackgroundDrawStyle* to something besides *bdsTile*, then you will likely want to set *FastRecordScrolling* to *False*.

PictureMaskFromDataSet

Set this to *false* if you wish for the design time settings of your picture masks to be retrieved from the related dataset component. This property is only relevant if you are using *TwWTable*, *TwWQuery*, *TwWQBE*, *TwWStoredProc*, or *TwWClientDataSet* and is ignored otherwise. It may be convenient for you to retrieve the picture masks in the dataset if you wish to use the dataset's *ValidateWithMask* property, or wish to use masks you have previously defined in the dataset.

Data Type: Boolean

Note: When this property is *false*, your picture masks assigned through the data inspector's designer are not used. Instead the dataset's picturemasks are used.

SetFocusTabStyle

This property controls which row becomes the active row when the user tabs to the control. If *SetFocusTabStyle* is set to *itsPreserveActiveItem*, then the inspector's active row is restored to its last value when it lost focus. If *SetFocusTabStyle* is set to *itsResetActiveItem*, then the active row is reset to the first row when the inspector receives focus.

Data Type: *TwWInspectorTabSetFocusStyle*

TreeLineColor

When you enable *Options | ovTreeLines*, the inspector will paint tree lines to more clearly reveal the tree-like structure of the inspector. You can set this property to change the color of the tree lines.

Added Events

OnAfterSelectCell

The *OnAfterSelectCell* event occurs after focus moves to a new row.

<i>Sender</i> :	<i>TwWDataInspector</i>	The <i>TwWDataInspector</i> associated with this event
<i>ObjItem</i> :	<i>TwWInspectorItem</i>	<i>TwWInspectorItem</i> associated with row that received focus.

OnBeforePaint

This event remains for backward compatibility, since now you can load a bitmap or tile with the *PaintOptions* property at design-time.

Write an *OnBeforePaint* event handler to paint a background image to the inspector. The parameters for this event are as follows:

<i>Sender</i> :	<i>TwWDataInspector</i>	The <i>TwWDataInspector</i> associated with this event
-----------------	-------------------------	--

Example 1: The following example paints a background image to the inspector. The bitmap originates from the file ‘Yourbitmap.bmp’.

```
var i, j: integer;
    ABitmap: TBitmap;
begin
    ABitmap := TBitmap.Create;
    ABitmap.LoadFromFile('Yourbitmap.bmp');

    if ABitmap.Width = 0 then exit;
    for i := 0 to Sender.Width div ABitmap.Width do
        for j := 0 to Sender.Height div ABitmap.Height do
            Sender.Canvas.Draw(i*ABitmap.Width,
                               j*ABitmap.Height, ABitmap);
        ABitmap.Free;
    end;
```

Example 2: Using the 1stClass TfcImager as the background

Example: If you also own Woll2Woll’s 1stClass product, you may want to use the 1stClass imager for enhanced background effects. Just drop a TfcImager into your form, load its *Picture* property, and set the imager’s properties to reflect how the image should be painted. Then to have the image painted into the inspector’s client area, use the following code in the OnBeforePaint event.

```
with fcimager1 do begin
    if WorkBitmap.Empty then UpdateWorkBitmap;
    WorkBitmap.TileDraw(Sender.Canvas, Sender.ClientRect);
end;
```

OnBeforeSelectCell

The OnBeforeSelectCell event occurs immediately before focus moves to a new row.

<i>Sender</i> : TwwDataInspector	The <i>TwwDataInspector</i> associated with this event
<i>ObjItem</i> : TwwInspectorItem	<i>TwwInspectorItem</i> associated with the row that received focus.
<i>var CanSelect</i> : Boolean	Set to False to prevent the row from receiving focus

OnCalcDataPaintText

Write an *OnCalcDataPaintText* handler to change the text displayed in the data column for a row. This event is useful when you wish to calculate the displayed text based on other criteria.

<i>Sender</i> : TwwDataInspector	The <i>TwwDataInspector</i> associated with this event
<i>ObjItem</i> : TwwInspectorItem	<i>TwwInspectorItem</i> associated with the data cell to be painted.
<i>var PaintText</i> : String	Assign this property to change the text that is displayed for the cell.

Example: The following computes the text of an item based on the captions of the enabled child items (as defined by their checkbox). This example assumes that each child uses a checkbox with True and False values.

```

{ Paint parent item based on the captions of the enabled child items }
if (Item.Caption = 'Non-focus Borders') or
  (Item.Caption = 'Focus Borders') then
begin
  CurItem:= Item.GetFirstChild;
  PaintText:= '';
  while CurItem<>nil do begin
    if curItem.checked then
    begin
      if PaintText<>' ' then PaintText:= PaintText + ',';
      PaintText:= PaintText + curItem.Caption;
    end;
    CurItem:= CurItem.GetNextSibling;
  end;
  PaintText:= '[' + PaintText + ']';
end;

```

OnCanCollapse

Write an *OnCanCollapse* handler to prevent a node from being collapsed by the user. This event is fired immediately after the end-user has tried to collapse an item, but before a node is actually collapsed. The end-user can collapse an item by clicking on its collapse button, or using a keyboard shortcut to collapse the node. The keyboard shortcuts include Ctrl-LeftArrow or LeftArrow (ReadOnly items). Set *CanCollapse* to false to prevent the item from being collapsed.

<i>Sender:</i> TwwDataInspector	The <i>TwwDataInspector</i> associated with this event
<i>ObjItem:</i> TwwInspectorItem	<i>TwwInspectorItem</i> associated with the data cell to be collapsed.
<i>CanCollapse:</i> Boolean	Set <i>CanCollapse</i> to false to prevent the item from being collapsed.

OnCanExpand

Write an *OnCanExpand* handler to prevent a node from being expanded by the user. This event is fired immediately after the end-user has tried to expand the item, but before a node is actually expanded. The end-user can expand an item by clicking on its expand button, or using a keyboard shortcut to expand the node. The keyboard shortcuts include Ctrl-RightArrow or RightArrow (ReadOnly items). Set *CanExpand* to false to prevent the item from being expanded and showing its children.

<i>Sender:</i> TwwDataInspector	The <i>TwwDataInspector</i> associated with this event
<i>ObjItem:</i> TwwInspectorItem	<i>TwwInspectorItem</i> associated with the data cell to be expanded.
<i>CanExpand:</i> Boolean	Set <i>CanExpand</i> to false to prevent the item from being expanded and showing its children.

OnCollapsed

Write an *OnCollapsed* event handler to perform your own action after an item has collapsed so that its children are no longer visible to the end-user.

Sender: TwwDataInspector The *TwwDataInspector* associated with this event
ObjItem: TwwInspectorItem *TwwInspectorItem* associated with the data cell that has collapsed.

OnCreateDateTimePicker

Write an *OnCreateDateTimePicker* event handler to customize the properties of the default datetimepicker used by the inspector. The default datetimepicker is automatically used by any row bound to a TDateTimeField. To disable the default datetimepicker for a row, set the item's *Options | iioAutoDateTimePicker* to false. The parameters for this event are as follows.

Sender: TwwDataInspector The *TwwDataInspector* associated with this event
ADateTimePicker: TwwDBCUSTOMDateTimePicker
The default datetimepicker. You can set its properties if you wish to change the properties of this control.

OnCreateDefaultCombo

Write an *OnCreateDefaultCombo* event handler to customize the properties of the default combobox used by the inspector when your item's PickList properties are assigned.

Sender: TwwDataInspector The *TwwDataInspector* associated with this event
Combo: TwwDBCcomboBox The default combobox auto-created and used by the inspector. You can set its properties if you wish to change the properties of this control.

OnCreateHintWindow

Use this event to customize the painting of the hint window. This event is fired before the hint window is actually displayed. The parameters are as follows:

Sender : TObject TwwDataInspector that is associated with this event.
HintWindow: TwwInspectorHintWindow
Hint window that was created. You can refer to its Canvas property to customize how the hint window is painted.
AField: TField Field that the hint window is displaying information about.
R: TRect Rectangle coordinates of the hint window
var *WordWrap:* Boolean Set *WordWrap* to True to cause the hint window to wordwrap
var *MaxWidth:* integer Set *MaxWidth* to limit the width of the hint window
var *MaxHeight:* integer Set *MaxHeight* to limit the height of the hint window
var *DoDefault:* Boolean Set *DoDefault* to False if you wish to prevent the datainspector from painting the hint window.

Example: The following code attached to this event makes the hint window's background `clYellow`.

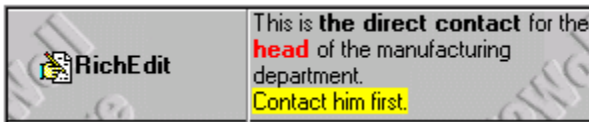
```
HintWindow.Color := clYellow;  
HintWindow.Canvas.Brush.color := clYellow;
```

OnDrawCaptionCell

Write an *OnDrawCaptionCell* event handler to customize the painting of the caption cells in the inspector. If you are just customizing the cells in the data column, you should instead use the *OnDrawDataCell* event. The parameters for this event are as follows:

<i>Sender</i> : <code>TwwDataInspector</code>	The <i>TwwDataInspector</i> associated with this event
<i>ObjItem</i> : <code>TwwInspectorItem</code>	<i>TwwInspectorItem</i> associated with the data cell to be painted.
<i>ASelected</i> : <code>Boolean</code>	True, if the row to be painted is the active row
<i>ACellRect</i> : <code>TRect</code>	<i>ACellRect</i> is the coordinates of the cell's rectangle with respect to the Inspector's top left position.
<i>var ACaptionRect</i> : <code>TRect</code>	<i>ACaptionRect</i> contains the rectangle coordinates of the area that inspector is to paint the text to. This differs from <i>ACellRect</i> as the text rectangle does not include the area to the left of the text. You may also wish to set <i>ACaptionRect</i> if you wish for an individual cell's caption to be placed at a different location. See the example described later in this event.
<i>var DefaultDrawing</i> : <code>Boolean</code>	Set to False to prevent the default painting from taking place.

Example: The following example paints an image from an imagelist to the left of caption. In this example it paints the image for the item associated with the row whose caption is 'RichEdit'.



```
procedure TForm1.wpDataInspector1DrawCaptionCell(Sender: TwwDataInspector;  
  ObjItem: TwwInspectorItem; ASelected: Boolean; ACellRect: TRect;  
  var ACaptionRect: TRect; var DefaultTextDrawing: Boolean);  
begin  
  if ObjItem.Caption = 'RichEdit' then begin  
    { Paint cell using ImageIndex of 3 }  
    ImageList1.Draw(Sender.Canvas, CaptionRect.Left, ACaptionRect.Top, 3);  
    ACaptionRect.Left := ACaptionRect.Left + ImageList1.Width+1;  
  end  
end;
```

OnDrawDataCell

Write an *OnDrawDataCell* event handler to customize the painting of the data cells in the inspector. If you are just customizing the cells in the caption column, you should instead use the *OnDrawCaptionCell* event. The parameters for this event are as follows:

<i>Sender</i> : TwwDataInspector	The <i>TwwDataInspector</i> associated with this event
<i>ObjItem</i> : TwwInspectorItem	<i>TwwInspectorItem</i> associated with the data cell to be painted.
<i>ASelected</i> : Boolean	True, if the row to be painted is the active row
<i>ACellRect</i> : TRect	<i>ACellRect</i> is the coordinates of the cell's rectangle with respect to the Inspector's top left position.
<i>var DefaultDrawing</i> : Boolean	Set to False to prevent the default painting from taking place.

Example: The following example paints the data cell containing the field 'State' with a yellow background if it's value is 'CA'.

```
with (Sender as TwwDataInspector) do
begin
  if (ObjItem.Field<>nil) and (ObjItem.field.fieldname='State') and
    (ObjItem.Field.asstring='CA') then
  begin
    Canvas.brush.color:= clYellow;
    Canvas.fillrect(ACellRect);
  end
end;
```

Example: The following example paints with a yellow font the data cells which have their item's *readonly* property as true.

```
if Objitem.readonly then sender.canvas.font.color:= clyellow;
```

OnDrawIndicatorCell

Write an *OnDrawIndicatorCell* event handler to customize the painting of the indicator cells in the inspector. The *IndicatorRow* property must be enabled and the *DataColumns* property must be greater than 1.

The parameters for this event are as follows:

<i>Sender</i> : TwwDataInspector	The <i>TwwDataInspector</i> associated with this event.
<i>ACol</i> : Integer	<i>ACol</i> indicates which indicator column is being painted..
<i>ACellRect</i> : TRect	<i>ACellRect</i> is the coordinates of the cell's rectangle with respect to the Inspector's top left position.
<i>var DefaultDrawing</i> : Boolean	Set to False to prevent the default painting from taking place.

Example: The following example paints the indicator column cell red when it is the active column of the TwwDataInspector control.

```

procedure TForm1.wpDataInspector1DrawIndicatorCell(Sender: TwwDataInspector;
ACol: Integer; ACellRect: TRect; var DefaultDrawing: Boolean);
begin
  if Sender.Col=ACol then begin
    Sender.Canvas.Brush.Color := clRed;
    Sender.Canvas.FillRect(ACellRect);
  end;
end;

```

OnExpanded

Write an *OnExpanded* event handler to perform your own action after an item has expanded to show its children.

<i>Sender</i> : TwwDataInspector	The <i>TwwDataInspector</i> associated with this event
<i>ObjItem</i> : TwwInspectorItem	<i>TwwInspectorItem</i> associated with the data cell that has expanded

OnItemChanged

Write an *OnItemChanged* event handler to perform your own action after the user has modified the text for a row.

<i>Sender</i> : TwwDataInspector	The <i>TwwDataInspector</i> associated with this event
<i>ObjItem</i> : TwwInspectorItem	<i>TwwInspectorItem</i> that has been changed.
<i>NewValue</i> : String	The new value of the row's data

OnTopLeftChanged

Write an *OnTopLeftChanged* event handler to perform any action when the inspector scrolls resulting in the top row changing.

OnValidationErrorUsingMask

Write an *OnValidationErrorUsingMask* event handler to perform any custom action after the user tries to leave the cell with a value that does not satisfy the picture mask constraints assigned for the cell.

The default behavior is to raise an exception with the message "Invalid input value. Use escape key to abandon changes".

The parameters are as follows:

<i>Sender</i> : TwwDataInspector	The <i>TwwDataInspector</i> associated with this event
<i>ObjItem</i> : TwwInspectorItem	<i>TwwInspectorItem</i> whose edited value does not satisfy the picture mask constraints
<i>Var Msg</i> : String	You can set this value to change the actual message used by the default error handler.

Var DoDefault: Boolean Set this to *False* to prevent the default handler from executing. The default error handler raises an exception with the message defined by *Msg*

Example: The following example changes the error message from the default string to your own message.

```
procedure TForm1.TwwDataInspector1ValidationErrorUsingMask(
  Sender: TwwDataInspector; Item: TwwInspectorItem;
  var Msg: String; var DoDefault: Boolean);
begin
  Msg:= 'Characters are not valid!';
end;
```

Added Methods

BeginUpdate

Call this method to suspend painting operations to the inspector. You may wish to call this method if you are performing many lengthy operations on the inspector as this may improve the performance of those operations.

```
procedure BeginUpdate;
```

EndUpdate

Call this method to resume painting operations to the inspector after a call to *BeginUpdate* has suspended them. Set *Repaint* to true if you wish for the entire tree inspector to be repainted.

```
procedure EndUpdate(Repaint: boolean = false);
```

GetFirstChild

Call this method to get the first child of the inspector. If *VisibleItems* is true, this method ignores items whose *Visible* property is false.

```
function GetFirstChild(VisibleItemsOnly: boolean = True):
  TwwInspectorItem;
```

GetItemByFieldName

Returns the *TwwInspectorItem* associated with the field name specified by *AFieldName*

```
Function GetItemByFieldName(AFieldName: string): TwwInspectorItem;
```

GetItemByRow

Returns the *TwwInspectorItem* associated with the row specified by *ARow*

```
Function GetItemByRow(ARow: integer): TwwInspectorItem;
```

GetItemByCaption

Returns the *TwwInspectorItem* associated whose *Caption* property matches the parameter *ACaption*.

```
Function GetItemByCaption(ACaption: string): TwwInspectorItem;
```

GetItemByTagString

Returns the *TwwInspectorItem* associated whose *TagString* property matches the parameter *ATagString*.

```
Function GetItemByTagString(ATagString: string): TwwInspectorItem;
```

GetRowByItem

Returns the row number a *TwwCollectionItem* is appearing in for the inspector.

```
Function GetRowByItem(AItem: TwwInspectorItem): integer; virtual;
```

HaveVisibleItem

Returns True if the inspector has at least one visible item.

```
function HaveVisibleItem: boolean;
```

InvalidateRow

Call this method to invalidate a row in the inspector so that it is repainted.

```
procedure InvalidateRow(ARow: integer);
```

MouseToCell

This event converts the screen coordinates defined by the parameter *X,Y* to inspector cell coordinates in *ACol, ARow*. You may wish to use this method if you want to determine which row and column of the inspector the mouse is over.

```
procedure MouseToCell(X, Y: Integer; var ACol, ARow: Longint);
```

MouseToItem

This event returns the inspector item correlating with the screen coordinates defined by the parameters *X,Y*. You may wish to use this method if you want to determine which item is associated with the mouse position.

```
function MouseToItem(X, Y: Integer): TwwInspectorItem;
```

How To

Set the active row in the inspector

You can set the active row in the inspector based on a number of different properties. Use the *GetItemByFieldName*, *GetItemByRow*, *GetItemByCaption*, and *GetItemByTagString* methods to return an item based on the information you have. Then set the inspector's *ActiveItem* property.

```
with wwDataInspector1 do
begin
  ActiveItem:= GetItemByCaption('MyCaption');
end
```

Embedding 3rd Party Controls.

Each item of the `TwwDataInspector` can have a custom control attached to it. Many 3rd Party controls will work right away, but testing is required to see how well those controls work in the `datainspector`. This is especially true if you are using the controls in a data-aware multi-column `TwwDataInspector`. When assigning the control to the item at design-time all you need to do is select the control from the dropdown list in the object inspector for that particular item.

Hide the Caption Column

To hide the caption column all you need to do is set the `Options | ovHideCaptionColumn` to `True`. This is especially useful when embedding a `TwwDataInspector` in the `TwwDBGrid`. See the `TwwDBGrid` *How To* section for an example of this.

Add a background image

See the `PaintOptions` property

Conditionally color the items

See the examples in the inspector's `OnDrawDataCell` event.

Iterate through the items

See the example in the method `GetNext`

Change the expand and collapse glyphs used by the inspector

See the `ButtonOptions` property

Associate a picklist with an inspector item.

See the `TwwInspectorItem` `PickList` property

Display a checkbox for an item

See the `TwwInspectorItem.PickList.DisplayAsCheckbox` property. Alternatively you can embed a `TwwCheckbox` control as a custom control by using the `CustomControl` property.

TwwInspectorCollection

The following methods are part of TwwInspectorCollection, which you can use to manipulate the inspector's collection of items during runtime. See also TCollection for other methods inherited from the base class TCollection.

Ancestor

TCollection

Added Properties

None

Added Methods

Add

Call this method to add a new TwwInspectorItem to the data inspector's collection of items. The return value is the inspector item that was added.

```
function Add: TwwInspectorItem;
```

Example: The following example creates 50 rows in the data inspector, with each row being displayed as a checkbox.

```
for i:= 1 to 50 do begin
  with wwdatainspector1.items.Add do
    begin
      Caption:= 'Checkbox #' + inttostr(i);
      PickList.DisplayAsCheckbox:= True;
      PickList.Items.Add('True');
      PickList.Items.Add('False');
    end
  end;
wwdatainspector1.invalidate;
```

LoadFromFile

Call this method to load the current information contained in the inspector's collection from a file. This can be useful if your inspector is not tied to a datasource. By calling this method and the *SaveToFile* method, you can save and load the user's runtime changes.

```
procedure LoadFromFile(const FileName: string);
```

LoadFromStream

Call this method to load the current information contained in the inspector's collection from a file. This can be useful if your inspector is not tied to a datasource. This method should be

used in conjunction with the *SaveToStream* method. See the Delphi documentation on TStream for more information on stream manipulation.

```
procedure LoadFromStream(s: TStream);
```

Insert

Call this method to insert a new TwwInspectorItem to the data inspector before the item specified by index.

```
function Insert(index: integer): TwwInspectorItem;
```

SaveToFile

Call this method to save the current information contained in the inspector's collection to a file. This can be useful if your inspector is not tied to a datasource. By calling this method and the *LoadFromFile* method, you can save and load the user's runtime changes.

```
procedure SaveToFile(const FileName: string);
```

SaveToStream

Call this method to save the current information contained in the inspector's collection to a stream. This can be useful if your inspector is not tied to a datasource. By calling this method and the LoadFromStream method, you can save and load the user's runtime changes. See the Delphi documentation on TStream for more information on stream manipulation.

```
procedure SaveToStream(s: TStream);
```

TwwInspectorItem

The following properties and methods are part of TwwInspectorItem, which you can use to manipulate an individual inspector collection item during runtime. See also TCollectionItem for other methods inherited from the base class TCollectionItem.

Each row in the data inspector is associated with a TwwCollectionItem. You can manipulate the properties at runtime by dbl-clicking on the inspector at design time.

Added Properties

Alignment

The *Alignment* property sets the default alignment of the text in the data portion of the TwwDataInspector.

Data Type: TAlignment

Caption

The *CaptionColor* property defines the label that appears in the first column of the data inspector.

Data Type: String

CellHeight

Assign this property to change the height of an individual row in the grid. Defaults to 0, which means it will use the data inspector's *DefaultRowHeight* property.

Data Type: Integer

Checked (Runtime only)

This property returns the current state of an item whose checkbox is enabled (See the PickList property). You can also assign this property to set or clear the checkbox.

Data Type: Boolean

CustomControl

Assign this property to attach a custom edit control to a data cell in the inspector. This custom control is used when the cell receives focus. The custom control also handles the painting of the inspector's cell when it does not have focus if *CustomControlAlwaysPaints* is set to True.

InfoPower allows you to select a wider variety of controls to embed in the inspector. However not all controls will behave well. The InfoPower TwwDBEdit, TwwCheckbox, TwwRadioGroup, TwwDBSpinEdit, TwwDBComboBox, TwwDBDateTimePicker, TwwDBLookupCombo, 1stClass edit controls, and the Delphi TDBImage are supported. The TwwDBGrid and TwwDataInspector are not currently supported as custom controls. Some 3rd party controls will behave well, but you will need to experiment to determine the ones that do. A smaller subset of 3rd party controls will work when

CustomControlAlwaysPaints is true, so you may wish to set *CustomControlAlwaysPaints* to False before experimenting.

Customizing the custom control: If you have previously assigned the *CustomControl* property, you can select this control from the inspector's collection editor, by holding down the ALT key before you click on the inspector item. After doing so, the object inspector will show you the properties and events of the custom control.

Warning: You should not attach the same custom control to more than one row in the inspector when *CustomControlAlwaysPaints* is set to True. Otherwise the painting of the cells sharing the same custom control is not reliable. When *CustomControlAlwaysPaints* is set to False, then you can share the same custom control with more than one row.

Data Type: TCustomEdit

CustomControlAlwaysPaints

Set this property to False if you wish for the inspector to paint the cell as simple text. When this property is true, then the custom control paints the cell instead of the inspector. This allows controls that display icons to appear in the inspector even when they do not have focus. You may wish to set this property to false if you want the inspector to paint the cell instead of the custom control. This allows you the ability to share a single custom control with more than one row in the inspector. See also the *CustomControl* property.

Warning : If this value is True, then you should not share the same custom control with more than one row. If you do, the painting of the cells sharing the same custom control may display the wrong text.

Data Type: boolean

CustomControlHighlight

Default is False. When set to True a frame is drawn around the active customcontrol in a multiple column TwwDataInspector control. This may be helpful when trying to indicate which column/cell has the focus for example by drawing a frame around an embedded TDBImage control.

DataField

Assign this property to bind the row to a field in the datasource specified by the *DataSource* property.

Data Type: String

DataSource

This property specifies the datasource associated with the *DataField* property. This property defaults to the inspector's *DataSource* property. If you wish to attach the inspector item to another datasource, then set this property.

Data Type: TDataSource

DisableDefaultEditor

Set to true to disable the default editor from being used in the row when it gets the focus. This causes the control to display in the same background as the inspector even when it gets the focus. This property is ignored if you have attached a custom edit control.

Data Type: boolean

DisplayText (Runtime)

Returns the display text for an item that is mapped using the `PickList.MapList` property. If the item's `PickList.MapList` is false, then this property is equivalent to the `EditText` property.

Data Type: String

Enabled

Set this property to false to disable the inspector item from being edited or receiving the focus when it is tabbed to. In addition, the inspector will attempt to paint the cell using the system disabled color.

EditText

The current value for the item. After the user edits the text in the cell, this property is updated. If you are using a mapped picklist, then this item represents the stored value, not the displayed value. See the `DisplayText` to retrieve the display text.

Data Type: String

Expanded

When true, the current item is expanded to show its children. This property only applies to inspector items, which contain children items.

Data Type: Boolean

Field (Runtime)

Returns the `TField` associated with this row.

Data Type: `TField`

Items (Runtime)

Collection, containing inspector items that are the immediate children of this item.

Data Type: `TwvInspectorCollection`

Level (Runtime)

Returns the hierarchical level of this row. If this is a root item, then level returns 0.

Data Type: Integer

Options

Assign this property to change selected behavior of this inspector item.

Data Type: `TwvInspectorItemOptions`

<i>iiioAutoDateTimePicker</i>	Set to False to prevent the automatic creation of a date time editor when the inspector detects that the row is associated with a date or time field.
<i>iiioAutoLookupCombo</i>	Set to True to have the data inspector automatically create a lookupcombo control if this row is associated with a lookup field.

ParentItem (Runtime)

This property returns a handle to the parent item. *ParentItem* is nil if this item has no parent.

Data Type: TwwInspectorItem

PickList

Use this property to define a custom combo list or checkbox for the row.

Data Type: TwwInspectorPickList

The sub-properties of PickList are detailed below.

AllowClearKey	See the TwwDBComboBox AllowClearKey property
ButtonStyle	See the TwwDBComboBox ButtonStyle property
DisplayAsCheckbox	Set to True to display the item as a checkbox. The first two strings in the items property are used as the checked and unchecked values. To initialize the checkbox, set the item's EditText property.
Items	See the TwwDBComboBox Style property
MapList	See the TwwDBComboBox Style property
ShowMatchText	See the TwwDBComboBox ShowMatchText property
Style	See the TwwDBComboBox Style property

Picture

Assign this property if you wish to use a picture mask when editing this inspector item. Please reference chapter 4, *Selecting a Picture Mask* for details on this property.

Data Type: TwwDBPicture

ReadOnly

Set to True to disable editing for this row. When a row is readonly and it has children items, the inspector will allow the right and left arrow keys to expand and collapse the item.

Data Type: Boolean

Resizable

Set to True to allow the row to be resized at runtime

Data Type: Boolean

TabStop

Set to False to disable the row associated with this item as a tabstop

Data Type: Boolean

Tag

Integer field in which you can use for your own purposes.

Data Type: Integer

TagString

String field in which you can use for your own purposes.

Data Type: String

Visible

Set to False to disable the display of this item and its children.

Data Type: Boolean

WordWrap

When true, the text for the item supports wordwrapping.

Data Type: Boolean

Added Events

OnEditButtonClick

Write an *OnEditButtonClick* event handler to perform your own action when the end-user clicks on a button when the *PickList | ButtonStyle* is set to *cbsEllipsis* or if it is set to *cbsCustom* and the *PickList | ButtonGlyph* is assigned and the .

Sender: *TwwDataInspector* The *TwwDataInspector* associated with this event

Item: *TwwInspectorItem* *TwwInspectorItem* which fired this event.

OnItemChanged

Write an *OnItemChanged* event handler to perform your own action after the user has modified the text for a row. This event performs the same function as the *DataInspector*'s *OnItemChanged*, except this event is fired only when its related item is modified.

Sender: *TwwDataInspector* The *TwwDataInspector* associated with this event

ObjItem: *TwwInspectorItem* *TwwInspectorItem* which has been changed.

NewValue: string The new value of the row's data

Added Methods

Some of the methods below refer to the parameters *VisibleItemOnly* and *ExpandedOnly*. These parameters are defined below.

- VisibleItemsOnly** If *VisibleItemsOnly* is False, then the method ignores the item's visible property. The default value is True, which indicates that only items whose visible property are True are considered by the method.
- ExpandedOnly** If *ExpandedOnly* is True, then items that are not in an expanded branch or part of a root node, are ignored. The default value is false, which indicates that the method will include non-expanded nodes.

GetFirstChild

Call this method to retrieve this item's first child item. If no child is found, then nil is returned.

```
function GetFirstChild(
    VisibleItemsOnly: boolean = True;
    ExpandedOnly: boolean = False): TwwInspectorItem;
```

GetLastChild

Call this method to retrieve this item's last child item. If no child is found, then nil is returned.

```
function GetLastChild(
    VisibleItemsOnly: boolean = True;
    ExpandedOnly: Boolean = False): TwwInspectorItem;
```

GetNext

Call this method to retrieve the inspector item immediately following this item. This method will include child items as well as parent items. This method is useful for iterating through the data inspector's entire list of inspector items. See the *GetNextSibling* method to get the next item in the same level.

```
function GetNext(
    VisibleItemsOnly: boolean = True;
    ExpandedOnly: Boolean = False): TwwInspectorItem;
```

Example: The following code iterates through all the inspector items and displays the caption property for each one.

```
var
    item: TwwInspectorItem;
begin
    item:= wwdatainspector1.getfirstchild;
    while item<>nil do
        begin
            showmessage(item.caption);
            item:= item.getnext;
        end;
    end
```

GetNextSibling

Call this method to retrieve this item's next sibling.

```
function GetNextSibling(
    VisibleItemsOnly: boolean = True): TwwInspectorItem;
```

GetPrior

Call this method to retrieve the inspector item immediately above this item. This method will include child items as well as parent items. See the `GetPriorSibling` method to get the prior item in the same level.

```
function GetPrior(  
    VisibleItemsOnly: boolean = True;  
    ExpandedOnly: Boolean = False): TwwInspectorItem;
```

GetPriorSibling

Call this method to retrieve this item's prior sibling.

```
function GetPriorSibling(  
    VisibleItemsOnly: boolean = True): TwwInspectorItem;
```

TwwDataSource



Provided for backwards compatibility. InfoPower allows you to directly use the TDataSource component instead, so this component is no longer necessary when using InfoPower controls.

Ancestor

TDataSource

Required property assignments

Dataset.

Added events

None.

How To

InfoPower's TwwDataSource component functions in the same manner as Delphi's TDataSource component. Please refer to your Delphi manuals for more information about this component.

Tips

Since InfoPower's TwwDataSource component is a direct descendent of Delphi's TDataSource component, you are provided with 100% backward compatibility. Thus, you can safely replace your use of TDataSource with TwwDataSource at any time.

TwwDBComboBox



InfoPower greatly expands the capabilities of a regular data aware combo-box. It has the following advantages over Delphi's TDBComboBox.

- InfoPower gives its combo-box the ability to remember the user's previously entered values. The next time the user's program is executed, these previously entered values are automatically filled into the combobox's dropdown list. You can also specify a separate MRU list so that the most recently entered entries appear at the top of the list.
- Allows you to enter mapped storage and display values so that you can display understandable text versions of stored codes in your table, instead of displaying only the codes themselves where users have to remember what they all mean. Alternatively you could use a TwwDBLookupCombo to display one field from a LookupTable, and store a different field. However the TwwDBLookupCombo approach requires greater complexity as a LookupTable is required to fill the drop-down list. The TwwDBComboBox's drop-down list comes directly from a string list. Use the *TwoColumnDisplay* property if you wish to display both the code and display value in the drop-down list.
- The glyph in the combo is configurable through the control's ButtonGlyph and ButtonWidth properties.
- Supports the '*Quicken*' style display of the matching value, by simultaneously searching and displaying the matching text in the search controls. Set the ShowMatchText property to True to achieve this effect.
- Use the AllowClearKey property to give your end-user's a convenient way of clearing the combo's selection.
- Since InfoPower is derived from the InfoPower base editor class, you can access the following additional properties which are not available in Delphi's TDBComboBox: *AutoSelect*, *AutoSize*, *BorderStyle*, *CharCase*, *MaxLength*, and InfoPower's *Picture* property.



Figure 5.1 - The TwwDBComboBox component.

Ancestor

TwwDBCustomEdit

└ TwwDBCustomCombo

└ TwwDBCustomComboBox

Added properties:

This component has the properties of the TwwDBEdit, plus the following additional properties:

AllowClearKey

When the ComboBox style is set to csDropDownList, the user is not able to clear their selection. The *AllowClearKey* property when set to True, gives the user a convenient way to clear the combos current selection simply by entering either the or <BACKSPACE> character.

Data Type: Boolean

AutoDropDown

When True, the lookup list drops down automatically when a keystroke is entered. The default value is False.

Data Type: Boolean

ButtonEffects

See the topic “Key properties for enabling custom button effects in the edit controls” in chapter 4 for information on this property.

Data Type: TwwButtonEffects

ButtonGlyph

This property defines the custom bitmap used for the icon in the control when ButtonStyle is set to cbsCustom.

Data Type: TBitmap

ButtonStyle

This property defines the icon used for this component. If the property *ShowButton* is False, then this property is ignored.

Data Type: TwwComboButtonStyle

Valid Values: cbsEllipsis, cbsDownArrow, cbsCustom

cbsDownArrow The  bitmap is displayed

cbsEllipsis The  bitmap is displayed

cbsCustom: The icon defined by the ButtonGlyph property.

ButtonWidth

This property defines the width of the icon for the control. You may wish to set this property if your custom bitmap assigned to the ButtonGlyph property is larger than the default button width This property defaults to 0, which indicates to the control to compute the button width based on the system settings..

Data Type: Integer

Column1Width

Use the *Column1Display* property to set the number of pixels to occupy for column 1. This property is ignored unless *TwoColumnDisplay* is true.

Data Type: Boolean

DisableDropDownList

Set this property to True to disable the drop-down list from appearing. You can then use the OnDropDown event to perform your own action, or bring up your own custom dialog.

Data Type: Integer

DisableThemes

If your project has enabled XP themes but you do not wish for this control to be theme-enabled, then set this property to *False*.

DropDownCount

This property determines how many entries will appear in the drop-down list box.

Data Type: Integer

DropDownWidth

The *DropDownWidth* property determines how wide the drop-down list box is in pixels. The default value is 0, which will automatically size the box based on the width of the control.

Data Type: Integer

DroppedDown

Run-time only. The *DroppedDown* property determines whether the drop-down list of the combo box is open or closed. If *DroppedDown* is True, the drop-down list is visible. If *DroppedDown* is False, the drop-down list is closed.

Data Type: Boolean

Frame

See the topic “Key properties and events for custom framing” in chapter 4 for more information on this property.

Data Type: TwwEditFrame

HistoryList

This property contains information on managing the built-in history list. The TwwDBComboBox control supports automatic management of a history list to reflect the end-user’s previously entered values. This convenient feature can be managed in a number of different ways.

To enable history lists, set the Enabled property. This will tell the combo to remember values the user types into the control. These values are remembered until the program exits. If you wish to have these values remembered the next time the user starts the program, then you will

need to define the location of where the history list is stored. This is done by setting the *StorageType* property in conjunction with setting the *Section* and *FileName* properties.

You can also enable a most-recently-used list that appears at the top of the combo control. To enable this, you will need to have both the *Enabled* and *MRUEnabled* properties set to True.

Data Type: TwwHistoryList

Enabled	Set to True to enable the combo's history list
FileName	Set this property to indicate the INI filename, or the registry location to store the history list items.
MaxSize	Set this property to limit the size that the history list can grow to. For instance, if you wish for the control to remember only the 100 most recent entries, then set this property to 100. This property defaults to -1, which indicates that the history list does not limit its size.
MRUEnabled	Set to True to enable the combo's MRU (most-recently-used) list. When enabled, the combo control will display the most recently selected values at the top of the drop-down combo control. This property is ignored if the History Enabled property is set to false.
MRUMaxSize	Set this property to specify the maximum number of entries to show in the MRU section of the combo's drop-down list.
Section	If <i>StorageType</i> is set to <i>stIniFile</i> , then this property indicates the section within the INI filename to store the history list. If <i>StorageType</i> is set to <i>stRegistry</i> , then this property indicates the key within the registry path to store the history list.
StorageType	Set to <i>stIniFile</i> to store the history list to an IniFile. Set to <i>stRegistry</i> to store them in your registry. You also need to set the <i>FileName</i> and <i>Section</i> properties in order for the history list to be saved when your program exits.

ItemHeight

The *ItemHeight* property is the height of an item in the combo box list in pixels when the combo box's *Style* property is *csOwnerDrawFixed*. If the *Style* property is any other setting, the value of *ItemHeight* is ignored.

Data Type: Integer

Items

The default property editor dialog box for the *Items* property was redefined in InfoPower to allow you to enter either a single list of values used for both the Displayed and Stored Values, or a two-column list of values for both Displayed and Stored Values. The dialog box is now called Edit Combo List. The *MapList* property can be set via the **Map Displayed Value to Stored Value** check box contained within the dialog box (see the *MapList* property described above).

When the Map Displayed Value to Stored Value check box is checked (or the *MapList* property is set to True), the dialog box contains two columns: Displayed Value and Stored Value, where the Displayed Value is always shown on the display screen and the Stored Value is always stored to the database table. When the check box is unchecked (or the *MapList* property is set to False), the dialog box contains only a single list of strings which defines both the Displayed Value *and* Stored Value that are to be used. See the *How To* section below for details on using the Items property.

Data Type: TStrings

LimitEditRect

Set this property to true if you wish to force the combobox's editing rectangle to not overlap the icon in the control. The negative consequence of this being set to true is that the combobox will no longer close the modal form on an escape, as the escape goes to the control instead.

MapList

This property defines whether or not the display values specified in the *Items* property are mapped to a corresponding Stored Value assignment. When True, the *Items* property dialog box will display two columns of information, a Displayed Value and a Stored Value. The Displayed Value is always shown on the display screen and the Stored Value is always stored to the table. When False, the single list of display values you enter is used for both display *and* storage purposes. See the *How To* section below for details on using the *MapList* property.

Data Type: Boolean

ShowButton

When this property is False, the combo's bitmap button is not shown. The default value is True.

Data Type: Boolean

ShowMatchText

When this property is True this combo will have Quicken Style incremental searching by simultaneously searching and displaying the matching text in the search control. The default value is False.

Data Type: Boolean

Sorted

The *Sorted* property indicates whether the items in a list box or combo box are arranged alphabetically. To sort the items, set the *Sorted* value to True. If *Sorted* is False, the items are unsorted. If you add or insert items when Sorted is True, InfoPower automatically places them in alphabetical order. Defaults to False.

Data Type: Boolean

Style

This property determines the style of the ComboBox. The *csDropDown Style* creates a drop-down list with an edit box in which the user can enter text. The *csDropDownList Style* creates

a drop-down list with no attached edit box, so the user can't edit an item or type in a new item. Please refer to the Delphi documentation for more information on details of `TComboBox` *style* property.

Note: When the `ShowMatchText` property is `False` and `Style` is `csDropDownList`, `InfoPower` now adheres to the Windows combobox search behavior where the entered character is used to find a match starting with that one character. Set `ShowMatchText` to `true` if you desire continuous incremental searching where all entered characters are used to search the list.

Data Type: `TComboBoxStyle`

Valid Values: `StdCtrls.csSimple`, `StdCtrls.csDropDown`, `StdCtrls.csDropDownList`, `StdCtrls.csOwnerDrawFixed`, `StdCtrls.csOwnerDrawVariable`.

Example: When setting the `Style`, you may need to scope the value when you do the assignment.

```
wwDBComboBox1.Style := StdCtrls.csDropDown;
```

TwoColumnDisplay

When `TwoColumnDisplay` is `True`, both the mapped and stored values are displayed in the drop-down list. Use the `Column1Display` property to set the number of pixels to occupy for column 1.

Data Type: `Boolean`

Value

When `MapList` is `True`, this property represents the hidden stored value. When `MapList` is `False`, this property is equivalent to the text property. Setting this property will also update the controls `ItemIndex` property.

Required property assignments

Items.

Added Events

This component has all the events of the `TwwDBEdit`, plus the following additional events.

OnAddHistoryItem

This event allows you to perform some custom action when an enduser has typed in a new entry into the control and a new item is about to be saved to the history list when `HistoryList.enabled` is `true`. Set `Accept` to `False` to prevent the item from being added to the history list.

Parameter	Description
<i>Value:</i> <code>String</code>	New string that is about to be added to the History List.
<i>Accept:</i> <code>Boolean</code>	Set to <code>False</code> if you wish to prevent this item from being added to the History list.

OnCloseUp

The *OnCloseUp* event occurs when the user closes a combo box's drop down list.

Sender : TwwDBComboBox The TwwDBComboBox that is being closed up.
Select : Boolean *False*, if the user closed the drop-down list with the Escape key.

OnDrawItem

Please see the Delphi documentation for information on this event.

OnDropDown

The *OnDropDown* event occurs when the user opens (drops down) a combo box.

Added Methods

AddItem

Call this method to add an item from the drop-down list. If *AddToHistory* is true, then the item is also added to the history list. Note: Do not use the form's *OnCreate* event to add items as the added items would be overwritten when the combo's properties are streamed in. If you wish to add items when your form is opened, then use the form's *OnShow* event. Note: You can also manipulate the *Items* property if you wish to manipulate the items as a string list.

```
procedure AddItem(Value: string; AddToHistory: boolean = False)
```

ApplyList

Call this method if you manipulate the *Items* property during program execution. This applies any run-time changes made to the *Items* property by updating the combo's drop-down list.

```
Procedure ApplyList;
```

ClearHistory

Call this method to clear the items in the history list. After calling this method, the user's history of entered items is gone.

```
Procedure ClearHistory;
```

DeleteItem

Call this method to remove an item matching the string *Value* from the drop-down list. If *DeleteFromHistory* is true, then the item is also removed from the history list.

```
procedure DeleteItem(Value: string;  
                    DeleteFromHistory: boolean = False)
```

DroppedDown

Returns *True* if the list is currently dropped down.

GetComboDisplay

Use this to retrieve the display text of a mapped list that correlates with a given stored value.

```
Function GetComboDisplay(Value: string): string;
```

GetComboValue

Use this to retrieve the stored value of a mapped list that correlates with the give display value.

```
Function GetComboValue(DisplayText: string): string;
```

How To

Use the Items property and Edit Combo List dialog box:

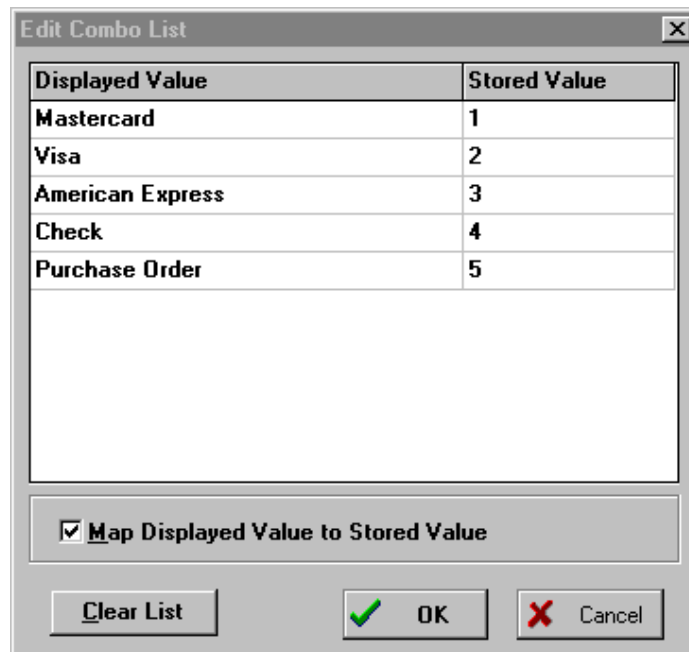


Figure 5.2 - Edit Combo List dialog box

Adding and removing entries from the Edit Combo List dialog box is easy. The first time this dialog box is displayed, the first entry is blank so you can simply enter the necessary value(s). When you want to add an entry in between two existing entries, position the highlight to the entry just below where you want the new entry inserted and press the Insert key. This adds a blank entry immediately *above* the currently highlighted entry. To remove the currently selected entry, press the Ctrl+Delete keys.

Example: If your application involves the storage of *payment type* information, you can save disk storage space by assigning a series of integer values, or other single character codes, to each of the allowable payment types you accept. You would first set the *MapList* property to True. Then, via the Edit Combo List dialog box (*Items* property editor), you would enter the Displayed Value and Stored Value assignments to look something like the following:

<u>Displayed Value</u>	<u>Stored Value</u>
MasterCard	1
Visa	2
American Express	3
Check	4

Both the Displayed Value and Stored Value assignments are string values, providing you with the most flexible options possible, so you can enter any type of data (text, numbers, etc.) you need into either column.

If you wanted to store the entire Displayed Value string in your table (i.e. “Visa” instead of “2”), you would first set the *MapList* property to False and then, via the Edit Combo List dialog box, you would enter the list of Displayed Value assignments shown above. This single list will then be used for both display and storage purposes.

Change the Displayed and Stored Values of the Items property at runtime:

To change the Displayed and Stored Values programmatically at runtime, you must first realize that the *Items* property value is merely a TStrings data type with tab character delimiters between each Displayed Value and Stored Value entry. Thus, to define the above example at runtime, you would use the following Object Pascal code, where the entry #9 specifies the tab character (extra white space was added simply to make the example easier to read):

```
myComboBox.Items.Clear;
myComboBox.Items.Add('MasterCard' + #9 + '1');
myComboBox.Items.Add('Visa' + #9 + '2');
myComboBox.Items.Add('American Express' + #9 + '3');
myComboBox.Items.Add('Check' + #9 + '4');
myComboBox.Items.Add('Purchase Order' + #9 + '5');
myComboBox.ApplyList;
```

After changing the *Items* property, be sure to call the method *ApplyList* to have your changes reflected in the component.

Displaying Two Columns

It is sometimes desirable to display the code along with the longer form of the string. For example, one might wish to store a U.S. State abbreviation as a 2 Character field in the database, but display the long state name to the enduser in the combo. When the combo is dropped down it is possible to display both the state and the abbreviation in the drop down list.

To do this, use the Edit Combo List dialog box (*Items* property editor), set *MapList* to True, and then enter the Displayed Value and Stored Value assignments like the following:

<u>Displayed Value</u>	<u>Stored Value</u>
California	CA
Colorado	CO
Connecticut	CT
Deleware	DE
Florida	FL
Georgia	GA

Texas

TX

Then close the dialog and set *TwoColumnDisplay* to True. For optimal display you may wish to set the *DropDownWidth* property (i.e. 200) and the *Column1Width* property (i.e. 125) based on your data.

Caution

When the *Style* property is set to *csDropDown*, this normally allows the end-user to manually enter a value that is not listed in the drop down list box, via the edit box portion of the component. However, when *MapList* is set to True and the *csDropDown Style* is selected, the end-user may only select a value from the contents of the drop down list box, which is similar to the behavior provided when the *Style* property is set to *csDropDownList*.

TwwDBComboDlg



TwwDBComboDlg is a visual interface component that looks and behaves similar to a DBComboBox edit component, in that it allows the user to enter and edit data in the edit box portion of the component. However, when the user clicks the component's "..." button, instead of the normal drop-down list being displayed, any program-controlled action you define in the *OnCustomDialog* event can take place, such as displaying a custom dialog box of your own design.



Figure 5.3 - The *TwwDBComboDlg* component.

Ancestor

TwwDBCustomEdit

└ TwwDBCustomCombo

Added Properties

In addition to all the properties of the TwwDBEdit, this component also has the following additional properties.

AutoEnableEdit

Set this property to False if you wish to prevent the control from automatic enabling the dataset's edit state when the icon is clicked.

Data Type: Boolean

ButtonEffects

See the topic "Key properties for enabling custom button effects in the edit controls" in chapter 4 for information on this property.

Data Type: TwwButtonEffects



ButtonGlyph

This property defines the custom bitmap used for the icon in the control when ButtonStyle is set to cbsCustom.

Data Type: TBitmap

ButtonStyle

This property defines the icon used for this component. If the property *ShowButton* is False, then this property is ignored. The following are the possible values:

<i>cbsDownArrow</i>	The  bitmap is displayed
<i>cbsEllipsis</i>	The  bitmap is displayed
<i>cbsCustom</i>	The icon defined by the ButtonGlyph property.

Data Type: TwwComboButtonStyle

Valid Values: cbsEllipsis, cbsDownArrow, cbsCustom

ButtonWidth

This property defines the width of the icon for the control. You may wish to set this property if your custom bitmap assigned to the ButtonGlyph property is larger than the default button width. This property defaults to 0, which indicates to the control to compute the button width based on the system settings..

Data Type: Integer

Frame

See the topic “Key properties and events for custom framing” in chapter 4 for more information on this property.

Data Type: TwwEditFrame

LimitEditRect

Set this property to true if you wish to force the combobox’s editing rectangle to not overlap the icon in the control. The negative consequence of this being set to true is that the combobox will no longer close the form on an escape, as the escape goes to the control instead.

ShowButton

When this property is False, the combo's bitmap button is not shown. The default value is True. When the icon is clicked the *OnCustomDlg* event is called.

Data Type: Boolean

Style

When *Style* is set to csDropDown, the end-user can directly edit the control. If *Style* is set to csDropDownList the user cannot edit by typing into the control.

Data Type: TwwDBLookupComboStyle

Valid Values: Wwdblook.csDropDown, Wwdblook.csDropDownList

Example: When setting the Style property for wwDBComboDlg, wwDBLookupCombo, wwDBLookupComboDlg, or the wwDBLookupCombo you may need to scope the value when you do the assignment.

```
wwDBComboDlg1.Style := Wwdblook.csDropDown;
```

Required property assignments

Custom code in the *OnCustomDlg* event.

Added Events

In addition to all the events of the `TwwDBEdit`, this component also has the following additional event.

OnCustomDlg

This event is a modified version of the `OnDropDown` event and is executed when the user clicks the “...” button of the component.

Example: The following example demonstrates how the `TwwDBComboDlg` component can be used, assuming the `TwwDBComboDlg` component is named `UserResponse`. Add the following code to the `OnCustomDlg` event:

```
if MessageDlg('Click Yes or No', mtConfirmation, [mbYes, mbNo], 0) = mrYes
then
    UserResponse.Text := 'Yes'
else
    UserResponse.Text := 'No';
```

When the user clicks the “...” button, Delphi’s Confirm message dialog is displayed with the message “Click Yes or No”, as shown in Figure 5.4.

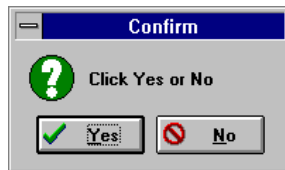


Figure 5.4 - A simple Delphi Confirm message dialog box.

If the user clicks the Yes button, the text of the `TwwDBComboDlg` component is set to “Yes”. Otherwise, the text is set to “No”. You can display a custom dialog box instead of a message dialog box by replacing the call to `MessageDlg` with your own dialog box call. The actual syntax you use depends on the value(s) returned by your dialog box.

TwwDBDateTimePicker



InfoPower's TwwDBDateTimePicker is the ideal component for entering and selecting a date or a time value.

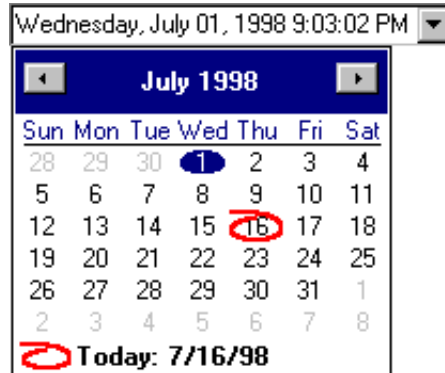


Figure 5.5 – TwwDBDateTimePicker control

InfoPower's version includes the following functionality.

- When used against a date, it has a built in drop-down calendar for selecting a date.
- Embed within InfoPower's Grid and RecordView components
- Use with or without a database.
- Smart data entry: auto-advances when enough characters have been entered, and auto-fills the date and time when the space key is entered.
- Display the date in the format of your choice using a format mask. Also supports International date-time formats
- Spins up/down with the arrow keys and preserves the display format during spinning.
- Numerous display options for controlling the look of the drop-down calendar, such as display of week numbers, display of current date, automatic circling of the current date, and support for event based determination of which dates should be in bold.
- Year 2000 compliance
- Support for simultaneous entry of both date and time in the same control
- Support for custom framing, transparency, and custom glyphs

Ancestor

TCustomEdit

└ TwwCustomDateTimePicker

Required supporting components

None

Added Properties

ButtonEffects

See the topic “Key properties for enabling custom button effects in the edit controls” in chapter 4 for information on this property.

Data Type: TwwButtonEffects



ButtonGlyph

This property defines the custom bitmap used for the icon in the control when ButtonStyle is set to cbsCustom.

Data Type: TBitmap

ButtonStyle

This property defines the icon used for this component. If the property *ShowButton* is False, then this property is ignored. The following are the possible values:

<i>cbsDownArrow</i>	The  bitmap is displayed
<i>cbsEllipsis</i>	The  bitmap is displayed
<i>cbsCustom</i>	The icon defined by the <i>ButtonGlyph</i> property.

Data Type: TwwComboButtonStyle

Valid Values: cbsEllipsis, cbsDownArrow, cbsCustom

ButtonWidth

This property defines the width of the icon for the control. You may wish to set this property if your custom bitmap assigned to the ButtonGlyph property is larger than the default button width. This property defaults to 0, which indicates to the control to compute the button width based on the system settings..

Data Type: Integer

CalendarAttributes

This property defines the attributes of the pop-up month calendar. See the TwwDBMonthCalendar for further information on the following properties.

Alignment	This property determines the how the drop-down calendar is aligned with the date time picker. This property defaults to <i>wwdtaLeft</i> , which means that the drop-down calendar is aligned with the left-hand border of the date time picker. Valid Values: <i>wwdtaLeft</i> , <i>wwdtaRight</i> , <i>wwdtaCenter</i>
Colors	See the TwwDBMonthCalendar’s <i>CalColors</i> property.
FirstDayOfWeek	See the TwwDBMonthCalendar’s <i>FirstDayOfWeek</i> property.
Font	Determines the default font used by the drop-down calendar.
Options	See the TwwDBMonthCalendar’s <i>Options</i> property.
PopupYearOptions	See the TwwDBMonthCalendar’s <i>PopupYearOptions</i> property.

DataField

This property defines the name of the field you want to bind the DateTimePicker to. The default value is blank (unbound).

Data Type: String (FieldName)

DataSource

This property defines the name of the TDataSource you want to bind the DateTimePicker to. The default value is blank (unbound).

Data Type: TDataSource

Date

This property defines the date the DateTimePicker initially displays. This property is ignored if the component is bound to a database field, as the date will then originate from the value of the database field.

Data Type: TDateTime

DateFormat

This property defines whether the data is displayed using the system short date format, or the system long date format. If the *DisplayFormat* is assigned, then this property is ignored. If the control is bound to a datafield with its *TField.DisplayFormat* property assigned, then this property is also ignored.

Data Type: TDateTimeFormat

Valid Values: dfShort, dfLong

DisableThemes

If your project has enabled XP themes but you do not wish for this control to be theme-enabled, then set this property to *False*.

DisplayFormat

This property defines the format the control uses to both display and edit the date/time. See Delphi's TDateTimeField *DisplayFormat* property for details on specifying a display format.

Data Type: String

Valid Values: Blank or valid Delphi format string

Example: The following are some examples of setting this property

<u>DisplayFormat</u>	<u>Sample Displayed Text</u>
mmm dd, yyyy	July 01, 1998
hh:mm:ss AMPM	08:23:23 AM
mm/dd/yyyy	07/01/1998

Epoch

This property defines the epoch date to determine how 2 digit years resolve to 4 digit years. This property defaults to the *TwwIntl.DefaultEpochYear* property (1950), which will translate 2 digit years less than 50 to 20xx', and years greater than 50 to 19xx.

Date Type: Integer

Valid Values: Valid year greater than 1900

Frame

See the topic “Key properties and events for custom framing” in chapter 4 for more information on this property.

Interval.MinutesInterval

When the user enters the up/down arrows when the cursor is placed into the minutes, the control adds or subtracts this amount of minutes from the current minute value.

Interval.RoundMinutes

If you wish for the minutes to round up/down to the nearest multiple of `MinutesInterval`, then set `RoundMinutes` to true. For instance, if `MinutesInterval` is set to 5, and the current time is 1:03:00, then if you increment the minutes the resulting value will be 1:05:00 (rounded to a multiple of 5).

MaxDate

This property defines the maximum allowable date that the date time picker will allow the end-user to select. The default is blank means that the upper range of the date is not restricted.

Data Type: TDateTime

MinDate

This property defines the minimum allowable date that the date time picker will allow the end-user to select. The default is blank which means that the lower range of the date is not restricted. **Note:** The date time picker does not support dates less than the year 1900.

Data Type: TDateTime

ShowButton

Set this property to False, to hide the control’s drop-down button. The button is automatically hidden if the control is only displaying/editing the time.

Data Type: Boolean

Time

This property defines the internal time that the month calendar stores. This is never displayed to the end-user, but is used internally when updating a database field

Data Type: TDateTime

UnboundDataType

If the control is unbound, then this property determines the control’s data type. To edit an unbound date time picker as a time, set this property to `wwDTEdtTime`. Similarly set this property to `wwDTEdtDate` to edit the control as a date.

If the control is bound (datasource and datafield properties assigned), then the control determines the field type from the TField information and this property is ignored

Note: For more detailed control over the formatting, see the *DisplayFormat* property for this control, as InfoPower will automatically edit and view based on this format.

Data Type: TwwDTEditDataType

Valid Values: wwDTEdtDateTime, wwDTEdtDate, wwDTEdtTime

Required property assignments

None

Added Events

OnCalcBoldDay

This event allows you to calculate which dates should be displayed in bold in the drop-down calendar. This event is only called if *CalendarAttributes | Options | mdoDayState* is True. See the *TwwDBMonthCalendar*'s *Options* property for detailed documentation on this event.

TwwDBEdit



The TwwDBEdit component is an InfoPower enhanced data aware edit component. Some of its enhancements include the following:

- ◆ InfoPower gives Delphi programmers the power to define a data entry template, or mask, for the values that can be entered into a field displayed on the screen. Please reference chapter 4, *Selecting a Picture Mask* for details on using Picture Masks.
- ◆ Automatically detects date fields and allows the end-user to automatically fill in the current date by entering the spacebar a few times.
- ◆ Full integration with InfoPower's grid, which allows you to embed any InfoPower edit control directly into the grid.
- ◆ Support for custom framing and transparency
- ◆ InfoPower adds new events for hot-tracking the mouse.



Figure 5.6 - TwwDBEdit component

Ancestor

TCustomMaskEdit

└─TwwCustomMaskEdit

└─TwwDBCUSTOMEdit

Required supporting components

None.

Inherited properties

The TwwDBEdit has the following standard Delphi properties:

AutoSelect, AutoSize, BorderStyle, CharCase, Color, Ctl3D, DataField, DataSource, DragCursor, DragMode, Enabled, Font, MaxLength, ParentColor, ParentCtl3D, ParentFont, ParentShowHint, PasswordChar, PopupMenu, ReadOnly, ShowHint, TabOrder, TabStop, Visible.

See the Delphi documentation and help files for more information on these properties.

Added properties

AutoFillDate

When True, the user can automatically fill in a TDateField with the current date by pressing the spacebar a few times. The user's cursor position must be at the end of the text for *AutoFillDate* to work. This property is ignored when the database field is assigned a Delphi edit mask or an InfoPower picture mask with *AutoFill*.

Data Type: Boolean

DisableThemes

If your project has enabled XP themes but you do not wish for this control to be theme-enabled, then set this property to *False*.

Frame

See the topic "Key properties and events for custom framing" in chapter 4 for more information on this property.

Picture

Picture mask specification. Please reference chapter 4, *Selecting a Picture Mask* for details on this property.

ShowVertScrollBar

When True, the control will display vertical scrollbars in the edit box. Use this with the *WordWrap* property set to True.

When using this property, you may also wish to set the *AutoSize* property to False and resize the control.

Data Type: Boolean

UnboundAlignment

When this component is used without a datasource and datafield, this property determines how the component will align the text within the control when it does not have the focus.

Data Type: TAlignment

UnboundDataType

When this component is used without a datasource and datafield, this property determines how the component will auto-fill when the space key is entered by the end-user. See the Delphi documentation on *date/time formatting* to manipulate the format of the filled text.

AutoFillDate must be set to True in order for this property to be used.

Data Type: TwwEditDataType

Valid Values:

<i>wwDefault</i>	No auto-filling of date or time
<i>wwEdtDate</i>	Auto fill using current date
<i>wwEdtTime</i>	Auto-fill using current time
<i>wwEdtDateTime</i>	Auto-fill using current date and time

UsePictureMask

When True, picture masks are used by the InfoPower controls during editing.

When False, picture masks are not used during editing. In either case, True or False, picture masks are still used to verify the validity of the fields in the record before the record is posted, and when the user moves focus away from the component.

Data Type: Boolean

WantReturns

When True, the editor will accept carriage returns.

Data Type: Boolean

WordWrap

When True, the editor supports word wrapping. You will probably want to set AutoSize to False when enabling this property.

Data Type: Boolean

Modified properties

None

Required property assignments

None

Added Events

OnCheckValue

This event allows you to perform some custom action based on any change to the edit component's text. For instance, you may want to put the edit control in yellow when it does not satisfy the picture mask requirements. Please reference chapter 4, *Selecting a Picture Mask* for details and examples on this event.

Note: This event is only fired if you have a picture mask defined for the field.

OnMouseEnter

Occurs when the mouse cursor passes from outside the control to inside the control. Use this along with the *OnMouseLeave* event for hot-tracking effects.

OnMouseLeave

Occurs when the mouse cursor passes from inside the control to outside the control.

Added Methods

UpdateRecord

This method flushes the currently edited value to the dataset record buffer. You may wish to call this method if you wish for other controls that are tied to this field to immediately reflect the new value. You usually will not need to explicitly call this method, since this method is automatically called for you when the user exits the control.

Be aware that once you call this method the <Escape> key will no longer restore the original contents of the field. You can still cancel the record's changes by calling the table's *Cancel* method.

TwwDBGrid



The TwwDBGrid component is one of the most powerful components in the InfoPower library, greatly expanding upon the capabilities of Delphi's built-in TDBGrid component.

Account #	Name		Interests/Hobbies	Photo	Billing Information		
	First	Last			Shipping Address	Payment Method	Balance
1023495	Jennifers	Ardeny	Enjoys horseback riding and paints.		Street: 100 Cranberry... City: Abilene State: MA Zip: 02181	 	\$33.00
2094056	Arthur	Jones	♥'s chess.		Street: 10 Hunnewell... City: Lowell State: CA Zip: 94024	 	\$22.00
1209395	Debra	Parker	Owens his company.		Street: 74 South St City: Astoria State: CA Zip: 98765	 	\$44.00
3094095	Dave	Sawyer	Retired. Enjoys travel and bungee jumping.		Street: 101 Oakland St City: Los Altos State: CA Zip: 94022	 	\$21.00

Figure 5.7 - An example of the visual portion of an InfoPower TwwDBGrid component in action.

What is new in InfoPower 4000

InfoPower has significantly enhanced its masterpiece grid with significant new functionality. Some of the new features are described below.

Allow grouping of related data in grid

InfoPower 4000 allows you to group common data in the grid, by displaying only the text for the first instance, as well as removing the lines in between. See the *Company* field below.

Company	OrderNo	SaleDate	ShipDate	EmpNo	S
Shangri-La Sports Center	1101	04/07/1989	04/07/1989	37	
Divers of Corfu, Inc.	1298	09/01/1995	09/01/1995	11	
	1098	14/06/1989	14/06/1989	118	
	1046	12/11/1988	13/11/1988	24	
	1146	13/02/1994	13/02/1994	94	
	1198	14/09/1994	14/09/1994	65	
Kirk Enterprises	1096	09/06/1989	09/06/1989	34	
	1196	08/09/1994	08/09/1994	9	
George Bean & Co.	1069	05/04/1989	06/04/1989	136	
Professional Divers, Ltd.	1201	04/10/1994	04/10/1994	114	
Divers of Blue-green	1260	10/12/1994	10/12/1994	107	
	1078	01/05/1989	02/05/1989	39	
Gold Coast Supply	1103	13/07/1992	13/07/1992	94	
	1081	07/05/1989	08/05/1989	109	
	1063	13/03/1989	14/03/1989	24	
San Pablo Dive Center	1175	22/07/1994	22/07/1994	110	
Underwater Sports Co.	1250	24/11/1994	24/11/1994	61	

Edit aggregate fields or detail information using drop-down panels

InfoPower's superb grid can now display and edit aggregate or detail information from a drop-down panel. Previously you were restricted to using a drop-down grid or drop-down inspector from its clickable expand button. By allowing a panel, your grid's capabilities and display options are dramatically improved. See the how-to topics for details on this topic. See the `TwwExpandButton` for information on embedding a drop-down panel in the grid.

Improved custom control integration and flexibility

Supports custom control (i.e. `richedit`) to grow larger when it has the focus. This allows for increased display and editing conveniences.

`IstClass` buttons can now seamlessly be integrated into the InfoPower grid, giving your grid a clickable component for each record. In addition the button colors can be dynamically computed allowing the color to be different based on the record information.

Auto-sizing of column

Dbl-clicking the sizing line for a grid can automatically grow or shrink the column's width, based on the widest displayed text in the column

Improved flexibility of custom painting

InfoPower 4000 adds new painting events : *OnBeforeDrawCell* and *OnAfterDrawCell*, and new methods *GetPriorRecordText* and *GetNextRecordText* which you can access in these events to allow you to have the painting logic be based on the next and previous record's data.

Ditto Capability

Allow the user to conveniently copy the previous record's value into the cell. See the *DittoAttributes* property, and the *OnDitto* event.

In addition to the functionality of TDBGrid, InfoPower's TwwDBGrid provides you with:

Custom control integration and flexibility

InfoPower allows you to embed a wide variety of controls in the grid. You can even embed non-InfoPower controls, such as the *TDBImage*. The grid also allows each custom control to do their own painting in the grid so that even graphics, richedits, etc. will be displayed for every row in the grid without any code on your part.

The data inspector can also be embedded in the grid, giving a multi-row record display. The new checkbox and radiogroup controls can be embedded, which significantly improves the versatility of the grid. When embedding non-text controls in the grid, you should set the *Control Always Paints* checkbox in the Select Fields Dialog. This will allow the control to be painted graphically, instead of as simple text.

True Master / Detail Relationships displayed from a single grid

InfoPower brings you a new paradigm to display and *edit* your master/detail relationships. Detail tables can be initially hidden, and then expanded into full view when the end-user expands a expand/collapse button in the parent grid. Each child-grid is fully customizable as in the parent grid, and the control preserves the liveness of each expanded detail grid.

☐	Cust No	Country	Last Invoice Date	Company		
☐	CN 1354	British West Indies	1/30/92 2:00:56 AM	Cayman Divers World Un		
☐	CN 1356	US Virgin Islands	3/20/92 9:35:40 AM	Tom Sawyer Diving Cent		
☐	CN 1380	US	11/8/94 11:22:08 PM	Blue Jack Aqua Center		
☐	CN 1384	US Virgin Islands	2/1/95 6:45:23 PM	VIP Divers Club		
☐	CN 1510	US	11/9/94 1:22:22 AM	Ocean Paradise		
▶ ☐	CN 1513	Columbia	7/18/94 5:17:01 PM	Fantastique Aquatica		
	Items	Order No	Date	Ship To Contact	Cust No.	Items Total
			Sale	Ship		
▶ ☐	#1086	5/18/89	5/19/89		CN 1513	\$14,049.95
		Item No	Part No	Qty	Discount	
		1	2367	1	0	
		2	11518	7	0	
☐	#1090	5/25/89	5/26/89		CN 1513	\$8,507.00
☐	#1109	11/13/92	11/13/92		CN 1513	\$203.00
☐	#1156	5/9/94	5/9/94		CN 1513	\$12,367.00
						\$90,143.40
☐	CN 1551	Canada	7/7/90 4:20:58 AM	Marmot Divers Club		
☐	CN 1560	US	4/9/94 8:14:52 AM	The Depth Charge		

To allow a grid to display an expand button for a detail grid, you must attach a `TwwExpandButton` to the column in the grid. See the `TwwExpandButton` component for more information. See also the how-to topic “How to embed a grid within the grid”.

You can also embed a drop-down inspector or `TPanel` with the expand button to further increase the flexibility of the grid’s representation. See the `TwwExpandButton` for information on embedding a drop-down panel within the grid.

Support for exporting from the grid to various formats.

InfoPower supports exporting to various formats to aid your end-users in extracting their displayed data to be used with other applications. In addition you can copy the selected records to the clipboard. See the following for an example of the generated html.

<http://www.woll2woll.com/infopower/exportexample.html>

See the `ExportOptions` property for more information on exporting data from the grid.

Background texture tiling

InfoPower allows your applications to further impress by adding support for background texture tiling. The component takes care of blending your tile with the color of the grid region, giving you a truly impressive and professional display. See the `TwwDataInspector.PaintOptions` property for details on enabling the background texture tiling.

Expand/Collapse buttons for composite calculated fields

Account #	Name	ShippingAddress	Interests/Hobbies								
1023495	Jennifers Ardeny	100 Cranberry St., Abilene, MA 02181	Enjoys								
2094056	Arthur Jones	10 Hunnewell St, Lowell, CA 94024									
1209395	Debra Parker	74 South St, Astoria, CA 98765	Owns his <i>BMW</i>								
		<table border="1"> <tr> <td>Street</td> <td>74 South St</td> </tr> <tr> <td>City</td> <td>Astoria</td> </tr> <tr> <td>State</td> <td>CA</td> </tr> <tr> <td>Zip</td> <td>98765</td> </tr> </table>	Street	74 South St	City	Astoria	State	CA	Zip	98765	
Street	74 South St										
City	Astoria										
State	CA										
Zip	98765										
3094095	Dave Sawyer	101 Oakland St, Los Altos, CA 94022	Retired. Enjoys								
1024034	Cindy White	1 Wentworth Dr, Los Altos, CA 94022	Enjoys fishing.								

Use expand/collapse buttons to allow the user to edit a composite field. You can display a calculated field such as full name (composed of first name + last name), and then the user can expand the composite calculated field to edit the individual portions.

Grid header enhancements

The titles in the grid can be displayed and managed hierarchically. This allows you to group related fields together. Also for header dragging operations, the grid more clearly indicates the new placement of the column.

Account #	Name		Interests/Hobbies	Photo	Billing Information		
	First	Last			Shipping Address	Payment Method	Balance
1209395	Jennifers	Ardeny	Enjoys horseback riding and paints.		100 Cranberry St., Abilene, MA 02181	MasterCard	\$33.00
2094056	Arthur	Jones	loves chess.		10 Hunnewell St, Lowell, CA 94024	VISA	\$22.00

If you wish to enable grouped headers, you must uncheck the “Store Display Settings in TFields” checkbox in the *Select Fields Dialog*. Then enter the group name for the field. Be sure to consecutively couple the fields that should be grouped.

Flicker-free display

The grid improves both its painting performance and visual effect with its new flicker-free display. The new flicker-free display is automatically enabled without any new property settings.

Fixed column enhancements

InfoPower allows your end-users to edit and resize fixed columns. To enable this new functionality, set the *Options | dgFixedEditable* and *Options | dgFixedResizable* to True.

Clickable URL link support

Define columns as URL links (i.e. Email addresses). The grid will automatically handle its display and the opening of the link. To enable URL-links for a column you will need to define a column as a custom control (using the Select Fields Dialog). The format in the database is <URL Display String>#<URL Link Address>. Alternatively you can omit the <URL Display String># and the grid will display the raw address instead of the display string.

Native Alternating color

New property to automatically alternate the colors of the rows in the grid. This provides a pleasing look to the end-user and helps differentiate one record from another. To enable alternating colors, see the `TwwDataInspector.PaintOptions` property.

Proportional column sizing

The grid can now automatically size all the columns to fit perfectly in the grid's client area. If the grid is resized, all the columns still fit perfectly. Any trailing column space after the last column is removed. To enable proportional column sizing, you must set the grid's `UseTFields` property to `False` and set `Options | dgProportionalColResize` to `True`.

End-user row sizing

User's can enlarge the sizes of the rows by dragging the horizontal line in the indicator column. To enable this capability, set the `Options | dgRowResize`. When resizing a row, all the data rows in the grid will use the new size.

Line color customizations

You can override the default line colors by setting the `LineColors` property. See also the `LineStyle` property.

Support editing with rowselect

Previously with `rowselect`, you could not edit within the grid. InfoPower adds a new option to highlight the active row, and still allow editing. See the `PaintOptions.ActiveRecordColor` property for more information.

InfoPower's TwwDBGrid also provides you with the following abilities:

- ◆ **Cell-level hints when the cell's text does not fit in the cell :** InfoPower automatically displays the full text of a cell as a tool tip when the mouse is moved over the cell. The tool tips also support memo fields and multiple lines.
- ◆ **Footer cell support -** Developer can embed footer cells at the bottom of the grid to display summaries of column information.
- ◆ **Saving and loading of the user's runtime settings:** The grid now can automatically stream its current display settings to and from an INI file or the

system registry during program execution. This allows the end-user to conveniently order and size the columns, and then save their settings for the next time they run your application.

- ◆ **DateTime picker support** : Embed DateTimePicker controls directly in the grid. InfoPower's grid will even detect date or time fields and automatically use the appropriate control.
- ◆ **ImageList support** : Display bitmaps from image lists in both the column headers and the data cells.
- ◆ **Embed a wider variety of powerful controls into the grid**: Display a field as a normal cell edit box, checkbox, combo box, spinedit, date time picker, bitmap, lookup combo box, or your own custom edit box.
- ◆ **Display the text of RTF fields in the grid**, and also display a customizable word processor window where your end-users can view and/or edit the contents of the rich edit.
- ◆ **Scaleable row heights** : Scale cells to double or triple the height and word-wrap text in resulting cells.
- ◆ **Clickable column headers** as each title caption in the grid can depress like a button.
- ◆ **Use InfoPower's powerful picture edit-masks** when editing cells in the grid.
- ◆ **Embed a TSpeedButton into the Indicator column**, allowing smooth integration with the RecordView component.
- ◆ **Calculated fields can now be edited in the grid**. As a result you will be able to edit calculated linked fields or lookup fields in a grid with only a few lines of code.
- ◆ **Display memo fields within the grid** and also display a customizable pop-up memo-editing window where your end-users can view and/or edit the contents of a memo field, depending on how you set the properties.
- ◆ **Complete control over how the titles are displayed**: Set the alignment of column headings to left, center or right justified. You can even separately control each column's heading attributes (font color, background color, alignment), as well as display multi-line headings, and icons from ImageLists.
- ◆ **Change the background color and font color** displayed within individual cells and entire rows.
- ◆ **Define fixed, non-scrollable columns** in the left-hand side of the grid.
- ◆ **Built-in support for selecting multiple records**: Select contiguous records using shift-select, and auto-unselect the selected records when you just click on a record.

- ◆ **Smart key mapping** support if you want carriage returns automatically converted to a tab.
- ◆ **Can hide horizontal or vertical scrollbars.**
- ◆ **And much more!**

Ancestor

TCustomGrid

└─ TwwCustomDBGrid

Required supporting components

TDataSource.

Added properties

CalcCellCol

Runtime property used specifically by the *OnCalcCellColors* event. Reference this property from the *OnCalcCellColors* event if you wish to have your cell painting logic be dependent upon the column number being painted.

CalcCellRow

Runtime property used specifically by the *OnCalcCellColors* event. Reference this property from the *OnCalcCellColors* event if you wish to have your cell painting logic be dependent upon the row number being painted.

Example: The following code will paint the entire active row with the color, clHighlight. Normally setting dgRowSelect to True can do this, but the side effect of dgRowSelect is that you can no longer edit the row. The code below allows you to preserve the editing capabilities but still paint the entire row.

```

procedure TBitmapForm.InvoiceGridCalcCellColors (
  Sender: TObject; Field: TField;
  State: TGridDrawState; highlight: Boolean;
  AFont: TFont; ABrush: TBrush);
begin
  with (Sender as TwwDBGrid) do
    if CalcCellRow = GetActiveRow then
      ABrush.Color:= clHighlight;
end;

```

ColWidthsPixels

Runtime property to allow precise pixel control of a column's width during runtime. The array index is the column number you wish to manipulate or evaluate. Returns the width of a column in pixels.

Data Type: Array of Integer values

ControlInfoInDataset

Set this property to *False* if you wish for the grid to store the information about the embedded controls into the Grid. You may wish to set this property to *False* if you want the grid to have no dependency upon the embedded control information stored in the dataset. By default this property is *False*, which means that information about the grid's embedded controls is stored in the related *TDataSet*.

Note: Normally you will want to leave this property as *True*. Set this property to *False* if you have more than one grid, attached to the same dataset, but on different forms

ControlType

This property is equivalent to the *ControlType* property (See *TwWTable ControlType*). InfoPower stores the control information into this property if the *ControlInfoInDataSet* property is *False*. Otherwise this property is not used.

DataSource

Lists only TDataSource components.

DisableThemes

Set this property to *False* to prevent themes from being used in the painting in the data area of the grid.

DisableThemesInTitle

Set this property to *False* to prevent themes from painting in the title area of the grid. Themes will still be used within the data portion of the grid (unless *DisableThemes* is *False*).

DittoAttributes

InfoPower supports a mechanism to allow the user to conveniently copy the previous or next record with a single keystroke sequence.

DittoDirection

Set this property to determine if the ditto functionality should copy data from the next or previous record.

wwDittoPrior – Copy previous record's data when ditto shortcut is pressed. If the active record is the top displayed record in the grid, then no action is performed.

wwDittoNext - Copy next record's data when ditto shortcut is pressed. If the active record is the bottom displayed record in the grid, then no action is performed.

wwDittoPriorOrNext – Copy prior record's data when ditto shortcut is pressed. If the active record is the top displayed record in the grid, then the next record is copied.

Data Type: TwWDittoDirection

ShortCutDittoField

Set this property to assign a shortcut key to copy a single field value of the dittoed record.

Data Type: TShortCut

ShortCutDittoRecord

Set this property to assign a shortcut key to copy the field values of the dittoed record. Assign the *Options* property if you wish to configure which fields are dittoed. Use the *OnDitto* event if you wish to further customize which fields are copied.

Data Type: TShortCut

Options

This property customizes the ditto functionality. The following flags are available:

<code>wwdoSkipBlobFields</code>	Set to true to disable the copying of blob fields
<code>wwdoSkipReadOnlyFields</code>	Set to true to disable the copying of readonly fields
<code>wwdoSkipHiddenFields</code>	Set to true to disable the copying of fields not visible in the grid.

Data Type: TwwDittoOptions

DragVertOffset

InfoPower allows the user to drag a column to another position, and animates the header column being dragged. This property determines the number of pixels to offset the dragged header.

Data Type: Integer

Valid Values: Any Value greater than 0

EditCalculated

When set to true it allows you to edit a calculated or lookup field. As a result you will be able to edit calculated linked fields or lookup fields in a grid with only a single line of code.

Tabstops will automatically be created on these columns. See example on how to edit linked and lookupfields in the Grid.

Data Type: Boolean

EditControlOptions

This property defines specific settings for controls embedded in an InfoPower grid

Data Type: TSet()

Valid Values: `ecoCheckboxSingleClick`, `ecoSearchOwnerForm` (described below)

ecoCheckboxSingleClick When True, the end-user need only single-click a checkbox cell in the grid to toggle it. When False, a double-click is required.

ecoSearchOwnerForm When True, the grid searches for embedded controls on the grid's owner form. When False, the grid will search the grid's parent form. Usually you will want to set this to True.

<i>ecoDisableCustomControls</i>	If True, then the grid will not use the assigned custom controls during editing.
<i>ecoDisableDateTimePicker</i>	If False, then the grid will disable the automatic creation and use of the <i>TwwDBDateTimePicker</i> control to edit dates or time fields.
<i>EcoDisableEditorIfReadOnly</i>	If True, then the grid will disable the inplace editor if the field is not editable.

ExportOptions

This property defines specific settings for exporting data from the grid to a series of different file types or to the clipboard for other applications to use. Choose from HTML, formatted text, tabbed delimited text, comma delimited text (this is a common spreadsheet format known as .CSV), or the Excel SYLK (.SLK) format. Or by setting *ExportOptions | Options | esoClipboard* to True, the data will be saved to the clipboard in the chosen *ExportOptions | ExportType* format. If *dgMultiSelect* is enabled in the grid and an enduser has selected some records, then you have the choice of exporting only the selected records or the current contents of the filtered or nonfiltered dataset.

Data Type: *TwwExportOptions*, which consists of the following sub-properties.

Delimiter

Defaults to comma. *Delimiter* used to separate the fields from each other when the *ExportType* is set to *wwgetTxt*. To export the data as tabbed text, set the *Delimiter* property to #9 at runtime. To export in a CSV format, set the *Delimiter* property to ','. To export records as formatted text (or spaced output), set the *Delimiter* property to ''.

Data Type: String

ExportType

This property determines the actual format that the data is stored in. The default is *wwgetTxt*, which means that the data will be saved in text only format with its fields separated by the character defined by the *Delimiter* property.

Data Type: *TwwGridExportType*

Valid Values: *wwgetTxt*, *wwgetHTML*, *wwgetSYLK*, *wwgetXML* (described below)

wwgetTxt When *ExportType* is set to *wwgetTxt*, records will be saved to the specified *FileName* in a Text Format, or if *esoClipboard* is in the *ExportOptions | Options* property then the record data will be saved to the clipboard in the *CF_Text* clipboard format. The *delimiter* settings will determine the actual format of the exported text. See the *Delimiter* property for more details.

wwgetHTML This format is one of the most flexible and powerful export formats. Depending on the *ExportOptions | Options* settings, it is possible to preserve the colors, fonts, group headings, column widths, footers, and controls that are in the grid. Save to an HTML file and use an internet browser to view the resulting table, or save to the clipboard and paste to Microsoft Word or Excel.

- wwgetSYLK* Microsoft Excel supports the spreadsheet format .SLK, which can retain the current column widths and group headings, fonts that are set at the time of export. Setting *ExportOptions | ExportType* to *wwgetSYLK* will cause data to be saved in that format.
- wwgetXML* Not Implemented Yet. Provided for future XML exporting.

FileName

Defaults to blank, which indicates that the exported data will be saved in a .txt file, which will get its name from the Applications executable name. You should set the filename's extension based on the *ExportOptions | ExportType* property setting.

For *wwgetTxt*, you may wish to set the extension of the filename to .TXT or .CSV. For *wwgetHTML*, set the filename extension to .HTML. For *wwgetSYLK*, set the filename extension name to .SLK. For *wwgetXML*, set the filename extension to .XML.

Data Type: String

HTMLBorderWidth

This property defaults to 1. This means that when the *ExportOptions | ExportType* property is set to *wwgetHTML*, that the resulting table generated from the data will be stored in an HTML table with a border width of 1.

Data Type: Integer

Options

This property defines the display of the resulting exported table.

Data Type: TSet()

Valid Values: *esoShowHeader*, *esoShowFooter*, *esoDynamicColors*, *esoShowTitle*, *esoDbQuoteFields*, *esoSaveSelectedOnly*, *esoAddControls*, *esoBestColFit*, *esoShowRecordNo*, *esoEmbedURL*, *esoShowAlternating*, *esoTransparentGrid*, *esoClipboard* (described below)

- esoShowHeader* When True, the exported table will contain the grid's titles.
- esoShowFooter* When True, if the grid is displaying footer cell information, then the resulting exported table for certain export types will also display the footer cell information.
- esoDynamicColors* When True, colors from the *TwwDBGrid*'s *OnCalcCellColors* event will display in the resulting exported table. Currently only *wwgetHTML* supports this.
- esoShowTitle* When True, the string defined by the *TitleName* property will be exported as the Title of the exported file.
- esoDbQuoteFields* When True, fields will be quoted when exporting text using *wwgetTxt*.
- esoSaveSelectedOnly* When True, only the multiselected records in the grid will be saved. *dgMultiSelect* needs to be enabled in the grid.

<i>esoAddControls</i>	When the <i>ExportType</i> is set to <i>wwgetHTML</i> , then controls will be created in the generated HTML Table. As this can create a lot of controls, you should use this property with caution.
<i>esoBestColFit</i>	When True, the exported table will try to find the best fit for the columns based on the data in the resulting table. This currently only applies to tables exported to SYLK or HTML.
<i>esoShowRecordNo</i>	When True, the rows in the resultant table will be numbered from 1 to record count.
<i>esoEmbedURL</i>	When the <i>ExportType</i> is set to <i>wwgetHTML</i> , then URL field types will be displayed as clickable URLs in the resultant HTML table. This is a powerful feature.
<i>esoShowAlternating</i>	When the <i>ExportType</i> is set to <i>wwgetHTML</i> , then the resultant table will display every other row in the alternating color defined by the Grid's <i>PaintOptions AlternatingRowColor</i> property.
<i>esoTransparentGrid</i>	When the <i>ExportType</i> is set to <i>wwgetHTML</i> , then the resultant table be displayed transparently.
<i>esoClipboard</i>	When True, the exported data will be saved to the clipboard in the format defined by the <i>ExportType</i> property and the <i>Delimiter</i> setting. This is particularly useful with the powerful HTML format for example, as you can use the <i>Save</i> method to save to the clipboard and paste to Microsoft Word or Microsoft Excel.

OutputWidthinTwips

This property defaults to 0. This property applies to HTML exporting only. Set this property to the desired HTML Table width in twips or inches (1440 Twips=1 Inch). A value of zero will mean the columns of the resulting table are not proportionally sized to achieve the desired table width setting based on *OutputWidthinTwips*.

Data Type: Integer

Note: If there are too many columns/fields being exported, then it may not be possible to squeeze the table to this desired width.

TitleName

This property defines the *TitleName* of the resulting table/file.

Data Type: String

Example 1 (Saving to HTML File): The following example demonstrates how you would save the grids data to an HTML File.

1. Set the Grid's *Filename* property to xxxxx.html. Where xxxxx is the name of the HTML file you wish to generate.
2. Set the *ExportOptions | ExportType* to *wwgetHTML*
3. Call the *Save* method and the generated file will be created. If you wish to save this to the clipboard so that you can paste the resulting HTML into Microsoft

Word or Excel 2000, then set the *ExportOptions* | *Options* | *esoClipboard* to True.

4. If you want to programmatically display this with the default browser. Add `shellapi` to your form's uses clause and call it like:

```
ShellExecute(Handle, 'OPEN', PChar(wwDBGGrid1.exportoptions.FileName),  
nil, nil, sw_shownormal);
```

Example 2 (Saving selected data to Excel using the clipboard): The following example demonstrates how you would save selected data to Excel using the clipboard.

1. Set the *ExportOptions* | *ExportType* to `wwgetSYLK`. You could optionally also set it to `wwgetTxt` for exporting just the data, or `wwgetHTML` (for special formatting, coloring, and for Excel 2000).
2. Set the *ExportOptions* | *Options* | *esoClipboard* to True. If you wish to export to a file set this to False and set the filename to the format `xxxxx.slk`.
3. Call the save method.
4. Open Excel and then select `Edit | Paste` or `Edit | Paste Special`.

FixedCols

This property defines the number of columns, from the left-hand side of the grid (not including the record indicator column), that should be non-scrollable. If you specify a number larger than the number of columns actually visible in the grid, the value is defaulted to 0. The default value of 0 defines the current record indicator column as non-scrolling.

Data Type: Integer

Valid Values: Zero to the positive number of columns that are displayed in the *visible* portion of the grid.

FooterCellColor

This property defines the color of the cells in the footer. The default is `clBtnFace`.

Data Type: TColor

FooterColor

This property defines the footer background color. The default is `clBtnFace`.

Data Type: TColor

GroupFieldName

Assign this property so that the grid paints common data as a group. You must ensure that the dataset is already sorted by the common data. The grid will automatically manage the painting of the grid so that the horizontal lines between the common data are removed, as well as the repeating data values are only shown the first time. See the `griddatagroup.pas` file within the `maindemo` program for an example.

Data Type: String

FooterHeight

This property defines the height of the footer section in the grid. The default is 0, which indicates to the grid to compute the height based on the grid's row height. To enable the display of the footer in the grid, set the *Options | dgShowFooter* to True.

Data Type: Integer

HideAllLines

Set this property to true to disable the other lines drawn in the grid that are not controlled by *Options | dgColLines* and *Options | dgRowLines*.

ImageList

This property relates a TImageList to the grid's data cells. This ImageList is used for any column whose ControlType is set to ImageIndex (See *Using the Select Fields Dialog Box | Edit Control* in chapter 4). See also the "How-to" section for this component under "Display an image from an ImageList".

Data Type: TImageList

IndicatorButton

This property creates a TSpeedButton on the upper left corner of the Grid. You can attach any code to this button. It has all the events and properties of a normal TSpeedButton. Use this as a convenient way for invoking the TwwRecordViewDialog.

Data Type: TSpeedButton

IndicatorColor (Obsolete property – replaced by IndicatorIconColor)

This property defines the color used to draw the record indicator graphic. The default value is icBlack. (Note: Only black and yellow are allowed because each color requires three separate, color-specific bitmaps.)

Data Type: Constant

Valid Values: icBlack or icYellow

IndicatorIconColor

This property defines the color used to draw the record indicator graphic. The default is clBlack.

Data Type: TColor

IniAttributes

This property defines specific settings for saving and storing the column positions, sizes, display names and readonly properties into a system .INI file or as a registry entry. When *IniAttributes.Enabled* is True, the grid will automatically save the grid's column information when the grid is freed. It will also automatically load the grid's column information from the registry or .INI file when the grid's window handle is created. If you would like to load and save the grid's settings with code, you can use the LoadFromIniFile and *SaveToIniFile* methods.

Data Type: TwwIniAttributes, which consists of the following sub-properties.

CheckNewFields

When set to true the grid will automatically include new fields added to the grid's selected property). This property is useful when you add new fields to the grid and wish those new fields to be automatically included into the grid's display. When this property is false, the grid would use the exact setting stored in the INIFile and thus would not include any new fields or registry.

Data Type: String

Delimiter

Delimiter used to separate the column/field information that is stored in the .ini file or registry.

Data Type: String

Enabled

Defaults to False, which means that the column information is not automatically saved. Set to True to save the column/field information into the registry or an .ini file.

Data Type: Boolean

FileName

Defaults to blank or to the value of the TwwIntl *IniFileName* setting when *Connected* is true. When *SaveToRegistry* is False, then the settings are saved into an .ini file. Of special note, unless you specify the full pathname for the .ini file name, then the file will be placed in the Windows directory. A blank filename will result in an .ini file in the Windows directory named after your application name. When specifying a full path, if the directory does not exist then no settings will be saved or loaded.

When *SaveToRegistry* is set to True, then this property **must** include the registry path with the key name. For example: Software\Company\UserSettings\MyApp will be store the grid settings in the registry in

HKEY_CURRENT_USER\Software\Company\UserSettings\MyApp\SectionName.

Data Type: String

SaveToRegistry

When True and *IniAttributes* is *Enabled*, then the Grid's column/field information is stored into the system registry. *FileName* must be set. Defaults to False, which means that the information is stored in an .ini file.

Data Type: Boolean

SectionName

This property will default to blank. This means that the grid's column/field information will be stored in the registry or .ini file in a section named FormName+GridName.

Data Type: String

Example 1 (Saving to Windows .ini File): The following example demonstrates how you would save the column information into an .ini file in the user's windows directory.

1. Set the Grid's *IniAttributes.Enabled* property to True.
2. If you wish the .ini file to have a different name than your application name, then set *FileName* to the name of the .ini file that you wish to set. (For example: MyProject.ini)
3. You can override the name of the section that the grid's column/field information is stored in by setting the *SectionName* property. The default is blank which means that a unique section name will be generated based on the name of your form and the name of your grid.

Example 2 (Saving to the Windows Registry): The following example demonstrates how you would save the column information into the user's windows registry for your application.

1. Set the Grid's *IniAttributes.Enabled* property to True.
2. Set the Grid's *IniAttributes.SaveToRegistry* property to True.
3. Now, set the *FileName* to point to the Path where the grid's column/field information should be stored in the registry. So for example if you wish to store the information in HKEY_CURRENT_USER\Software\YourCompanyName\InvoiceApp, then just set the *FileName* to Software\YourCompanyName\InvoiceApp.
4. You can override the name of the section that the grid's column/field information is stored in by setting the *SectionName* property. The default is blank which means that a unique section name will be generated based on the name of your form and the name of your grid.

InplaceEditor

Runtime only. *InplaceEditor* is the default editor used by the grid to edit a field's value. You may wish to reference this property if you need to access the currently edited value or dynamically manipulate the editor. For instance, if you wish to change the color of the editor when it has an invalid value. See the TwwDBGrid event *OnCheckValue* for an example of doing this.

Data Type: TwwInplaceEdit

You can use the following properties of the *inplaceeditor* dynamically at runtime.

WordWrap	Boolean
TColor	Color
TFont	Font
String	Text
Integer	SelStart
Integer	SelLength

Example: The following code will automatically exit the State field after the user has entered 2 characters.

```
Procedure TForm1.InvoiceGridKeyUp(Sender: TObject;
var Key: Word; Shift: TShiftState);
Function isValidChar(key: word): boolean;
begin
    result:= (key = VK_BACK) or (key=VK_SPACE) or
    (key=VK_DELETE) or
        ((key >= ord('0')) and (key<=VK_DIVIDE));
end;
begin
    with (Sender as TwwDBGrid) do begin
        if (InplaceEditor<>Nil) and
            (GetActiveField.FieldName='State') then
            begin
                if not isValidChar(key) then exit;
                if (length(InplaceEditor.Text)>=2) then begin
                    SelectedIndex:= SelectedIndex + 1;
                end
            end
        end;
    end;
end;
```

KeyOptions

This property defines specific settings for keyboard behavior within the grid.

Data Type: TSet()

Valid Values: dgEnterToTab, dgAllowDelete, dgAllowInsert

- dgEnterToTab* When True, the enter-key is automatically converted to a tab in the grid.
- dgAllowDelete* When True, the grid will attempt to delete the current record when the end-user enters the CTL-DELETE.
- dgAllowInsert* When True, the grid will attempt to insert a new record when the end-user enters the INSERT character.

LineColors

Use the *LineColors* property to override the color of the lines in the grid.

- DataColor** Set this property to override the color of the lines in the data cells
- FixedColor** Set this property to override the color of the lines in the fixed columns or borders
- HighlightColor** Set this property to override the color of the highlight lines
- ShadowColor** Set this property to override the color of the highlight lines

LineStyle

Specify whether the row and column lines are painted in 3d or as a single line. If this property is set to *glsDynamic*, then the cell is painted as a single line unless the background color is not white.

Valid Values: glsSingle, gls3D, glsDynamic

LoadAllRTF

In general it is recommended to use an embedded TwwDBRichEdit control with Custom Control Always Paints set to True. But, if you do not have a RichEdit control attached to that field, then this property is used to determine what is loaded into the grid. With LoadAllRTF set to False instead of loading the entire blob for each visible richedit field it will only load the first 3 packets. With LoadAllRTF set to True it will load the entire blob and display the text of the richedit.

MemoAttributes

This property contains a set of Boolean values that control the display of Memo data, as described below. The defaults are mSizable and mWordWrap.

Data Type: TSet()

Valid Values: mSizable, mWordWrap, mGridShow, mViewOnly (described below)

<i>mSizable</i>	When True, the end-user is allowed to resize the pop-up memo editor window. When False, the pop-up editor is displayed as a dialog box. The default value is True.
<i>mWordWrap</i>	When True, word wrapping is automatic. When False, the entire display scrolls horizontally as words are added. The default value is True.
<i>mGridShow</i>	When True, the memo data is displayed in the grid. When False, memo data is not displayed in the grid. The default value is False. (Warning: Enabling this option dramatically slows down the grid display since memo data must be retrieved from a file other than the table being accessed.)
<i>mViewOnly</i>	When True, the user may not edit the contents of the memo and only the OK button is displayed. When False, the user may edit the memo data and both OK and Cancel buttons are displayed. The default value is False. (Note: If the grid or field <i>ReadOnly</i> property is set to True, this property is automatically set to True.)
<i>mDisableDialog</i>	When True, the pop-up memo dialog for memo fields is disabled

MultiSelectOptions

This property defines specific settings for multiselection behavior within the grid.

Data Type: TSet()

Valid Values: msoAutoUnselect, msoShiftSelect

<i>msoAutoUnselect</i>	When True this property will unselect all records when a user clicks on a different record without the Ctrl key being pressed. In addition it will automatically select the current record. Note: When msoAutoUnselect is True (and an unselectall is triggered), the OnMultiSelectRecord event will not be fired. Use the <i>OnMultiSelectAllRecords</i> event to clear totals if you are calculating totals in the OnMultiSelectRecord event.
------------------------	---

msoShiftSelect When set to True this property allows the ability to select multiple contiguous records with the mouse while the shift key is pressed.

Options

Same as the TDBGrid with the following new options

Data Type: TSet()

New Valid Values: Wwdbigrd.dgMultiSelect, Wwdbigrd.dgWordWrap, Wwdbigrd.dgPerfectRowFit

dgMultiSelect When True, the grid will automatically use CTL-Click to select/deselect a record. This provides a convenient way to perform multi-selection. Embedded controls will not appear in the grid when this property is enabled, so turn this property off if you wish to edit. If you also wish to enable shift-select, see the MultiSelectOptions property

dgWordWrap Support word wrapping when displaying and editing text in the grid. When setting this property to True, also increase the *RowHeightPercent* property so your grid can display more than one line for each row.

dgPerfectRowFit This property is only used during design time. When True, the grid will automatically shrink the grid height so that there is no blank space at the bottom. This property allows you a convenient way to set the grid's height so that the rows fit perfectly in the grid frame during design time.

dgShowFooter When True, the grid will display a footer section at the bottom of the grid. In order for footer cells to appear within the footer, you need to assign a value to each footer cell using the *ColumnByName(FieldName).FooterValue* property. See also the grid's *OnUpdateFooter* event.

```
MyGrid.ColumnByName('Balance Due').FooterValue:= '100.00';
```

dgFooter3Dcells This property determines whether the footer cells have a three-dimensional (3-D) or two-dimensional look.

dgNoLimitColSize Set this to True if you wish to allow a column to shrink to a smaller size than its display label.

dgTrailingEllipsis Set this property to True to display trailing ellipsis in a data cell if the text will not completely fit in the cell. This property only has an effect when the cell is a single line.

dgShowCellHint Set this property to True to display the text of a cell as a tool-tip when its entire contents are not in view. You may wish to use the *OnCreateHintWindow* event to customize the size of the tool-tip window.

dgTabExitsOnLastCol	Set this property to True to allow the Tab key to exit the grid if the active column is the last column. Similarly if the active column is the first column, then shift-tab will go to the previous control. The Options.dgTabs must be true for this property to function.
dgFixedResizable	Set this property to True to allow fixed columns to be resizable in the same way as non-fixed columns.
dgFixedEditable	Set this property to True to allow fixed columns to be editable.
dgProportionalColResize	Set this property to true if you wish to automatically size all the columns to fit perfectly in the grid's client area. If the grid is resized, all the columns still fit perfectly. Any trailing column space after the last column is removed. (Note: To enable proportional column sizing, the grid's UseTFields property must be False.)
dgRowResize	Set this property to True to allow the end-user a way of resizing the sizes of the data rows by dragging the horizontal line in the indicator column. When resizing a row, all the data rows in the grid will use the new size.
dgRowLinesDisableFixed	Set this property to True to hide the row lines that normally appear in the fixed column area when dgRowLines is set to True.
dgColLinesDisableFixed	Set this property to True to hide the column lines that normally appear in the fixed column area when dgColLines is set to True.
dgFixedProportionalResize	When dgFixedProportionalResize is set to True, then this property determines whether or not the fixed columns will be resized along with the data columns when the grid is resized.
dgHideBottomDataLine	Set to false to prevent the last data row from painting a line underneath the data.
dgDbClickColSizing	Set to True to support the auto-sizing of columns when the title area is dbl-clicked. The column is sized so that it completely fits the widest displayed field in that column. <i>Note: The column will not enlarge the column if the column would occupy more than half the horizontal span of the grid.</i>

Example: When setting the TwwDBGrid's Options make sure that you scope the values. You can initialize, add or subtract values from *TwwDBGrid | Options* in the following way.

```
{The following initializes the Options at runtime}
wwDBGrid1.Options := [Wwdbigrd.dgEditing, Wwdbigrd.dgTabs,
```

```

Wwdbigrd.dgColLines, Wwdbigrd.dgRowLines,
Wwdbigrd.dgWordWrap];

{The following line will add titles in the grid.}
wwDBGrid1.Options := wwDBGrid1.Options + [Wwdbigrd.dgTitles];

{The following line will remove titles from the grid.}
wwDBGrid1.Options := wwDBGrid1.Options - [Wwdbigrd.dgTitles];

```

PadColumnStyle

This property allows you to fine-tune the painting if there are not enough records or columns to paint data in the entire client area of the grid.

- pcsPadHeader* The header title area is padded so that there is no whitespace on the right-hand side of the header area.
- pcsPadHeaderAndData* Any whitespace on the right-hand side of the grid or at the bottom of the grid is filled with the title color, or the title blended bitmap.
- pcsPlain* Any whitespace remains visible on the grid.

Data Type: TwwPadColumnStyle

PaintOptions

See the TwwDataInspector PaintOptions property.

PictureMaskFromDataSet

This property is only relevant if your datasource is attached to a TwwTable, TwwQuery, TwwQBE, or TwwClientDataset component, as it is always treated as false in other cases.

When customizing the picture masks through the select fields dialog (invoked by clicking on the selected property at design time), the mask information is stored in the related dataset if this property is True. Otherwise the mask information is stored as a property in the related visual component. By storing the mask information in the dataset, you do not need to re-enter the picture mask for other visual controls attached to this same database field.

Data Type: boolean

PictureMasks

The assigned picture mask information is stored in this property if PictureMaskFromDataset is false. See the *PictureMaskFromDataSet* property.

Data Type: TStrings

RowHeightPercent

This property controls the size of each row in the grid. By default this property is set to 100%, which uses the default height. You can scale the row heights proportionally by changing this value. For instance using 200% will cause each row to be twice as large. This property will often be used when you have enabled word-wrapping in the grid. See the *Options* property (dgWordWrap). You may also want to increase your RowHeightPercent if you are displaying a bitmap in the grid.

Data Type: Integer

Valid Values: A positive integer value

Selected

Clicking the “...” button or double-clicking the grid component displays the Select Fields dialog box. This dialog box allows you to select the fields you want displayed in the grid, their titles, widths, control types and link information. (See *Using the Select Fields Dialog Box* at the beginning of Chapter 4.) The default value is *all* fields selected, using the *field name* as it’s title, displayed as a *Field* control for a width equal to the number of characters in the field or the title, whichever is longer.

Data Type: TStrings

Valid Values: List of strings, with each entry being tab-delimited and containing the field name, the field width, and the field title.

Note: See also the grid’s *ColumnByName* method if you wish to set a column’s *ReadOnly*, *DisplayLabel*, *DisplayWidth*, and *FooterValue* properties during program execution.

Example: The following will clear the grid’s *selected* property and add two fields, each with a display width of 10.

```
with wwDBGrid1 do
begin
  Selected.Clear;
  Selected.Add('Buyer' + #9 + '10' + #9 + 'Buyer');
  Selected.Add('First Name' + #9 + '10' + #9 + 'First Name');
  ApplySelected;
end;
```

If *UseTFields* is set to True, modifying the *Selected* property does not update the grid until the dataset is re-opened. As a result, one should normally use the *Visible*, *Index*, *DisplayLabel*, and *DisplayWidth* properties of the TField to change the field attributes during program execution. Below is an example.

```
with wwDBGrid1, wwDBGrid1.DataSource.DataSet do
begin
  DisableControls;
  FieldByName('Field1').DisplayLabel := 'NewDisplayLabel';
  FieldByName('Field1').DisplayWidth := 6;
  FieldByName('Field2').Visible := False;
  FieldByName('Field3').Index := 0;
  EnableControls;
end;
```

SelectedList

This runtime only property returns a list of currently selected records.

Data Type: TList of TBookmarks

Valid Values: Use the InfoPower supported methods: *IsSelected*, *SelectRecord*, *UnselectRecord*, *SelectAll*, *UnselectAll*, and *SortSelectedList*

Example: How to delete all selected records of a TwwDBGrid from its related table.

```
procedure TForm1.DeleteButtonClick(Sender: TObject);
var i: integer;
```

```

begin
  with wwdbgrid1, wwdbgrid1.datasource.dataset do begin
    DisableControls;      {Disable controls to improve performance}
    for i:= 0 to SelectedList.Count-1 do begin
      GotoBookmark(SelectedList.items[i]);
      Freebookmark(SelectedList.items[i]);
      Delete;             { Delete Record }
    end;
    SelectedList.clear; { Clear selected record list }
                       { since they are all deleted }
    EnableControls;      { Re-enable controls }
  end;
end;

```

See also the TwwDBGrid *How-to* section - *Iterating through the list of selected records in a multi-selection grid*.

ShowHorzScrollBar

This property defines whether or not to display the horizontal scroll bar within the grid. The default value is True, which displays the horizontal scroll bar. Set this property to False when you do not want the horizontal scroll bar displayed.

Data Type: Boolean

ShowVertScrollBar

This property defines whether or not to display the vertical scroll bar within the grid. The default value is True, which displays the vertical scroll bar. Set this property to False when you do not want the vertical scroll bar displayed.

Data Type: Boolean

TitleAlignment

This property allows you to define how the text contained within each of the column headings is to be aligned. The default value is taLeftJustify. See also the *OnCalcTitleAttributes* event if you want to individually customize each column heading.

Data Type: Constant

Valid Values: taCenter, taLeftJustify or taRightJustify

TitleButtons

When this property is True, the column headings of each column in the grid will act as a button. When the user clicks on one, it will depress and fire the *OnTitleButtonClick* event.

Data Type: Boolean

Example: This example uses the wwDBGrid's *TitleButtonClick* event to change the table's selected index to that of the clicked on column and then set that column heading's color to yellow. It sets the table's index with the *IndexFieldName* property.

```

procedure TForm1.wwDBGrid1TitleButtonClick(Sender: TObject;
  AFieldName: String);
begin
  wwTable1.IndexFieldName := AFieldName;
end;

procedure TForm1.wwDBGrid1CalcTitleAttributes(Sender: TObject;
  AFieldName: String; AFont: TFont; ABrush: TBrush);

```

```

    var ATitleAlignment: TAlignment);
begin
    if (UpperCase(AFieldName)=
        UpperCase(wwTable1.IndexFieldName)) then
        ABrush.Color := clYellow;
end;

```

TitleColor

This property allows you to define the background color applied to both the column and row headings of the grid. The default value is *clBtnFace*.

Data Type: Constant

Valid Values: Valid Delphi color constant

TitleImageList

This property relates a *TImageList* to the grid's title. This ImageList is used in conjunction with the *OnCalcTitleImage* event. See the *OnCalcTitleImage* event for more information on how this property is used.

Data Type: TImageList

TitleLines

This property allows you to define the number of text lines to be used in the column title area. The default value is 1, which provides for a single column title line of text. To define a multi-line title: double-click the grid component to display the Select Fields dialog box; in the Selected Fields list box, select the field name you want to change the title of; modify the value displayed in the Title edit box within the Currently Selected Field group. Use the "~" character to separate lines within the title.

For example, a Title of "Customer~Account~Number" would require the *TitleLines* property to be set at 3 and would display each of the three words Customer, Account and Number on separate title lines at the top of the grid column.

Note: If the *TitleLines* property is set to a value less than the number of text lines you specify by using the "~" character, this character is displayed in the title text as though it were part of the title.

Data Type: Integer

Valid Values: A positive integer value

UseTFields

When the UseTFields property is set to true the *Selected* property's display settings are stored and retrieved from the dataset that the grid is attached to. When it is set to False the *Selected* property's display settings are stored with the grid. Defaults to True.

Data Type: Boolean

Modified properties

FixedColor

Replaced by the expanded *TitleColor* property described in the *Added properties* section.

Required property assignments

DataSource.

Added Events

OnAfterDrawCell

This event is fired after the grid paints each cell. Use this event to perform any additional painting for the cell. If you wish to disable certain aspects of the default drawing then use the *OnBeforeDrawCell* event.

Sender: TObject The *TwwDBGrid* associated with this event

DrawCellInfo: TwwCustomDrawGridCellInfo

See the *OnBeforeDrawCell* event for a description of this parameter.

OnBeforeDrawCell

This event is fired before the grid paints each individual cell. Use this event to disable certain types of painting operations (i.e. line drawing) from taking place for the cell.

Sender: TObject The *TwwDBGrid* associated with this event

DrawCellInfo: TwwCustomDrawGridCellInfo

Contains information about the cell to be painted, as well as properties which you can set to disable certain aspects of the cell painting. *TwwCustomDrawGridCellInfo* is a record structure with the following definition.

Rect: TRect Rectangle of cell that is about to be painted

Field: TField Field associated with cell

State: TGridDrawState State of the cell being drawn

DataCol, DataRow: integer Column and Row of the cell (0 based where the first field is 0, and the first displayed data row is 0). *DataCol* is -1 if the cell is in the indicator column, and *DataRow* is -1 if the cell is in the title area.

DefaultDrawBackground: Boolean Set to False to disable the background painting of the cell

<i>DefaultDrawHorzTopLine</i> : Boolean	Set to False to disable the horizontal line from being painted
<i>DefaultDrawHorzBottomLine</i> : Boolean	Set to False to disable the bottom horizontal line from being painted
<i>DefaultDrawContents</i> : Boolean	Set to False to disable the painting of the text within the cell

OnBeforePaint

This event fires before the grid starts painting each individual cell. Use this event to paint your own background for the TwwDBGrid.

Write an *OnBeforePaint* event handler to paint a background image to the inspector. The parameters for this event are as follows:

Sender: TObject The *TwwDBGrid* associated with this event

Example: The following example makes sure that the detail grid in a master/detail relationship paints its background image the same color as the current active background for the master record. This example is taken from our master/detail grid demo. The *CustomerGrid* is the master Grid and the *InvoiceGrid* is the detail grid.

```

procedure TMasterDetailGridForm.InvoiceGridBeforePaint(Sender: TObject);
begin
  if CustomerGrid.IsActiveRowAlternatingColor then begin
    with TwwDBGrid(Sender) do
      Canvas.CopyRect(ClientRect,
        CustomerGrid.PaintOptions.AlternatingColorBitmap.Canvas,
        ClientRect);
  end
  else
    with TwwDBGrid(Sender) do
      Canvas.CopyRect(ClientRect,
        CustomerGrid.PaintOptions.OrigBitmap.Canvas, ClientRect);
  end;
end;

```

OnCalcCellColors

This event, which executes just prior to painting the field values within each grid cell, allows you to change both the font and background colors of individual cells. Continuous color support is available so that you can have unlimited colors in the Grid. Previously you were restricted to just the basic Windows color palette.

The parameters for this event are as follows.

<i>Sender</i> : TObject	TwwDBGrid that is associated with this event.
<i>Field</i> : TField	Field that is associated with the column about to be drawn.
<i>State</i> : TGridDrawState	State of the cell being drawn
<i>Highlight</i> : Boolean	True if the cell is to be highlighted

<i>AFont</i> : TFont	Font to draw the text in the cell
<i>ABrush</i> : TBrush	Brush to use for drawing the cell background.

The examples below demonstrate how to accomplish these color changes. (Note: The parameters *ABrush*, *AFont*, *highlight* and *State* are all defined in the CalcCellColors header.)

Example 1. Changing the font and background color of individual grid cells. This example uses a State field within a TwwDBGrid. If the value in the State field is “CO”, the font color is changed to red and the background color is changed to yellow for the grid column ‘State’.

```

procedure TForm1.wtwDBGrid1CalcCellColors(
  Sender: TObject; Field: TField;
  State: TGridDrawState; Highlight: Boolean;
  AFont: TFont; ABrush: TBrush);
begin
  if (Field.FieldName = 'State') then begin
    if (Field.Text = 'CO') then begin
      AFont.Color := clRed;
      if (not Highlight) then ABrush.Color := clYellow;
    end;
  end;
end;

```

Example 2. Changing the font color of an entire grid row based on the value contained in an individual cell within the same row. This example uses the BalanceDue field of a TwwTable object named InvoiceTable within a TwwDBGrid. If the BalanceDue amount is greater than zero, the font color of the entire row is changed to red...

```

if (InvoiceTableBalanceDue.Value > 0) then Afont.Color := clRed;

```

OnCalcTitleAttributes

This event which executes just prior to painting each title cell, allows you to change both the font and background colors of each title header, as well as individually control the title alignments.

The parameters for this event are as follows.

<i>Sender</i> : TObject	TwWDBGrid that is associated with this event.
<i>AFieldName</i> : String	Name of the field that is associated with the column title about to be drawn.
<i>AFont</i> : TFont	Font to draw the text in the title cell
<i>ABrush</i> : TBrush	Brush to use for drawing the title cell background.
<i>ATitleAlignment</i> : TAlignment	Alignment used to draw the title cell

Example: The following right justifies the column header for the field *Total Invoice*. All other fields use the grid’s *TitleAlignment* property. The example also changes the background color for this column header cell to *clYellow*.

```

TForm1.wtwDBGrid1CalcTitleAttributes(Sender: TObject;

```



```

AFieldName: String; AFont: TFont; ABrush: TBrush;
var ATitleAlignment: TAlignment);
begin
  if (AFieldName = 'Total Invoice') then begin
    ATitleAlignment := taRightJustify;
    ABrush.Color := clYellow;
  end;
end;

```

OnCalcTitleImage

This event allows you to define the image that appears in the title cell at runtime. The *TitleImageList* property must be assigned to an *ImageList* for this property to have any effect.

Sender: TObject TwwDBGrid that is associated with this event.

Field: TField Field that is associated with the cell about to be drawn.

TitleImageAttributes: TwwTitleImageAttributes
Attributes of how the image should be drawn in the cell.
TwwTitleImageAttributes is a record containing the following:

ImageIndex: Integer Image index from *TitleImageList* to paint into the cell. Default is -1, which indicates no image is painted into the cell.

Alignment: TAlignment How to align the image. The image alignment can be set to *taLeftJustify*, *taCenter*, or *taRightJustify*. Default is *taRightJustify*.

Margin: Integer Number of pixels between edge of cell and the image. Default is 3 pixels.

IsGroupHeader: boolean This value returns true if the cell being painted is the group header, as opposed to a sub-header of the group. You may wish to refer to this property if you wish to paint the group header cell with a different image than its sub-headers.

Example The following example displays the 4th image (0 based) from the *TitleImageList* if the cell being painted is the field that the grid is sorted by. The code below assumes that your grid is attached to a *TwwTable*.

```

procedure TBtnGridForm.wwDBGrid1CalcTitleImage(
  Sender: TObject; Field: TField;
  var TitleImageAttributes: TwwTitleImageAttributes);
begin
  if Field=(Field.Dataset as TTable).indexFields[0] then
    TitleImageAttributes.imageIndex:= 4
  else
    TitleImageAttributes.imageIndex:= -1
end;

```

OnCellChanged

This event allows you to perform some custom action after the user moves to a new cell in the grid. In many cases this event is more useful than the *OnRowChanged*, *OnColEnter*, and *OnColExit* events, as it allows you to centralize all your custom code for cell movement into a

single event. To determine information about the new cell being entered, you can use the *GetActiveField* method.

The parameters for this event are as follows.

Sender : TObject TwwDBGrid that is associated with this event.

Example: (Updating a TLabel component to contain the text value of the active cell): The following example updates a label component (Label1), to show the active cell's text.

```
procedure TForm1.wwDBGrid1CellChanged(Sender: TObject);
begin
    Label1.caption:= (Sender as TwwDBGrid).GetActiveField.Text;
end;
```

OnCheckValue

This event allows you to perform some custom action while the user is editing cells in the grid. This event is only fired if you have a picture mask defined for the field. You can test the parameter *PassesPictureTest* to see if the user's entry passes the picture mask requirements. Note: If you want to check fields with custom editors attached, then you will also need to attach your code to the custom editor's *OnCheckValue* event.

The parameters for this event are as follows.

Sender : TObject Editor within the TwwDBGrid that is being checked.
Sender is a TwwInplaceEdit component. See the TwwDBGrid runtime property *InplaceEditor* for more information on the TwwInplaceEdit type.

PassesPictureTest : Boolean True if edited text passes the picture mask constraints.

Example (Coloring of a TwwDBGrid cell during editing) : The following example will give cells being edited with the default editor a yellow background whenever the edited text does not satisfy the picture mask constraints. If you are also attaching your own custom editors within the grid (i.e. TwwDBComboBox, TwwDBEdit), your code will be slightly different. See the *Picture Mask* section for more information on this event.

```
procedure TForm1.wwDBGrid1CheckValue(Sender: TObject;
PassesPictureTest: Boolean);
begin
    with Sender as TwwInplaceEdit do begin
        if not PassesPictureTest then color:= clYellow
        else color:= clWhite
    end
end;
```

OnColumnMoved

OnColumnMoved occurs when the user moves a column using the mouse. Write an OnColumnMoved event handler to take specific action just after a column in the grid has been moved. To retrieve the field name given a column number, you can use the *FieldName* method of the grid.

The parameters for this event are as follows.

<i>Sender</i> : TObject	TwwDBGrid that is associated with this event.
<i>FromIndex, ToIndex</i> : integer	The <i>FromIndex</i> parameter gives the position the column previously occupied. The <i>ToIndex</i> parameter gives the position the column now occupies.

OnColWidthChanged

This event occurs when the user resizes a column with the mouse. Write an `OnColWidthChanged` event handler to take specific action just after a column in the grid has been resized.

The parameters for this event are as follows.

<i>Sender</i> : TObject	TwwDBGrid that is associated with this event.
<i>Column</i> : integer	Column being resized <i>OnDrawDataCell</i>

OnCreateDateTimePicker

Use this event to customize the properties of the default `DateTimePicker` control for `DateTime` fields. The parameters are as follows:

<i>Sender</i> : TObject	TwwDBGrid that is associated with this event.
<i>ADateTimePicker</i> : TwwDBCUSTOMDateTimePicker	TwwDBDateTimePicker control created for datetime fields.

OnCreateHintWindow

Use this event to customize the painting of the hint window. This event is fired before the hint window is actually displayed. The parameters are as follows:

<i>Sender</i> : TObject	TwwDBGrid that is associated with this event.
<i>HintWindow</i> : TwwGridHintWindow	Hint window that was created. You can refer to its <code>Canvas</code> property to customize how the hint window is painted.
<i>AField</i> : TField	Field that the hint window is displaying information about.
<i>R</i> : TRect	Rectangle coordinates of the hint window
var <i>WordWrap</i> : boolean	Set <i>WordWrap</i> to <code>True</code> to cause the hint window to wrap
var <i>MaxWidth</i> : integer	Set <i>MaxWidth</i> to limit the width of the hint window
var <i>MaxHeight</i> : integer	Set <i>MaxHeight</i> to limit the height of the hint window

var DoDefault: boolean Set *DoDefault* to False if you wish to prevent the grid from painting the hint window.

Example: The following code attached to this event makes the hint window's background Blue.

```
HintWindow.Canvas.Brush.color:= clBlue;
```

OnDitto

This event is fired for each field that is being copied during a ditto operation. See also the *DittoAttributes* event.

<i>Sender</i> : TObject	TwwDBGrid that is associated with this event.
<i>DataSet</i> : TDataSet	DataSet associated with ditto operation
<i>Field</i> : TField	Field associated with ditto operation
<i>var DittoValue</i> : String	Assign this property to change the ditto value used to fill the field
<i>var AllowDitto</i> : Boolean	Assign to false to prevent this field from being copied during the ditto operation

OnDrawDataCell

OnDrawDataCell occurs when the grid needs to paint a data cell.

<i>Sender</i> : TObject	TwwDBGrid that is associated with this event.
<i>Rect</i> : TRect	Boundaries, in pixels, of the data cell to be painted
<i>Field</i> : TField	<i>TField</i> associated with the cell to be painted
<i>State</i> : TGridDrawState	TGridDrawState defines the possible states of a cell when drawing occurs. This property is a set containing zero or more of the following : {gdSelected, gdFocused, and gdFixed}.
<i>gdSelected</i>	The cell in the grid is selected.
<i>gdFocused</i>	The cell in the grid has input focus.
<i>gdFixed</i>	The cell is in the fixed (nonscrolling) region of the grid.

OnDrawFooterCell

This event occurs just before a footer cell is to be painted. Write an event handler to change the attributes of the footer cell. The parameters for this event are as follows.

<i>Sender</i> : TObject	TwwDBGrid that is associated with this event.
-------------------------	---

<i>Canvas</i> : TCanvas	Canvas used to paint the cell. Reference this property to change the cell's brush or font attributes.
<i>FooterCellRect</i> : TRect	Boundaries, in pixels, of the footer cell to be painted
<i>Field</i> : TField	<i>TField</i> associated with the footer cell to be painted
<i>FooterText</i> : string	Text of the footer cell
<i>DefaultDrawing</i> : boolean	Set this event to False to disable the default drawing of the footer cell. This property defaults to True.

OnDrawGroupHeaderCell

This event occurs just before a cell in the header is painted. Write an event handler to change the attributes of a cell in the header.

The parameters for this event are as follows.

<i>Sender</i> : TObject	TwwDBGrid that is associated with this event.
<i>Canvas</i> : TCanvas	Canvas used to paint the cell. Reference this property to change the cell's brush or font attributes.
<i>GroupHeaderName</i> : String	Name of GroupHeaderCell being painted.
<i>Rect</i> : TRect	Boundaries, in pixels, of the group header cell to be painted
<i>Var DefaultDrawing</i> : boolean	Set this event to False to disable the default drawing of the title cell. This property defaults to True.

OnDrawTitleCell

This event occurs just before a cell in the header is painted. Write an event handler to change the attributes of a cell in the header.

The parameters for this event are as follows.

<i>Sender</i> : TObject	TwwDBGrid that is associated with this event.
<i>Canvas</i> : TCanvas	Canvas used to paint the cell. Reference this property to change the cell's brush or font attributes.
<i>Field</i> : TField	<i>TField</i> associated with the title cell to be painted
<i>Rect</i> : TRect	Boundaries, in pixels, of the title cell to be painted
<i>Var DefaultDrawing</i> : boolean	Set this event to False to disable the default drawing of the title cell. This property defaults to True.

OnExportField

This event occurs when the *ExportOptions* | *Save* method is called. See *ExportOptions* for more details. This event fires for each field of each record that is to be exported allowing you to decide which fields that you wish your end-users to export to a file.

<i>Grid</i> : TwwDBGrid	TwwDBGrid that is associated with this event.
<i>Field</i> : TField	TField being updated
<i>Var Accept</i> : boolean	Set this parameter to False to prevent this Field from being exported. This property defaults to True.

OnFieldChanged

This event occurs after a cell is modified and its contents have been flushed to the TField buffers. Use this to perform your own custom action after a field in the grid has been modified.

<i>Sender</i> : TObject	TwwDBGrid that is associated with this event.
<i>Field</i> : TField	TField being updated

Warning: Be careful not to include any code in this event that could cause infinite recursion, such as modifying a field value (which would trigger this event again).

OnLeftColChanged

This event occurs after the grid scrolls horizontally.

OnMemoClose

This event executes immediately after the memo dialog box is closed. The parameters for this event are as follows.

<i>Grid</i> : TwwDBGrid	TwwDBGrid from which the memo dialog is being opened.
<i>Cancel</i> : Boolean	True when the user cancels the dialog without making changes.

OnMemoOpen

This event executes just before the memo dialog box is opened. Reminder: you can access any of the TwwMemoDialog properties to customize the dialog.

The parameters for this event are as follows.

<i>Grid</i> : TwwDBGrid	TwwDBGrid component from which the memo dialog is being opened.
<i>MemoDialog</i> : TwwMemoDialog	TwwMemoDialog being opened.

Example 1: The following code changes the height and position of the memo dialog box:

```

procedure TForm1.wwDBGrid1MemoOpen(Grid: TwwDBGrid;
MemoDialog: TwwMemoDialog);
begin
MemoDialog.dlgLeft:= 1;
MemoDialog.dlgHeight := 200;
end;

```

Example 2: The following steps will change the wwDBGrid's memo dialog box's background color to red. The grid's embedded TwwMemoDialog component is not created until runtime, so it is necessary to assign the OnInitDialog event during program execution.

1. Create an *OnInitDialog* event for a memo dialog box that is embedded in a TwwDBGrid.
 - a) Drop a TwwMemoDialog Component on your form, and add the following line of code to the *OnInitDialog* event of the TwwMemoDialog component:

```

Procedure TForm1.wwMemoDialog1InitDialog(Dialog: TwwMemoDlg)
begin
Dialog.Memo.Color := clRed;
end;

```
 - b) Delete the TwwMemoDialog Component from your form. The event and its declaration will be left in your code.
2. Now, add code to the grid's *OnMemoOpen* event that assigns the *OnInitDialog* event to the code you created above.

```

Procedure TForm1.wwDBGrid1MemoOpen(Grid: TwwDBGrid;
MemoDialog: TwwMemoDialog)
begin
MemoDialog.OnInitDialog := wwMemoDialog1InitDialog;
end;

```

OnMouseDown

This event (TMouseEvent) executes when the user presses a mouse button. See Delphi's documentation for further details on using this event. The parameters for this event are as follows.

<i>Sender</i> : TObject	The TwwDBGrid that received the MouseDown event.
<i>Button</i> : TMouseButton	Determines which mouse button the user pressed, (mbRight, mbLeft, mbMiddle).
<i>Shift</i> : TShiftState	Indicates which shift keys (Shift, Ctrl, or Alt) and mouse buttons were down when the user pressed or released the mouse button that generated the TMouseEvent. TShiftState = set of (ssShift, ssAlt, ssCtrl, ssRight, ssLeft, ssMiddle, ssDouble).
<i>X</i> and <i>Y</i> : Integer	Screen pixel coordinates of the mouse pointer. You can use InfoPower's MouseCoord(x,y) method to further determine which grid column (x) and row (y) were clicked on.

Example: See the example under the *FieldName* property.

OnMouseMove

This event (TMouseMoveEvent) executes when the user moves the mouse. See Delphi's documentation on the *OnMouseMove* event for further details on using a TMouseMoveEvent.

OnMouseUp

This event (TMouseEvent) executes when the user releases a previously pressed mouse button. See the *OnMouseDown* event for details on using a TMouseEvent.

OnMultiSelectRecord

This event (TwwMultiSelectRecordEvent) executes before a single record is selected or unselected in the TwwDBGrid. Use this event to perform some custom action based on a user selecting a record. The parameters for this event are as follows.

<i>Grid</i> : TwwDBGrid	The TwwDBGrid that is being used for multiselection.
<i>Selecting</i> : Boolean	Indicates whether the user is selecting or unselecting a record. True if the user is selecting.
<i>Accept</i> : Boolean	Set this property to True if the Grid should accept the selection/unselection that was made, otherwise it will reject it. This property defaults to True.

Example: This example prevents the user from selecting more than 3 records.

```
TForm1.wwDBGrid1MultiSelectRecord(Grid: TwwDBGrid;  
  Selecting: Boolean; var Accept: Boolean);  
begin  
  if (Selecting and (Grid.SelectedList.Count>=3)) then  
    Accept := False;  
end;
```

OnRowChanged

This event occurs after the user moves to a new row in the grid.

OnTitleButtonClick

The OnTitleButtonClick event occurs when the user clicks on one of the column headings in the TwwDBGrid. The *TitleButtons* property must be true in order for this event to occur. See also the *TitleButtons* property

OnTopRowChanged

This event executes when the grid scrolls to a different record, causing the top record in the grid to point to a different record.

OnUpdateFooter

The *TwwDBGrid* provides the convenient event, *OnUpdateFooter*, to assist you in updating the footer on certain pre-defined conditions. The *OnUpdateFooter* event is called when you

post or delete a record associated with the grid. If the grid is attached to a detail table, then the event also is called when the user moves to a new master record. If you require other situations where the footer should be updated, then you will need to invoke your code that updates the footer there. The following code updates the footer whenever the user moves to a new customer, and posts or deletes a record from the invoice table.

```
procedure TBitmapForm.InvoiceGridUpdateFooter(Sender: TObject);
begin
    InvoiceGrid.ColumnByName('Balance Due').FooterValue:=
        FloatToStrF(SumQuerySumOfBalanceDue.asFloat, ffCurrency, 10, 2);
end;
```

OnURLOpen

This event executes when the end-user clicks on a URL-Link field. Use this event to override the default behavior and provide some other default action.

Added Methods

ApplySelected

This method refreshes the grid based on the selected property. Call this method if you manipulate the selected property with your own code. See example under the `TwwDBGrid.Selected` property.

ColumnByName

This method allows you to retrieve and set certain column attributes during program execution. You can customize the following column properties with this method: (`ReadOnly`, `DisplayLabel`, `DisplayWidth`, and `FooterValue`, `DisableSizing`). Use *DisableSizing* to allow an embedded control to be larger than the cell size when the embedded control receives focus.

Example 1: The following code updates the `FooterValue` for a column.

```
MyGrid.ColumnByName('Balance Due').FooterValue:= '100.00';
```

Example 2: The following code sets the 'Balance Due' column to `ReadOnly`

```
MyGrid.ColumnByName('Balance Due').ReadOnly:= True;
```

Example 3: The following code changes the size of a richedit control embedded in a grid during editing. This allows the edit control to occupy more screen real-estate during actual editing within the control.

```
procedure TLargeGridEditForm.wwDBRichEdit1Enter(Sender: TObject);
begin
    with wwDBGrid1 do begin
        ColumnByName('RINTERESTS').DisableSizing:=True;
        wwDBRichEdit1.Width:= 200;
        wwDBRichEdit1.Height:= 40;
    end;
end;
```

DittoField

This method is called for each field, before each ditto operation takes place. Subclass this method if you wish to change the ditto behavior. You can call this method to force a ditto operation to take place in the currently active grid field.

SelectedField: TField Field to be copied
Direction: TwwDittoDirection Defaults to *wwDittoPrior*. See
DittoAttributes.DittoDirection property

FlushChanges

This method flushes the InplaceEditor's contents to the TField buffers. You may wish to call this method if you are invoking the RecordViewDialog, so that any current editing will be recognized by the record-view.

FieldName

This method returns the name of the field associated with a column.

```
Function FieldName(Column: integer): string;
```

GetActiveCol

This method returns the currently active column index of the TwwDBGrid.

```
Function GetActiveCol: Integer;
```

GetActiveField

This method returns the currently active field in a TwwDBGrid. The function declaration is as follows:

```
Function GetActiveField: TField;
```

GetActiveRow

This method returns the currently active column index of the TwwDBGrid.

```
Function GetActiveRow: Integer;
```

GetPriorRecordText

This method retrieves a field value from the previous record. The field retrieved is specified by *AFieldName*, and the value is placed into *AText*. If the method is unable to retrieve a value, it returns *False*.

```
function GetPriorRecordText(  
    AFieldname: string; var AText: string): boolean;
```

GetNextRecordText

This method retrieves a field value from the next record. The field retrieved is specified by *AFieldName*, and the value is placed into *AText*. If the method is unable to retrieve a value, it returns *False*.

```
function GetNextRecordText(  
    AFieldname: string; var AText: string): boolean;
```

InvalidateCurrentRow

Redraws the grid's currently active row by invalidating every cell in the current row. This is not the same as calling the table's *refresh* method, which would refetch the data from the physical table.

```
Procedure InvalidateCurrentRow;
```

IsSelected

This method tests if the active grid row is selected. Before a record can be selected, you need to configure the grid for multi-selection by following the steps in the TwwDBGrid's *How To* section. The grid row you are testing must be currently displayed in the grid for this method to be accurate. When using this method within the *OnCalcCellColors* event, it returns True when the cell's related record is selected.

```
Function isSelected: boolean;
```

Example: This example changes all selected rows so that their background color is clBlue.

```
procedure TGridDemo.wwDBGrid1CalcCellColors(Sender: TObject;  
    Field: TField; State: TGridDrawState;  
    highlight: Boolean; AFont: TFont; ABrush: TBrush);  
begin  
    if wwdbgrid1.isSelected then ABrush.Color:= clBlue;  
end;
```

LoadFromIniFile

This method loads the grid column information from the registry or .INI file. It respects the IniFile property settings. See also the *SaveToIniFile* method.

MouseCoord

This method converts the current screen coordinates of the mouse pointer to grid coordinates, which are valuable when using the new mouse-based events described earlier. The function declaration is as follows:

```
Function MouseCoord(X, Y: integer): TGridCoord;
```

TGridCoord is a record containing the column and row as follows:

```
TGridCoord = record  
    X: LongInt; { column }  
    Y: LongInt; { row }  
end;
```

Example: The following code displays the grid column number that the user clicked on:

```
procedure TGridDemo.wwDBGrid1MouseUp(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState;
    X, Y: Integer);
    var coord: TGridCoord;
begin
    coord := wwDBGrid1.MouseCoord(x,y);
    MessageDlg('You have selected column ' + IntToStr(coord.x),
        mtInformation, [mbOK], 0);
end;
```

SaveToIniFile

This method saves the grid column information to the registry or .INI file. It respects the IniFile property settings. See also the LoadFromIniFile method.

SelectAll

This method selects all the records in the current grid for multiselection. Turn the TwwDBGrid's *Option | dgMultiSelect* property to True when using this method.

```
Procedure SelectAll;
```

SelectRecord

This method adds the record relating to the current grid row to the list of currently selected records.

```
Procedure SelectRecord;
```

SetActiveField

This method allows you to change the currently active field, or column, in your grid. The function declaration is as follows:

```
Procedure SetActiveField(AFieldName: string);
```

AFieldName is the name of the table field you want to make active, or move to.

SetActiveRow

This method allows you to change the currently active visible row in your grid. The function declaration is as follows:

```
Procedure SetActiveRow(val: integer);
```

SetControlType

This method allows you to modify the control type of a grid column while your program is executing. If you wish for the whole grid to immediately repaint after you call this method, then call the *Invalidate* method of the Grid. For instance this may be desired when you set a column to a checkbox since every row would have a visible checkbox.

The function declaration is as follows:

```
procedure SetControlType(AFieldName: string;
    AComponentType: TwwFieldControlType; AParameters: String);
```

AFieldName is name of the field in the grid that is changing.

AComponentType is the type of component to give this grid column. It can be one of the following (fctField, fctBitmap, fctCheckbox, fctCustom, fctRichEdit)

AParameters depends on the *AComponentType* parameter, as demonstrated below:

fctField: This parameter should always be blank, like the following:

```
wwDBGrid1.SetControlType('Last Name', fctField, '');
```

fctCheckbox: This parameter is a ‘;’ delimited string that contains two values. The first value is used when the checkbox is checked, and the second value is used when the checkbox is unchecked. The following code example demonstrates how to define the field Buyer as a checkbox with the two possible values, Yes and No:

```
wwDBGrid1.SetControlType('Buyer', fctCheckBox, 'Yes;No');  
wwDBGrid1.Invalidate;
```

fctImageIndex: This parameter is either set to ‘Shrink To Fit’, or ‘Original Size’. If it is set to ‘Shrink To Fit’, then the image will shrink to fit the size of the cell it is being painted in.

```
wwDBGrid1.SetControlType('ImageFld', fctImageIndex, 'Original Size');
```

fctBitmap: This parameter is a ‘;’ delimited string that contains two values. The first value is the type of scaling to be used when the bitmap is copied into the cell. The second value is what kind of copy is performed when the bitmap is painted into the cell.

```
wwDBGrid1.SetControlType('MyBitmap', fctBitmap,  
    'Original Size;Source Copy');  
wwDBGrid1.Invalidate;
```

Valid values for the bitmap scaling are one of the following.

```
Original Size  
Stretch To Fit  
Fit Height  
Fit Width
```

Valid values for the copy operation are one of the following

```
Source Copy  
Source Paint  
Source And  
Source Invert  
Source Erase  
Not Source Copy  
Not Source Erase  
Merge Paint
```

For more information on the meaning of these values, see Chapter 4 on the Select Fields Dialog.

fctCustom: This parameter is the name of a InfoPower supported edit control

Example. The following code sets the TwwDBGrid's Zip column to use the lookup component named wwDBLookupCombo1:

```
wwDBGrid1.SetControlType('Zip', fctCustom, 'wwDBLookupCombo1');
```

fctURLLink : There are no parameters for this control type

fctRichEdit: This parameter is the name of an InfoPower TwwDBRichEdit component.

Example. The following code sets the TwwDBGrid's RichEdit column to use the TwwDBRichEdit component named wwDBRichEdit1:

```
wwDBGrid1.SetControlType('RichEdit', fctRichEdit, 'wwDBRichEdit1');
```

SetPictureAutoFill

Use this method if you want to change the *Picture* | *AutoFill* property of a field in a grid at runtime.

```
procedure SetPictureAutoFill(Fieldname: string; AutoFill: boolean);
```

SetPictureMask

Use this method if you want to change the picture mask of a field in a grid at runtime.

```
procedure SetPictureMask(String FieldName, String Mask);
```

SizeLastColumn

Use this method if you wish to programmatically size the last column to stretch across the remaining whitespace to the right of the grid. You may wish to use the Grid's *Options* | *dgProportionalColResize* property instead.

```
procedure SizeLastColumn;
```

SortSelectedList

This method sorts the list of selected records in the current index order using quicksort. Usually you will want to call *SortSelectedList* before accessing the *SelectedList* property.

```
procedure SortSelectedList;
```

UnselectAll

This method unselects all the records in the current grid for multiselection. Turn the TwwDBGrid's *Option* | *dgMultiSelect* property to True when using this method.

```
procedure UnselectAll;
```

UnselectRecord

This method removes the record relating to the current grid row from the list of currently selected records.

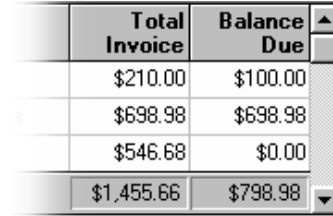
```
procedure UnselectRecord;
```

How To

Display a footer section with column summary information

To enable the grid to display a footer requires the following two actions:

- Set the Options | `dgShowFooter` to True.
- Update the footer cell by using the grid's `ColumnByName` method (i.e. `OnUpdateFooter` event).



	Total Invoice	Balance Due
	\$210.00	\$100.00
	\$698.98	\$698.98
	\$546.68	\$0.00
	\$1,455.66	\$798.98

You will need to determine when the footer cell should be updated (what events should trigger the updating of the footer, i.e. after you post a record). The *TwwDBGrid* provides the convenient event, `OnUpdateFooter`, to assist you in updating the footer on certain pre-defined conditions. The `OnUpdateFooter` event is called when you post or delete a record associated with the grid. If the grid is attached to a detail table, then the event also is called when the user moves to a new master record. If you require other situations where the footer should be updated, then you will need to invoke your code that updates the footer there.

See the InfoPower demo form, `GRDBITMP.PAS`, for a complete example of displaying a footer. This demo contains code and objects to reveal other aspects of InfoPower, so pay particular attention to the steps outlined below to understand which particular code and objects are associated with the footer.

This demo displays all the invoices for a given customer in the grid, and displays the sums of the *Balance Due* and *Total Invoice* fields in the footer. The footer is updated whenever you move to a new customer, post a record in the invoice table, or delete a record from the invoice table. The following details the setup of this demo form with respect to displaying and updating the footer.

1. The property `Options | dgShowFooter` of the *InvoiceGrid* is set to True.
2. The demo relies upon SQL to calculate the sums of the invoice information for a given customer. The *TwwQuery SumQuery* has the following property assignments:

```
SQL: select sum(ip4inv."balance due"),
           sum(ip4inv."Total Invoice") from ip4inv
       where ip4inv."Customer No"=:CustNo
```

Params: CustNo declared as a Data Type of Integer.

DatabaseName: InfoDemo5

3. The grid has the following code attached to its `OnUpdateFooter` event.

```
procedure TBitmapForm.InvoiceGridUpdateFooter(Sender: TObject);
begin
  SumQuery.active:= False;
  SumQuery.ParamByName('CustNo').asInteger:=
    CustomerTable.FieldByName('Customer No').asInteger;
  SumQuery.active:= True;

  InvoiceGrid.ColumnByName('Balance Due').FooterValue:=
```

```

FloatToStrF(SumQuerySumOfBalanceDue.asFloat,
ffCurrency, 10, 2);

InvoiceGrid.ColumnByName('Total Invoice').FooterValue:=
FloatToStrF(SumQuerySumOfTotalInvoice.asFloat,
ffCurrency, 10, 2);
end;

```

Create True Expandable master/detail relationships in a single TwwDBGrid.

InfoPower brings you a new paradigm to display and *edit* your master/detail relationships. Detail grids can be initially hidden, and then expanded into full view when the end-user expands a expand/collapse button (TwwExpandButton) in the parent grid. Each child-grid is fully customizable as in the parent grid, and the control preserves the liveness of each expanded detail grid. See *TwwExpandButton* for an explanation on how to setup this relationship with the InfoPower grids.

Summary and Expandable Fields in Grid using the TwwDataInspector.

Account #	Name	ShippingAddress	Interests/Hobbies
1023495	Jennifers Ardeny	100 Cranberry St., Abilene, MA 02181	Enjoys
		Street 100 Cranberry St. City Abilene State Massachusetts Zip Connecticut CT	
2094056	Arthur Jones	10 Hunne	
1209395	Debra Parker	74 South	Owns his <i>BMW</i>
3094095	Dave Sawyer	101 Oakla	Retired. Enjoys
1024034	Cindy White	1 Wentwo	Enjoys fishing.

Use expand/collapse buttons to allow the user to edit a composite field. You can display a calculated field such as full name (composed of first name + last name), and then the user can expand the composite calculated field to edit the individual portions. The advantages are obvious. See and edit more fields in a natural way.

1. Creating the Summary Fields.

- Drop a TTable on your form set the DatabaseName to InfoDemo5 and the TableName to clients.dbf.
- Double click on it to bring up the Field's Editor. Then create a string calculated fields called ShippingAddress.
- Now in the Table's OnCalcFields event do something like the following

```

procedure TGridExpandForm.Table1CalcFields(DataSet: TDataSet);
begin
  with dataset do begin
    {Set the Shipping Address Composite Field}
    fieldbyname('ShippingAddress').asstring:=
      fieldbyname('Address_1').asstring + ', ' +

```



```

        fieldbyname('City').asString + ', ' +
        fieldbyname('State').asString + ' ' +
        fieldbyname('Zip').asString;
    end;
end;

```

2. Setting up the dropdown TwwDataInspector Control.

- Drop a TwwDataInspector and assign the TwwDataInspector's Datasource property to the same as the TwwDBGrid's that you are using.
- Double Click and remove all fields except: ADDRESS_1,CITY,STATE,ZIP.
- We recommend other customizations like setting Ctl3D to False (If you do not see this in the object inspector, then right click on the object inspector and select View | Legacy in order to see Legacy properties),using dotted line styles for the data and caption cells, or hiding the caption column.

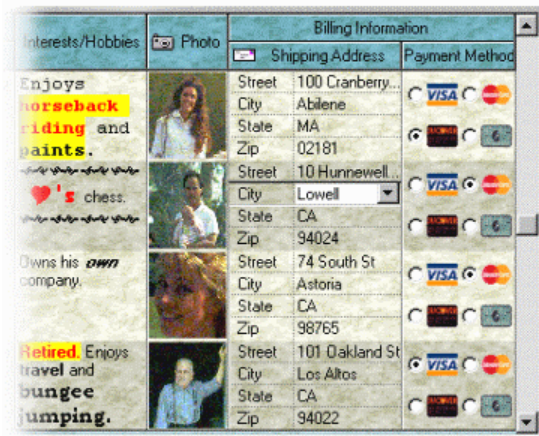
3. Attaching a dropdown expandable datainspector to the cell.

- Drop a TwwExpandButton on your form and set the *Grid* property to the TwwDataInspector used in step 2.
- Double Click on the grid and select the field you wish to attach this to. In this case it would be ShippingAddress. Then go to the Edit Control tab page, set Control Type to CustomEdit and in the Control Name dropdown combo select the Expand Button control.

Multiple Row Record Display

InfoPower's TwwDataInspector can also be embedded in the grid, giving a multi-row record display.

- Follow the steps in 2b in the How To topic for "Summary and Expandable Fields in Grid using the TwwDataInspector" to setup a datainspector.
- Attach the control to the desired grid column by using the *Select Fields Dialog | Edit Control* as defined in chapter 4 and then set the checkbox for Custom Control Always Paints to True on the Edit Control tab page.
- Set the RowHeightPercent of the Grid to be as tall as necessary. In the example above set it to 410 if you are showing four fields. Embed other multi-line memos, richedits, radiogroups, or dbimages for a powerful and space efficient way to edit and view your data.



Edit a grid's field using a custom control (i.e. combo box, spin edit, lookup combo, date time picker, etc.)

InfoPower's grid allows you attach different types of edit controls to any of the grid's columns. There are two basic steps in attaching a control. They are as follows:

1. Create the control that you wish to attach to the grid, and define its properties so it reflects the look and behavior you desire. Place this control on the same form as the grid.
2. Attach the control to the desired grid column by using the *Select Fields Dialog | Edit Control* as defined in chapter 4.

Edit a grid's field using a lookup combo box and a lookup field.

When using a *TwwDBLookupCombo* or a *TwwDBLookupComboDlg* component within the grid, you may desire to store a code value into the table and display a descriptive value from the lookup table to the end-user. In order to ensure that the user does not see the code value, you will need to do the following additional steps after creating your lookupcombo control.

1. Create a Delphi lookup field in the *TDataSet* associated with your grid, in order to display the field you want from the lookup table. See the Delphi documentation for the steps on creating a *TDataSet* lookup field.
2. Using the *Select Fields Dialog* as defined in chapter 4, perform the following.
 - a) Add the lookupfield to the list of the grid's selected fields, and remove the field that represents the hidden stored value.
 - b) Highlight the name of the lookup field, and then click the Edit Control tab within the Currently Selected Field section of the dialog box.
 - c) Select "Custom Edit" as the Control Type and your *TwwDBLookupCombo* control as the Control Name.
 - d) Click on the OK button of the *Select Fields Dialog* box.

Attaching an icon to the indicator button:

Attaching an icon to the indicator button can simply be done by clicking on the *IndicatorButton* property in the object inspector for the grid, and then load your glyph just like any *TSpeedButton* by using the *Glyph* property.

Edit a LookupField Field in the grid:

Editing a lookupfield can be accomplished with a single line of code. This is accomplished via the following steps:

1. Follow the steps in the Delphi manuals on creating a lookupfield field for your *TwwTable* or *TwwQuery*.
2. Change the grid's *EditCalculated* property to True.

3. Dbl-click the TwwTable or TwwQuery component to bring up the fields editor. Select the lookupfield or linked field in the fields editor and go to the events tab page in the Object Inspector. Now add the following line of code to the TField's *OnChange* event.

```
procedure TForm1.wwTable1cityzipChange(Sender: TField);
begin
    (Sender.Dataset as TwwTable).SetLookupField(Sender);
end;
```

Calling the RecordViewDialog from the grid:

Drop a TwwRecordViewDialog component onto your form and hook up the datasource property to the same datasource that the grid is using. Then from the object inspector, click the grid's IndicatorButton property to bring up the events and properties of the Indicator button. Then add the following code to the IndicatorButton's *OnClick* event.

```
procedure TForm1.wwDBGrid1IndicatorButtonClick(Sender: TObject);
begin
    wwRecordViewDialog1.Execute;
end;
```

Enabling a grid for multi-selection:

The TwwDBGrid can support 2 different types of ways for the end-user to multi-select.

1. If you wish for the end-user to be able to use the mouse, Ctrl-Click, Shift-Click, then set the grid's *Options | dgMultiSelect* to True, followed by updating the *MultiSelectAttributes* to conform to behavior you desire.
2. If you wish to allow the user to select records using a checkbox, then follow these steps.
 - a) Dbl-click the grid's related TDataSet component and define a new Boolean field named "Selected". Your field name must be named "Selected".
 - b) Dbl-click the grid and add the "Selected" field to the grid's list of selected fields.
 - c) Select the "Selected" field and then click on the *Edit Control* tab page, and change its *Control Type* to *CheckBox*.
 - d) Your grid will display the Selected column as a checkbox, and will automatically keep track of which records the user has selected.

Unselect all other records when choosing a new record for multiselection:

Set the *MultiSelectOptions | msoAutoUnselect* to True. Now when you click on a new selection for multiselect without hitting the control key, all previous multiselections will be unselected.

Iterating through the list of selected records in a multi-selection grid:

To iterate through the list of selected records, reference the *SelectedList* property. For example the following code displays the *Last Name* field of all the selected records, one at a time.

```
var i: integer;
begin
    with wwdbgrid1, wwdbgrid1.datasource.dataset do begin
```

```

    DisableControls; {Disable controls to improve performance}
    for i:= 0 to SelectedList.Count-1 do begin
        GotoBookmark(SelectedList.items[i]);
        ShowMessage(FieldByName('Last Name').asString);
    end;
    EnableControls; { Re-enable controls }
end;
end;

```

See also the property *SelectedList* for an example of deleting selected records.

Define the grid's column title alignment, color and font attributes:

Set the *TitleAlignment*, *TitleColor* and *TitleFont* properties as discussed in the *Added properties* section above. If you do not wish for all your title attributes to be the same, then use the *OnCalcTitleAttributes* event to individually control each title.

Example: The following example changes the “Last Name” field title in a grid so that its alignment is centered. Other field titles in the grid will use the alignment as defined by the *TitleAlignment* property.

```

procedure TForm1.wvDBGrid1CalcTitleAttributes(Sender: TObject;
    AFieldName: String; AFont: TFont; ABrush: TBrush;
    var ATitleAlignment: TAlignment);
begin
    if FieldName='Last Name' then ATitleAlignment:= taCenter;
end;

```

Color alternating rows in the grid with different colors:

To color alternating rows in a grid use the *PaintOptions.AlternatingRowColor* property.

Word-wrap text in the grid:

Set the *Options | WordWrap* property to True, and then change the grid's *RowHeightPercent* so multiple lines will be displayed for each row.

Convert carriage return to tabs in the grid:

Set the *KeyOptions | dgEnterToTab* property to True

Disable keyboard shortcuts for inserting and deleting into the grid:

Set the *KeyOptions | dgAllowDelete* and *KeyOptions | dgAllowInsert* properties to False.

Define the number of fixed columns:

Set the *FixedCols* property to the number of columns, from the left-hand side of the grid (not including the record indicator column), that you want to be “fixed in-place” or non-scrollable.

Define how memo fields are displayed:

Set the sub-properties of the *MemoAttributes* property. Refer to the *Added properties* section above for details about these sub-properties.

Define and display a related field from another table in the grid:

See the Delphi documentation on creating a lookup field

Updating other fields based on the contents of an embedded TwwDBLookupCombo component:

Use the *OnCloseUp* event of the TwwDBLookupCombo component. Refer to the TwwDBLookupCombo component description for details about using this event.

Displaying a TGraphic field as a Bitmap in the grid:

To display a Graphic field in your database you can attach a Delphi TDBImage as a custom control of the column. In addition, you need to check the *Control Always Paints* checkbox so that the image is painted in the grid for every row.

Alternatively, you can set the control type to a bitmap by following these steps:

Double-click the grid component to display InfoPower's Select Fields dialog box. Make sure the column/field you want to change is listed in the Selected Fields listbox. If the field is not in the selected fields listbox, then click on the *Add Fields* button add the field.

1. In the Selected Fields listbox, highlight the name of the field you want to change by clicking on it.
2. Click the Edit Control tab within the Currently Selected Field section of the dialog box to display the field's currently selected edit control.
3. Click the drop-down button of the Control Type field and click on *Bitmap*
4. Enter the Bitmap Scaling and Raster Operation you wish to use.
5. Click the OK button of the Select Fields dialog box.

This bitmap field will now display in the grid.

Display an image from an ImageList

Quite often you may want to use some type of icon or bitmap to give some visual clue about a record to the user. However it would not be desirable if you had to add another field in your table and store the bitmap for every record. InfoPower allows you to store these types of icons or bitmaps in an ImageList, and display them in the grid. This is accomplished via the following steps:

1. Drop a Delphi TImageList component into your form, and assign the grid's ImageList property to this component.
2. Dbl-click the TImageList to add the desired images to the ImageList. See the Delphi documentation for details on this component.
3. Dbl-click the Grid and select the field that is to represent the index into the ImageList. This field must be of type *TIntegerField*. (If no physical field represents the

ImageIndex, then create a calculated TIntegerField in your dataset and use the TDataSet's OnCalcFields to assign it the desired index into the ImageList).

4. Click the Edit Control tab within the Currently Selected Field section of the dialog box to display the field's currently selected edit control.
5. Click the drop-down button of the Control Type field and click on ImageIndex.
6. Check the "Shrink To Fit" checkbox if you wish the image to shrink to fit into the grid's cell.
7. Click the OK button of the Select Fields dialog box.

The image from the ImageList will now display in the grid. If you are attached to a calculated field, then the image will not appear until you execute the program.

Editing a memo field within the grid:

Editing a memo while in a grid cell can be accomplished via the following steps:

1. Drop a TwwDBEdit component on your form and change the *WordWrap* property to True and the *WantReturns* to True. If your memo text is long, then you may also wish to set the *ShowVertScrollBar* to True.
2. Changing the grid's *RowHeightPercent* property to something like 200 or more percent.
3. Double-click the grid component to display InfoPower's Select Fields dialog box. Make sure the memo field you want is listed in the Selected Fields listbox. If the field is not in the selected fields listbox, then click on the *Add Fields* button add the field.
4. In the Selected Fields listbox, highlight the name of the memo field.
5. Click the Edit Control tab within the Currently Selected Field section of the dialog box to display the field's currently selected edit control.
6. Click the drop-down button of the Control Type field and click on CustomEdit
7. Then click on the Control Name dropdown listbox and choose the *wwdbedit1* control that you wish to be bound to this field.
8. Click the OK button of the Select Fields dialog box.

Use multiple grids on one dataset each displaying different fields of one dataset:

This can be accomplished by setting the UseTFields property to false on each of the grids that is attached to the same dataset. Then you can just double click on each grid and add the fields that you want visible in each of the grids.

Detecting when you move to a new row or a new cell in the grid

Use the *OnRowChanged* or the *OnCellChanged* events of the grid.

Allow the user to dbl-click a column header to resize the column.

Set Options | dgDbClickColSizing to True

Tips

- ◆ When defining groups and subgroups in the titles of the TwwDBGrid, make certain that the subgrouped fields are right next to each other and have the same groupname.
- ◆ Many of the grid's capabilities are not supported unless UseTFields is False. We recommend that you set this property to false unless you absolutely want TField changes to be immediately reflected in the grid.
- ◆ To modify the properties of individual fields displayed in a grid, such as value alignment, use the Object Inspector. If the field does not appear in the Object Inspector, first make sure it's selected via Delphi's Fields editor window (double-click the TwwTable component and use the Add option as necessary). By default Delphi selects all fields in a table for retrieval, but you are not allowed to edit the attributes of individual fields unless they are physically listed in the Fields editor window. To manually add all fields to the listbox in the Fields window, click the Add button of the Fields editor, make sure all fields are highlighted and then click the OK button. You will now be able to select an individual field in the Object Inspector and modify its properties.
- ◆ When creating InfoPower components that are embedded in a TwwDBGrid component, resize the Combo component on your form to display only a single character (shrink the component from the left-hand side). These very small sized components are a visual reminder that the component is used in a grid instead of on the form.
- ◆ When you are manipulating many TField properties your property assignments will execute considerably faster if you call the related dataset's *DisableControls* method prior to doing the property assignments. After completing your property changes, don't forget to re-enable the controls bound to the dataset by calling *EnableControls*.

For example:

```
with wwtable1 do
begin
  DisableControls; {Display screen updates for wwtable1}

  {Make all your changes here}
  FieldByName('Field1').DisplayLabel := 'Field Title 1';
  FieldByName('Field1').Index := 1;
  FieldByName('Field2').DisplayLabel := 'Field Title 2';
  FieldByName('Field2').Index := 2;
  FieldByName('Field3').DisplayLabel := 'Field Title 3';
```

```
FieldByName('Field3').Index := 3;  
  EnableControls; {show changes}  
end;
```


TwwDBLookupCombo



The TwwDBLookupCombo visual interface component provides your end-users with the ability to enter, edit or select a value for a field from a drop-down list of values that is populated from a second lookup table.



Figure 5.9 - The TwwDBLookupCombo component.

InfoPower gives you the most flexible component for selecting entries from a lookup table. Here is some of what this powerful component can do.

- **New – LookupCombo as Navigating Tool**

The LookupCombo can now be used as a navigating drop-down control, without requiring code to initialize its display and synchronization. When the user moves to a new record, the lookupcombo's display automatically reflects the active table record. Set the Navigator property to True to use the control as a navigator instead of a lookup and fill control.

- **Quicken style incremental searching:** All of InfoPower's lookup components support the 'Quicken' style display of the matching value, by simultaneously searching and displaying the matching text in the search control
- **Support for many dataset types:** Fill a drop-down list with a table, query, QBE, ClientDataSet, and even parameterized queries.
- **End-user usability enhancements:** Smart properties to auto-drop down the list upon a valid keystroke as well as a convenient way of clearing the Lookup Combo's Text;
- **Flexible control over the appearance of the drop-down list:** Select any number of fields to be displayed in the drop-down list along with defining their display width and optional titles.
- **Embed images in the drop-down list:** Define a field in the drop-down list as an index into an Image list, and the control will automatically display the images.
- **Embed into InfoPower's Grid, Inspector, or RecordView components:** The component can be used in a TwwDBGrid component to replace any multiple-choice type of field in the grid, giving your end-users sophisticated lookup and fill capabilities within the grid or record-view components.
- **Sorting flexibility:** The values in the drop-down list can be sorted in the order of the first field you select to be displayed, if it's a secondary index field, instead of being sorted in primary key order.
- **Use unbound or bound:** The component does not have to be bound, or assigned, to a table's field (DataField and DataSource properties) which gives you greater flexibility in using this LookupCombo for general tasks where a source table is not involved.

- **Supports Transparency and custom framing :** The control can now be displayed transparently when it does not have the focus. In addition you can customize the border and of the control.
- **Improved icon support:** You can attach your own custom glyph to the control, and the glyph can be displayed in a flat or transparent style.

The first field displayed in the Selected Fields list of the Select Fields dialog box is the field displayed in the component and provides the end-user with incremental search capability. For performance reasons, it is best that an index for this field exist in the *LookupTable*. (See the *How to* section of this component for a description of overriding the default index.).

Set the *LookupTable* property to the dataset component that contains the list of lookup values.

Set the *LookupField* property to the field being read-in from the lookup table .

Ancestor

TwwDBCcustomLookupCombo

Added Properties

AllowClearKey

When the ComboBox style is set to *csDropDownList*, the user is not able to clear their selection. The *AllowClearKey* property when set to True, gives the user a convenient way to clear the combos current selection simply by entering either the or <BACKSPACE> character.

Data Type: Boolean

AutoDropDown

When True, the lookup list drops down automatically when a keystroke is entered. The default value is False.

Data Type: Boolean

ButtonEffects

See the topic “Key properties for enabling custom button effects in the edit controls” in chapter 4 for information on this property.

Data Type: TwwButtonEffects

ButtonGlyph

This property defines the custom bitmap used for the icon in the control when *ButtonStyle* is set to *cbsCustom*.



Data Type: TBitmap

ButtonStyle

This property defines the icon used for this component. If the property *ShowButton* is *False*, then this property is ignored.

Data Type: *TwwComboButtonStyle*

Valid Values: *cbsEllipsis*, *cbsDownArrow*, *cbsCustom*

<i>cbsDownArrow</i>	The  bitmap is displayed
<i>cbsEllipsis</i>	The  bitmap is displayed
<i>cbsCustom</i>	The icon defined by the <i>ButtonGlyph</i> property.

ButtonWidth

This property defines the width of the icon for the control. You may wish to set this property if your custom bitmap assigned to the *ButtonGlyph* property is larger than the default button width. This property defaults to 0, which indicates to the control to compute the button width based on the system settings..

Data Type: *Integer*

DisableThemes

If your project has enabled XP themes but you do not wish for this control to be theme-enabled, then set this property to *False*.

DropDownAlignment

This property defines which edge of the component the drop-down list should be aligned with. *taLeftJustify* draws the drop-down list aligned with the left edge of the component (grows to the right). *taRightJustify* draws the drop-down list aligned with the right edge of the component (grows to the left). The default is *taLeftJustify*.

Data Type: *Constant*

Valid Values: *taLeftJustify*, *taRightJustify*

Frame

See the topic “Key properties and events for custom framing” in chapter 4 for information on this property.

Data Type: *TwwEditFrame*

Grid (Runtime Only)

This property gives you control over drop-down list properties, such as color, font, etc, through the new public property *Grid*.

Data Type: *TwwPopupGrid*

Example: To change the color of the drop-down list to be yellow you can place the following code in the control's *OnDropDown* event.

```
wwDBLookupCombo1.Grid.color:= clYellow;
```

ImageList

See the TwwDBGrid *ImageList* property

LookupField

This property defines which fields of the lookup table are used when looking up the value in the lookup table. It was modified to support multiple lookup fields when performing a lookup against a multi-field index. Use semicolons to separate field names. For example, if your lookup table index contains the two fields LastName and FirstName, the value of this property would be “LastName;FirstName”.

InfoPower has the following limitations when specifying multiple fields with the LookupField property:

- It supports up to 3 fields. If you require more than 3 lookup fields then use the TwwLookupDialog component in conjunction with a TwwDBComboDlg and execute this dialog in TwwDBComboDlg's *OnCustomDialog* event. (See TwwLookupDialog for more details.)
- Your lookup field names must exist in both the LookupTable and the DataSource table.

LookupTable

This property defines the TDataSet component to be used for populating the grid. The default value is blank.

Data Type: TDataSet

Valid Values: Valid TwwTable, TwwQuery, TwwQBE, TwwStoredProc, or TwwClientDataSet component name

LookupValue

Internal stored value. This is the same as the displayed text if your *LookupField* and your first selected field are the same. You may wish to explicitly set this property and the text if you are initializing an unbound TwwDBLookupCombo.

Data Type: String

Navigator

Set the Navigator property to True to use the control as a navigator instead of a lookup and fill control. When used as a navigator, the LookupCombo handles matches the display of the control to reflect the current record that the lookuptable is pointing to. When the user drops down the list and selects a record, it does not perform a lookup and fill operation, but instead moves the lookuptable to the record that was selected. When the user moves to a new record through another control (such as a navigator) the lookupcombo's display automatically reflects the active table record.

OrderByDisplay

When True, this property will automatically change the LookupTable's *IndexName* so that the displayed field is the first field of the index. By changing the index, the drop-down list can be

viewed in the order of the displayed field instead of some hidden field which may be useless to the end-user. This property has no effect unless you are using a TwwTable component for your lookup. The default for this property is True.

Data Type: Boolean

Picture

Picture mask specification. Please reference chapter 4, *Selecting a Picture Mask* for details on this property.

SearchDelay

This property controls how many milliseconds to wait before beginning the search for the user's entered text. The purpose of this property is to reduce the number of searches that are performed as the user enters characters. Setting this to a larger value may improve your performance as fewer searches will need to be performed. Setting this to a smaller value will cause the search to begin more quickly. This property defaults to 0, which tells the control to determine the best delay. Currently 333 milliseconds is used by the control for the delay.

Data Type: Integer

SearchField

This is a runtime only property. When using a multi-field index or a dBASE expression index, you can incrementally search on the 2nd or 3rd field (instead of the first) by setting this property to the field you want to search. By default the first field will be searched. To set this property, add code to the *OnDropDown* event that sets the table index, and then sets the property *SearchField* for the TwwDBLookupCombo.

Note: This field is *required* for dBASE expression indexes, and must be set to one of the fields that compose the expression index.

Data Type: String

Valid Values: Valid field name

Example: The following code sets the index to the dBASE expression index, and then specifies which field in the expression index should be searched incrementally:

```
Procedure TForm1.wwDBComboBox1DropDown(Sender: TObject);
begin
  wwTable1.IndexName := 'MyExprIndex';
  wwLookupCombo1.SearchField := 'MySearchField';
end;
```

Selected

Clicking the “...” button or double-clicking the TwwDBLookupCombo component displays the Select Fields dialog box. This dialog box allows you to select the fields you want displayed in the drop-down list, the order in which they are listed, their titles, display widths, control types and link information. (See *Using the Select Fields Dialog Box* at the beginning of Chapter 4.) The default value is *no* fields selected.

Data Type: (Internal to InfoPower)

SeqSearchOptions

This property controls how the component incrementally searches the LookupTable when an index is not available. When used against a TwwTable this property is ignored and the case sensitivity of the index is instead used.

Data Type: TSet

Valid Values: ssoEnabled, ssoCaseSensitive

ssoEnabled When True, incremental searching is supported. Incremental searching is done by a sequential search through the result set.

ssoCaseSensitive When True, incremental searching considers case sensitivity when performing an incremental search. This property has no effect if ssoEnabled is False.

ShowButton

When this property is False, the combo's bitmap button is not shown. The default value is True.

Data Type: Boolean

ShowMatchText

When this property is True this combo will have Quicken Style incremental searching by simultaneously searching and displaying the matching text in the search control. The default value is False.

Data Type: Boolean

Text

Currently displayed text in the control

Data Type: String

UseTFields

When the UseTFields property is set to true the *Selected* properties Display settings information will be stored and retrieved from the *LookupTable* dataset. When it is set to False the *Selected* properties Display settings information is stored with the TwwDBLookupCombo. The default is True.

Data Type: Boolean

Modified properties

DataField

Optional. Can be left blank in conjunction with a blank DataSource in order to create an unbound component.

DataSource

Optional. Can be left blank in conjunction with a blank DataField in order to create an un-bound component.

LookupDisplay

Replaced by the Select Fields dialog box (double-click the component). The first selected field becomes the field that the combo displays when it is not dropped down. All the selected fields are displayed when the combo is dropped down.

LookupSource

Replaced by LookupTable.

Required property assignments

LookupField, LookupTable and Selected.

Added Events

OnCloseUp

Use this event to perform your own actions when the drop-down list closes (immediately after the user makes a selection). This allows you to fill-in one or more related fields. For example, if the TwwDBLookupCombo component is used to lookup a part number for an invoice, in this event you could then acquire and fill-in other related data, such as the unit price, manufacturer name and part description.

Parameters

<i>Sender</i> : TObject	TwwDBLookupCombo that is being closed up
<i>LookupTable</i> : TDataSet	DataSet being looked up
<i>FillTable</i> : TDataSet	DataSet that is being filled with the lookup value.
<i>Modified</i> : Boolean	True, if user has selected a value. False, if user has entered <ESC>.

OnDropDown

Use this event to perform your own actions just before the drop-down list is displayed to the end-user. For example, you could activate a table filter against the *LookupTable* to limit the records displayed in the drop-down list.

Parameters

<i>Sender</i> : TObject	TwwDBLookupCombo that is being dropped down.
-------------------------	--

OnNotInList

Use this event if you wish to perform your own actions when the user types in a value that is not in the lookup table list.

Parameters

<i>Sender</i> : TObject	TwwDBLookupCombo that is being edited
<i>LookupTable</i> : TDataSet	DataSet being looked up
<i>NewValue</i> : String	Value that user has typed in which is not in the table list.
<i>Accept</i> : Boolean	Set this variable to True to accept the entry, and False otherwise.

Example: The following example adds a new record to the lookup table. If more than one field exists in your lookup table, then you will need to provide a way for the end-user to enter the additional field values. In this example there are two fields in the lookup table, zip and city. The user is typing in the city name. The new zip value needs to be assigned by your own custom form or dialog.

```
procedure TLookupForm.wwDBLookupCombo1NotInList(Sender: TObject;
LookupTable: TDataSet; NewValue: String; var Accept: Boolean);
begin
  { Your custom form could do the following instead of here }
  LookupTable.Insert;
  LookupTable.FieldByName('City').AsString:= NewValue;
  LookupTable.FieldByName('Zip').AsString:= NewZipValue;
  LookupTable.Post;
  Accept:= True;

  { Refresh combo }
  with (Sender as TwwDBLookupCombo) do
  begin
    DataSource.DataSet.FieldByName(DataField).AsString:= NewZipValue;
    LookupValue:= NewZipValue
  end;
end;
```

OnPerformCustomSearch

When using a large lookup table from a remote server, the performance of the lookup combo's incremental searching can significantly degrade. To resolve this issue, InfoPower adds a new event where you can control the specific action that takes place after the user types a character, or when the control needs to look up a value. In particular the custom action can update the query to only return the records that you are interested in. When using this event, your code is responsible for manipulating the lookup table based on the parameter values passed in.

Parameters

<i>Sender</i> : TObject	TwwDBLookupCombo that is being edited
<i>LookupTable</i> : TDataSet	DataSet being looked up
<i>SearchField</i> : String	Field to search

<i>SearchValue</i> : String	Value of field to be searched in the lookup table.
<i>PerformLookup</i> : Boolean	If this value is true, then your code in the event should search the lookup table for the specific value indicated by <i>SearchValue</i> . If this value is false, then your code should find a partial match with starting with the string indicated by <i>SearchValue</i> .
<i>Var Found</i> : Boolean	Set to true to indicate that a matching record was found.

Example: The following example uses the event to modify the sql in the lookup table, and reactivates it to perform the custom search. Only 1 record is fetched from the remote server when doing the lookup, and only records matching the entered text are returned during the incremental searching. The lookup field in this example is 'zip'.

```

procedure TCustomComboForm.wwDBLookupCombo1PerformCustomSearch (
  Sender: TObject;
  LookupTable: TDataSet; SearchField, SearchValue: String;
  PerformLookup: Boolean; var Found: Boolean);
const dbl = '';
var q: TQuery;
begin
  q:= TQuery(LookupTable);
  q.active:= false;
  SearchValue:= AnsiUppercase(SearchValue);
  if PerformLookup then // Find exact match
  begin
    q.sql.clear;
    q.sql.add('Select * from ip4zip');
    q.sql.add('where ip4zip."zip" = ' + dbl + SearchValue + dbl);
  end
  else begin // Find partial match
    q.sql.clear;
    q.sql.add('Select * from ip4zip');
    q.sql.add('where ip4zip."zip" like ' + dbl + SearchValue + '%' + dbl);
  end;
  q.active:= true;
  found:= not q.eof;
end;

```

Added Methods

DropDown

Call this method to programmatically dropdown the list.

PerformSearch

Call this method to immediately perform an incremental search on the LookupTable with the current text value. In cases where you apply a filter in your *OnDropDown* event, you need to explicitly call *PerformSearch* if you are using *AutoDropDown*. The LookupCombo may have already completed its search prior to the filter being applied.

RefreshDisplay

Call this method to refresh the lookupcombo's display based on changes in the lookuptable's data.

How To

Update other fields based on the contents of a wwDBLookupCombo component:

Attach code to the *OnCloseUp* event that sets the *text* property of other fields on the form. For example, you have an invoice line item entry form that is bound to a table named *LineItem* that contains fields named *PartNo*, *PartDesc*, *UnitCost* and *Qty*. Your *LookupTable* name is *Parts*, contains fields named *PartNo*, *PartDesc* and *UnitCost*, and has a *DataField* property value of *PartNo*. The user has already entered a *Qty* value and has just selected the *PartNo* from the *Parts* table's lookup combo drop-down list (*PartNo* is not set because the *LookupCombo* is defined to fill that field)...

```
if modified then begin {only execute when PartNo is changed}
  LineItem.fieldByName('PartDesc').text :=
    Parts.fieldByName('PartDesc').text;
  LineItem.fieldByName('UnitCost').text :=
    Parts.fieldByName('UnitCost').text;
end;
```

Override the default LookupTable index:

Add the following line of code to the *OnDropDown* event to change the default active index (first field listed in the Selected Fields list box) used in the *LookupTable* to 'iLast':

```
with (Sender as TwwDBLookupCombo) do
  (LookupTable as TTable).IndexName := 'iLast';
```

Fill the drop-down list from a Query or QBE result.

The following example queries the *IP4ZIP* table, and retrieves all distinct states to fill into a *TwwDBLookupCombo*'s list. The table used by the *TwwQuery* in this example is *IP4ZIP.DB* and is located in the *InfoDemo5 DatabaseName* alias.

1. Add a new *TwwQuery* component to your form and set the following properties:

```
DatabaseName = InfoDemo5
Name = CustomerQuery
SQL = Select Distinct State from IP4ZIP
Active = True
```

2. Add a new *TwwDBLookupCombo* component to your form and set the following properties

```
LookupTable = CustomerQuery
LookupField = State
```

3. Save your project, run the program and click on the drop-down icon for your LookupCombo. If all went well, you should see one entry for each state in the zip code table.

Tips

- ◆ Remember to select the fields you want displayed in the drop-down list by double-clicking the TwwDBLookupCombo component or by clicking the “...” button in the *Selected* property.
- ◆ To display column titles, lines or row lines in the drop-down list, define the sub-options within the *Options* property.
- ◆ If you have two TwwDBLookupCombo components on the same form that access the same physical lookup table, they must use two different TwwTable components. This is necessary in order to keep the component’s use of the table indexes from conflicting with each other.
- ◆ You can press the **Alt+down** keyboard keys when the component has focus to activate the drop-down display.

TwwDBLookupComboDlg



Similar in functionality to the TwwDBLookupCombo component, InfoPower's TwwDBLookupComboDlg component provides the services of the TwwDBLookupCombo component in a dialog box. This component provides your end-users with the ability to enter, edit or select a value for a field from a list of values that is populated from a second *lookup* table. When the user clicks the "..." button on the visual component, the value selection list is displayed via a grid embedded in a dialog box, instead of in the usual drop-down list.

The dialog box contains a developer-controlled grid along with a search criteria edit box and an optional table index selection combo box. You can enable up to two optional developer-controlled buttons in this dialog box and define what actions take place when the user clicks on either button.

The TwwDBLookupComboDlg provides all the properties, events, and methods of the TwwDBLookupCombo with the exception of *DropDownAlignment*, *DropDownCount* and *DropDownWidth*. It additionally provides for the properties, events, and methods defined in the following pages.



Figure 5.10 - The TwwDBLookupComboDlg component.

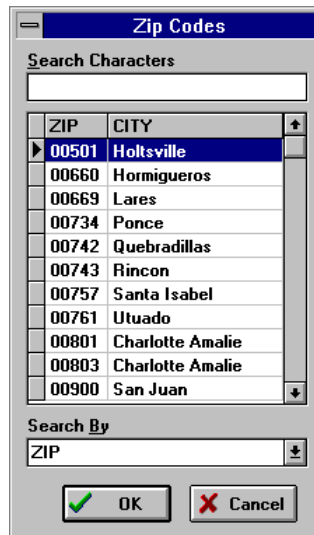


Figure 5.11 - An example dialog box portion of the TwwDBLookupComboDlg component.

Ancestor

TwwDBCUSTOMLOOKUPCOMBO

Required supporting components

TDataSource

Added Properties

AllowClearKey

When the ComboBox style is set to csDropDownList, the user is not able to clear their selection. The *AllowClearKey* property when set to True, gives the user a convenient way to clear the combos current selection simply by entering either the or <BACKSPACE> character.

Data Type: Boolean

AutoDropDown

When True, the lookup list drops down automatically when a keystroke is entered. The default value is False.

Data Type: Boolean

ButtonEffects, ButtonGlyph, ButtonStyle, ButtonWidth

See the TwwDBLookupCombo for a description of these properties.

Caption

This property contains a text value that is displayed in the editor window's title bar. The default value is "Lookup".

Data Type: String

DataField

Optional. Can be left blank in conjunction with a blank *DataSource* in order to create an unbound component.

DataSource

Optional. Can be left blank in conjunction with a blank *DataField* in order to create an unbound component.

Frame

See the topic "Key properties and events for custom framing" in chapter 4 for information on this property.

GridColor

This property defines the background color of the grid. The default value is clWhite. (When the first column of a grid is fixed, its colors are the same colors used for the grid's column titles as defined in the *TitleColor* property.)

Data Type: TColor

GridOptions

This property contains a set of standard Delphi grid options.

Data Type: TSet()

Valid Values: Valid Delphi grid options

GridTitleAlignment

Determines the text alignment of titles in popup-dialog's grid. The default value is taLeftJustify.

Valid Values: taCenter, taLeftJustify *or* taRightJustify

LookupField

See TwwDBLookupCombo *LookupField* property.

LookupTable

See TwwDBLookupCombo for description of *LookupTable*.

LookupValue

See TwwDBLookupCombo for description of *LookupValue*.

MaxHeight

Defines the maximum Height of the grid in the related dialog. Use this property to control the height of the popup-dialog. The default value for a standard VGA display (640 x 480) is 209.

Data Type: Integer (Positive)

MaxWidth

This property defines how wide the dialog box is allowed to grow, in pixels. The default value is 0, which allows the dialog box to grow to the entire width of the screen.

Data Type: Integer

Valid Values: Depends on your screen's display resolution

Options

This property contains a set of Boolean values that control the appearance of the dialog box, as described below. The default values are opShowOKCancel and opShowSearchBy.

Data Type: TSet()

Valid Values: opShowOKCancel, opShowSearchBy, opGroupControls, opFixFirstColumn and opShowStatusBar (described below)

opShowOKCancel When True, the OK and Cancel buttons are displayed in the dialog box. When False these buttons are not displayed—OK can be simulated by double-clicking an entry or by selecting it and then pressing the Enter key. Cancel can be simulated by pressing the Esc key or by closing the dialog box window. The default is True.

<i>opShowSearchBy</i>	When True, the Search By drop-down control is displayed in the dialog box. When False, this control is not visible. The default value is True.
<i>opGroupControls</i>	When True, the Search Characters and Search By controls are displayed side-by-side above the grid. When False, the Search Characters control is displayed above the grid and the Search By control is displayed below the grid. The default value is False.
<i>opFixFirstColumn</i>	When True, the left-most column of the grid is fixed (non-scrollable). When False, the left-most column can be scrolled out of view. The default value is True.
<i>opShowStatusBar</i>	<i>For use with Paradox tables only.</i> When True, a status bar is added to the dialog box that displays the table name, current record number and total number of records in the table.

OrderByDisplay

When True, this property will automatically change the LookupTable's *IndexName* so that the displayed field is the first field of the index. By changing the index, the drop-down list can be viewed in the order of the displayed field instead of some hidden field which may be useless to the end-user. This property has no effect unless you are using a TwWTable component for your lookup. The default for this property is True.

Data Type: Boolean

SearchDelay

See the TwWDBLookupCombo *SearchDelay* property.

SeqSearchOption

See TwWDBLookupCombo *SeqSearchOption* property.

ShowButton

When False, the drop-down button is not shown when the program executes. The default value is True.

Data Type: Boolean

ShowMatchText

When this property is True this combo will have Quicken Style incremental searching and highlight the text that is the closest match. The default value is False.

Data Type: Boolean

UserButton1Caption

When you want to display this button on the dialog box, enter the caption text for the button here and then add code to the *OnUserButton1Click* event. The default value is blank.

Data Type: String

UserButton2Caption

When you want to display this button on the dialog box, enter the caption text for the button here and then add code to the *OnUserButton2Click* event. The default value is blank.

Data Type: String

UseTFields

When the UseTFields property is set to true the *Selected* properties display settings are stored and retrieved from the TFields of the *LookupTable* dataset. When it is set to False the *Selected* properties display settings are stored with the *TwwDBLookupCombo*. The default is True. Set this property to False if you have multiple controls attached to the same dataset and each has different settings for its fields.

Data Type: Boolean

Removed properties

The following properties were removed: *DropDownAlignment*, *DropDownCount* and *DropDownWidth*.

Required property assignments

DataField, DataSource, LookupField, LookupTable and Selected.

Added Events

Some of the following events pass a handle to the form containing all of the components of the dialog. To see what objects are contained within this editing form, open up *wwidlg.pas* in the InfoPower source sub-directory. If you do not have the source code version of InfoPower, then perform the steps in Chapter 4's topic "Determining the object names of the controls contained in an InfoPower dialog" on the *wwidlg.dfm* file contained in the InfoPower lib directory.

If you want to customize any of the objects contained by the form you can use the *OnInitDialog* event. However if all you are trying to do is to change the labels and hints, then use the *TwwIntl | SearchDialog* property.

OnCloseDialog

This event allows you to perform any custom action before the dialog is actually closed.

OnInitDialog

Allows you to customize every aspect of the dialog box or perform some action during the initialization of the dialog box. When using this event, your code must reference **wwidlg** in your source file's *Uses* clause. This gives you access to all the components in the dialog. For example, you can modify the grid's properties, define custom events, etc.

Example: The following code tells the first user-defined button to show a hint when the user moves the mouse pointer over the button:

```
procedure TForm1.wwDBLookupComboDlg1OnInitDialog(
```



```

    Dialog: TwwLookupDlg);
begin
    Dialog.UserButton1.Hint := 'Hint for user button 1';
    Dialog.UserButton1.ShowHint := True;
end;

```

OnPerformCustomSearch

When using a large lookuptable from a remote server, the performance of the lookupcombo's incremental searching can significantly degrade. To resolve this issue, InfoPower adds a new event where you can control the specific action that takes place after the user types a character, or when the control needs to look up a value. In particular the custom action can update the query to only return the records that you are interested in. When using this event, your code is responsible for manipulating the lookuptable based on the parameter values passed in. This event is also fired during incremental searching within the popup dialog. See the TwwDBLookupCombo OnPerformCustomSearch event for a description of the events parameters.

OnUserButton1Click

When you want to display developer-defined button #1 on the dialog box, enter the caption text for the button in the *UserButton1Caption* property and then add code to this event that will be executed when the end-user clicks the button.

Tip: If you wish for this dialog to immediately close after executing your code, assign the *ModalResult* property of the dialog. The Sender parameter is cast to a TForm to get a handle to the actual dialog on the screen.

```

procedure TForm1.wwDBLookupComboDlg1UserButton1Click(
    Sender: TObject; LookupTable: TDataSet);
begin
    (Sender as TForm).ModalResult := mrOK;
end;

```

OnUserButton2Click

When you want to display developer-defined button #2 on the dialog box, enter the caption text for the button in the *UserButton2Caption* property and then add code to this event that will be executed when the end-user clicks the button.

How-to

Change the default position of the pop-up dialog.:

The following code attached to the OnInitDialog event will change the default position of the pop-up dialog to be (left=10, top=10).

```

procedure TForm1.wwDBLookupComboDlg1InitDialog(
    Dialog: TwwLookupDlg);
begin
    Dialog.Left:= 10;
    Dialog.Top:= 10;
end;

```

Make the dialog list show only records that meet certain criteria:

Attach code to the *OnDropDown* event that creates and activates a filter on the lookup table. See the How-to section of the `wwDBLookupCombo` component for an example of how to do this.

Update other fields based on the contents of a `wwDBLookupComboDlg` component:

Attach code to the *OnCloseUp* event that sets the *text* property of other fields on the form. See the How-to section of the `wwDBLookupCombo` component for an example of how to do this.

Tips

To activate the drop-down portion of this component via the keyboard, press the **Alt+down cursor arrow** keys when the component has focus.

TwwDBMonthCalendar



InfoPower's MonthCalendar control allows you to display a calendar to the end-user in a variety of formats.

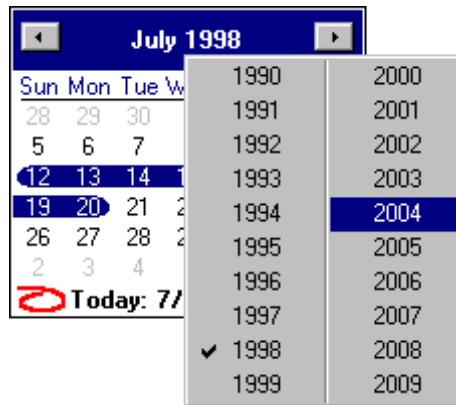


Figure 5.12 - A TwwDBMonthCalendar with the year pop-up menu

- Display one or more months of a year in a single InfoPower calendar control.
- Hide/Show the **Week Numbers**, **Today String**, and **Today Circle** in the Drop Down Month Calendar.
- Supports configurable pop-up menus when clicking on either the year or the month
- Code-based determination of which dates should be displayed in bold. For instance boldface weekends, holidays, and paydays.
- Change the colors and fonts of the calendar.
- Use with or without a database.

Ancestor

TWinControl

└ TwwMonthCalendar

└ TwwDBCUSTOMMonthCalendar

Required supporting components

None

Added Properties

CalColors

This property defines the colors the month calendar uses for its display.

MonthBackColor	Background color of the days in the calendar. Defaults to clWhite.
TextColor	Color used to display text within the month. Defaults to clWindowText.
TitleBackColor	Background Color displayed in the calendar's title. Defaults to clActiveCaption.
TitleTextColor	Color used to display text within the calendar's title. Defaults to clWhite.
TrailingTextColor	Color used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month calendar. Defaults to clInactiveCaption.

DataField

This property defines the name of the field you want to bind the MonthCalendar to. The default value is blank (unbound).

Data Type: String

Valid Values: Valid field name

DataSource

This property defines the name of the TDataSource you want to bind the MonthCalendar to. The default value is blank (unbound).

Data Type: TDataSource

Valid Values: Valid DataSource component name

Date

This property defines the date the month calendar initially displays as selected. This property is ignored if the component is bound to a database field, as the date will then originate from the value of the database field.

Data Type: TDateTime

EndDate

Run-time only. This property is used with *Options | mdoMultiSelect*. After the end-user has selected a range of dates, this property is updated to contain the last date in the selected range. See also the properties *MaxSelectCount* and *MinDate*.

Data Type: TDateTime

FirstDayOfWeek

Set this property to change the first day of week in the calendar. Default is *wwdowLocaleDefault*, where the Windows operating system determines the first day of the week.

Date Type: TwwCalDayOfWeek

Valid Values: wwdowMonday, wwdowTuesday, wwdowWednesday, wwdowThursday, wwdowFriday, wwdowSaturday, wwdowSunday, wwdowLocaleDefault

MaxDate

This property defines the maximum allowable date that the month calendar will allow the end-user to select. The default is blank which means that the upper range of the calendar is not restricted.

Data Type: TDateTime

MaxSelectCount

When Options | mdoMultiSelect is True, this property defines the maximum number of consecutive days that can be selected.

Data Type: Integer

MinDate

This property defines the minimum allowable date that the month calendar will allow the end-user to select. The default is blank which means that the lower range of the calendar is not restricted. **Note:** The calendar does not support dates less than the year 1900.

Data Type: TDateTime

Options

This property defines a set of options for the month calendar control.

Data Type: Set of TwwMonthOption;

Valid Values: mdoDayState, mdoWeekNumbers, mdoNoToday, mdoNoTodayCircle, mdoMultiSelect

<i>mdoDayState</i>	When True, the <i>OnCalcBoldDay</i> event is fired.
<i>mdoWeekNumbers</i>	When True, week numbers are displayed at the far left column of the calendar
<i>mdoNoToday</i>	When True, the month calendar control will not display the "today" date at the bottom of the control.
<i>mdoNoTodayCircle</i>	When True, the month calendar control will not circle the "today" date.
<i>mdoMultiSelect</i>	When True, the month calendar will allow the user to select a range of dates within the control. By default, the maximum range is one week. You can change the maximum range that can be selected by using the <i>MaxSelectCount</i> property.

PopupYearOptions

- NumberColumns** Set this property to change the number of columns in the pop-up year menu. The pop-up menu appears when the user clicks on the year in the calendar. This property defaults to 2
- StartYear** Set this property to change the starting year in the pop-up year menu
- YearsPerColumn** Change this property to change the number of year per column in the pop-up year menu. It defaults to 10
- ShowEditYear** When True, the pop-up year menu for the calendar control has a way of entering the year with an edit box.

StartDate

Run-time only. This property is used with *Options | mdoMultiSelect*. After the end-user has selected a range of dates, this property is updated to contain the first date in the selected range. See also the properties *MaxSelectCount* and *MinDate*.

Data Type: TDateTime

Time

This property defines the internal time that the month calendar stores. This is never displayed to the end-user, but is used internally when updating a database field

Data Type: TDateTime

Valid Values: Valid TDateTime value

Required property assignments

None

Added or modified events

OnCalcBoldDay

This event allows you to calculate which dates should be displayed in bold. This event is only called if *Options | mdoDayState* is True.

The parameters for this event are as follows.

- | | |
|----------------------------------|--|
| <i>Sender:</i> TObject | Calendar control to compute attributes |
| <i>ADate:</i> TDate | Date to evaluate |
| <i>Month, Day, Year:</i> Integer | Month, Day, Year to evaluate |
| <i>Accept:</i> Boolean | Set to True to boldface the date in the calendar |

Example: The following example boldfaces all weekend days in the calendar.

```
procedure TMainDemo.wvDBMonthCalendar1CalcBoldDay(Sender: TObject;
  ADate: TDate; Month, Day, Year: Integer; var Accept: Boolean);
begin
  if DayOfWeek(ADate)=1 then Accept := True;
  if DayOfWeek(ADate)=7 then Accept := True;
end;
```

OnMouseMove

Use the OnMouseMove event handler to implement any special processing that should occur as a result of the mouse moving over a date.

The parameters for this event are as follows.

<i>Sender</i> : TObject	Calendar control where the mouse action took place.
<i>Shift</i> : TShiftState	Indicates the state of the Shift keys at the time the mouse was pressed or released.
<i>X, Y</i> : integer;	The mouse position at the time that the mouse was moved. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.
<i>Month, Day, Year</i> : integer;	Indicates which date on the calendar the mouse cursor is over at the time the event is fired. If Day is 0, then the mouse is not over any date.

OnMouseDown, OnMouseUp

Use the OnMouseDown or OnMouseUp event handlers to implement any special processing that should occur as a result of pressing or releasing a mouse button

The parameters for this event are as follows.

<i>Sender</i> : TObject	Calendar control where the mouse action took place.
<i>Button</i> : TMouseButton	Indicates which mouse button was pressed or released.
<i>Shift</i> : TShiftState	Indicates the state of the Shift keys at the time the mouse was pressed or released.
<i>X, Y</i> : integer;	The mouse position at the time the mouse was pressed or released. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.
<i>Month, Day, Year</i> : integer;	Indicates which date on the calendar the mouse cursor is over when the mouse button is pressed or released. If Day is 0, then the mouse is not over any date.

How To

Display more than one month in the calendar control

By increasing the control's *Width* and the *Height* properties, the control will fit additional months into the client area.

Selecting a range of dates

To select a range of dates, you need to set the *Options* | *mdoMultiSelect* to True. To later determine the date range selected by the user, you can refer to the *StartDate* and *EndDate*

properties. To control the maximum number of dates that can be in the date-range use the *MaxSelectCount* property.

TwwDBNavigator



InfoPower includes an extendable DBNavigator component (database navigator to move through and manipulate the data in a dataset), which supports user-definable images and actions, integration with InfoPower's dialogs, flexible control over the layout, user-definable page sizes, and support for multiple rows of icons.



Figure 5.13 - A TwwDBNavigator component

The InfoPower navigator allows you to transparently display the navigator and its related buttons. This helps your applications present a very professional and polished look.

Ancestor

TwwCustomTransparentPanel

Required supporting components

None

Added Properties

AutoSizeStyle

This property defines how the navigator will auto-size itself and its buttons when the size of the navigator changes. Set this property to *asSizeNavigator* if you wish for the navigator to change its size to ensure the navigator's buttons fit. Set this property to *asSizeNavButtons* to adjust the size of the buttons to accommodate the size of the Navigator. When *AutoSizeStyle* is *asNone*, no auto sizing occurs.

Data Type: TwwNavAutoSizeStyle

Valid Values: *asSizeNavigator*, *asSizeNavButtons*, *asNone*

Buttons

This property contains a collection of buttons assigned to the navigator. Each collection item is of type *TwwNavButton*. Clicking on this property from the object inspector brings up InfoPower's collection editor.

Data Type: TwwNavButtons

TwwNavButtons has the following properties you can access during program execution:

Count	Returns the number of buttons in the navigator.
Navigator	Returns the corresponding <i>TwwDBNavigator</i> for the buttons

Items Items is an array containing *TwwNavButton* objects. The value of the *Index* parameter corresponds to the *Index* property of *TwwNavButton*. It represents the position of the item in the collection.

```
property Items[Index: Integer]: TwwNavButton
```

Each button is of type *TwwNavButton*, and has the following properties:

Caption Text displayed beneath button. Requires that the *ShowText* property be set to *True*.

Dialog This property is used in conjunction with the *Style* property to execute a custom InfoPower dialog. When the style property is set to one of the dialog styles, and this property is set, then the corresponding *Dialog* will execute.

Flat Runtime only - Set this property to *True* so that the navigator button appears flat, and does not have borders separating them.

Index Set this property to change the order within the *TwwNavButtons* collection.

ImageIndex Setting this property will override (if any) the image settings and display an image from the navigator's assigned *ImageList*.

LineBreak Set this property to *True* to force a line or column break within the navigator. The navigator's *Layout* property determines if it is a column break (*nVertical*), or a line break (*nHorizontal*).

Margin Margin is the number of pixels between the edge of the button and the image or caption drawn on its surface. If set to *-1*, then the image/caption are automatically centered.

NumGlyphs Set this property to change the number of glyphs in the *ImageList* to associate with this button. See also the *ImageIndex* property, and the navigator's *ImageList* property.

ShowText Set to *True* to display the *Caption* beneath the bitmap.

Spacing Set *Spacing* to the number of pixels that should appear between the image specified by the *ImageIndex* property and the text specified in the *Caption* property.

Style This property determines the behavior and appearance of the button. When setting this property, the image on the button will change to reflect the new style. The action that occurs when clicking on the button is dependent on this property. If this style is set to *nbsCustom*, then no default behavior occurs.

When this property represents one of the InfoPower dialogs, then one of two possible actions can occur. If the *Dialog* property is assigned, then the *Dialog* is executed when the button is clicked. If the dialog is

unassigned, then a dynamically generated Dialog will be created and executed. These dialogs will stay in memory until either the DataSet is destroyed (if style is *nbsFilterDialog*) or when the Navigator is destroyed (all other Dialog styles).

- NavButtons** Runtime only – Returns the collection (*TwwNavButtons*) that contains this button
- Navigator** Runtime only – Returns the navigator (*TwwDBNavigator*) that contains this button

DataSource

This property defines the name of the TDataSource you want to bind the Navigator to.

Data Type: TDataSource

Valid Values: Valid DataSource component name

Flat

Set this property to True so that the navigator buttons appear flat, and do not have borders separating them. When False, the buttons are clearly defined.

Data Type: Boolean

ImageList

Set this property to the TImageList you wish for the buttons to reference via their ImageIndex property.

Data Type: Boolean

Layout

This property specifies the way the buttons are positioned and sized. If this property is set to *nlHorizontal*, buttons are positioned in a left to right order and begin a new row of buttons when reaching the right side of the navigator. When this property is set to *nlVertical*, buttons are positioned in a top to bottom order and begin a new column of buttons when reaching the bottom side of the navigator.

Data Type: TwwNavLayout

Valid Values: nlHorizontal, nlVertical

MoveBy

This property determines the number of records the navigator moves forwards or backwards when the NextPage and PriorPage buttons are pressed.

Data Type: Integer

Options

Options to control the navigator's behavior.

Data Type: Set of TwwNavOptions

Valid Values: noConfirmDelete, noUseInternationalText

- noConfirmDelete* Set this property to True to bring up a confirmation dialog before deleting a record.
- noUseInternationalText* Set this property to True to force the TwwDBNavigator's hints to use the Text that has been set in the TwwIntl component. If this is False, then the default hints will be determined by the international settings when the control was created.

RepeatInterval

This property controls the auto-repeat timing of the navigator. When a user clicks a button and does not release the mouse, the navigator will re-execute the last clicked button after an initial delay. Thereafter it will continue to repeat the execution of the last clicked button until the button is released. For instance if the user presses the *Next Record* button and holds the mouse down, it will continually advance through the dataset until the mouse button is released.

- InitialDelay** The number of milliseconds that passes from the time that the user presses a button to when the button's action begins to repeat.
- Interval** The number of milliseconds that passes between each successive repeat of the button's action, after the initial delay.

Transparent

Set this property to true to paint the navigator transparently.

TransparentClearsBackground

This property is now obsolete. See the Transparent property.

Added Events

OnResize

This event is fired when the TwwDBNavigator is being resized.

TwwNavButton events

OnAfterCreateDialog

Use this event to customize the behavior and/or appearance of the dynamically generated InfoPower dialog. This event is called immediately after the dialog has been created. The event will only fire if the Dialog property for the Button is unassigned and the style property is set to that of an InfoPower dialog.

OnRowChanged

This event occurs immediately after the current record position of the TDataSet has changed.

OnUpdateState

This event is fired when an action occurs that may cause the state of the button to change. For example, when the Active property of the DataSet changes, this event is fired. The following parameters are passed to this event.

Navigator: TwwDBNavigator TwwDBNavigator containing this button

Button: TwwNavButton Button associated with event

Cause: TwwUpdateCause Reason the event was fired. Can be one of the following: usDataChanged, usEditingChanged, usActiveChanged, or usOther.

Added Methods

SetDataSourceFromComponent

Call this method to set the Navigator's DataSource property to the DataSource of *Component*. If *AllowNil* is true and the DataSource property of *Component* is nil, then the Navigator's DataSource property will be set to nil. Otherwise this method never clears the Navigator's DataSource property.

```
Procedure SetDataSourceFromComponent(  
    Component: TComponent; AllowNil: boolean); virtual;
```

TwwNavButtons methods

Add Adds a new button to the navigator. The *AStyle* and *AComponent* parameters set the Style and Dialog properties of the TwwNavButton.

```
function Add(  
    AStyle: TwwNavButtonStyle;  
    AComponent: TComponent): TwwNavButton;
```

AddInfoPowerDialogs Adds all the InfoPower dialogs to the navigator

Clear Clear all the buttons from the navigator

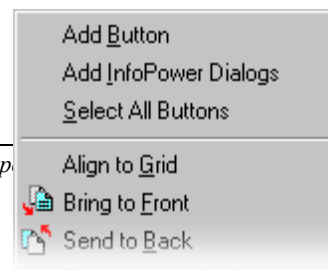
TwwNavButton methods

Click This method handles the default action of the Button. Call this method to simulate the action (clicking) of the button

IsVisible Returns *True* if the Button's position is such that it is within the boundaries of the TwwDBNavigator.

How-to

Design-time Tips



Right-click the navigator at design time to bring up the following selections:

Add Button – Adds a new navigator button.

Add InfoPower Dialogs – Adds a navigator button for the following InfoPower dialogs (i.e. TwwFilterDialog, TwwLocateDialog, etc.).

Select All Buttons – This enables you to select all the buttons at design time. This is useful if you want to modify a property for all the buttons.

Selecting a button without bringing up the collection editor

You can select any button in the navigator by holding down the space key and then clicking on the button.

Using the Navigator's collection editor

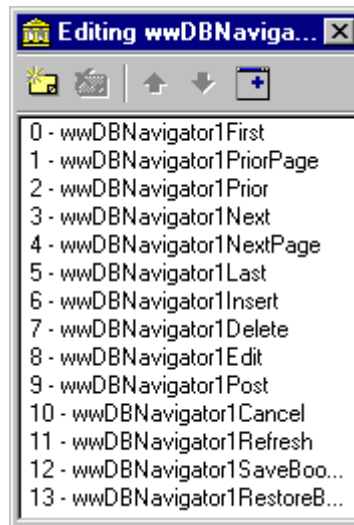
The Navigator's collection editor is modeled after Delphi's existing collection editor. The functionality of the buttons at the top are as follows (in order):

New Button: Creates a new TwwNavButton and adds into the Navigator. (Equivalent to the TwwNavButtons.Add method)

Delete Button: Deletes the currently selected button(s) from the Navigator. (Equivalent to calling TwwNavButton.Free)

Move Button Up: Moves the currently selected button up one level. (Equivalent to decrementing the TwwNavButton.Index property by one).

Move Button Down: Moves the currently selected button down one level. (Equivalent to incrementing the TwwNavButton.Index property by one).



Add InfoPower Dialogs: Adds a button corresponding to each of the InfoPower Dialogs to the Navigator. (Includes: TwwFilterDialog, TwwLocateDialog, TwwRecordViewDialog, and TwwSearchDialog)

Creating logical groupings and regions for the navigator buttons

Use multiple navigators assigned to the same datasource to increase the flexibility of your navigator's layout. For instance the diagram to the right uses three navigators to get its effect.



TwwDBRichEdit, TwwDBRichEditMSWord



Woll2Woll has greatly enhanced the native RichEdit in Delphi, and includes both data-aware and non data-aware versions.

TwwDBRichEditMSWord : InfoPower uses a separate richedit component for integration with MSWord. This component allows the end-users to spell check or grammar check the document using Microsoft Word's native spell checker. The reason we have introduced a separate new component is to avoid the dependence upon the COM Office automation packages when not using MSWord's spell checker. If you use the TwwDBRichEditMSWord component, we recommend that you do NOT put the corresponding IP4000WORD*** package in your project's runtime package list. By avoiding use of these packages as runtime packages, you can omit the distribution of the related Office automation packages as well as these packages.

InfoPower's richedit control now additionally supports the following:


- **New in InfoPower 4000** - Now supports importing from Microsoft Word or mporting and exporting to HTML. Other formats also supported based on the text filters installed on the client computer. For instance, you can import from an Excel spreadsheet. See the \ip4000\demos\richedit\converter.pas file for an exmple of how to import and export. In particular, see the richedit's Import and Export methods.
- **New in InfoPower 4000** - Define headers and footers when using the richedit's Print method. See the \ip4000\demos\richedit\printhead.pas file for an exmple of how to incorporate a header and footer in the richedit's hardcopy printout. In particular see the *PrintHeader* and *PrintFooter* properties.
- **New Mail Merge Example** - Use database fields to fill a richedit's contents. See the \ip4000\demos\richedit\mailmerge.pas file for an example of performing mail-merge with the InfoPower richedit control. The basic idea is to use a template richedit which contains the tags you wish to replace. Then use another richedit control that is to contain the actual contents with the replaced text.


The following lists some of the capabilities of these components.

- ◆ **Full Text Justification Support:** InfoPower now adds full text justification so that the text is aligned to both the left and right margins. This requires the latest riched20.dll (RichEdit Version 3).
- ◆ **Enhanced OLE Support:** InfoPower adds additional OLE dialogs to allow modification of an OLE object's properties. OLE link to file is now supported.
- ◆ **Supports Transparency and custom framing** – Since this control can be used transparently you can now easily use this control like a RichEdit Label control for rich formatted labels in your applications.





OLE support : Embed bitmaps and OLE objects directly into the RichEdit control. You can even save these to your database.


 **Paragraph ruler and spacing control:** Set paragraph indentations within the pop-up rich-edit dialog using an accurate ruler. End-user can also specify the space before and after the paragraph as well as defining the line spacing


 **Internet URL Links :** URL addresses in the rich-edit text are automatically underlined. The component will also automatically open the specified URL with the Internet Browser.

 **Multi-level undo and redo :** Undo or redo a series of actions.


 **Integrated RTF Word processor :** End-users can bring up InfoPower's powerful RTF word processor to give them a full word-processor.

 **Customize printer margins, orientation, and paper size :** Supports end-user customization of the page layout using the Windows PageSetup common dialog.

 **Database Search and Filter :** After storing RTF text into database blob fields, you can still have access to InfoPower's powerful database searching and filtering capabilities.

 **Integration with Microsoft Word's Spelling and Grammar checking:** Use Microsoft Word's Spell check to spell check or grammar check the document. You must use the TwwDBRichEditMSWord component.

 **Background highlighting of selected text:** The user can highlight selected text so that it stands out.

 **Extensive pop-up menu support :** All of the component's functionality is accessible to the end-user by right-clicking the component.

 **Design-time support for entering rich-edit text and OLE into the control.** Delphi's version cannot store formatted text into a control during design time.

 **Seamless integration with InfoPower's Grid and RecordView Components**

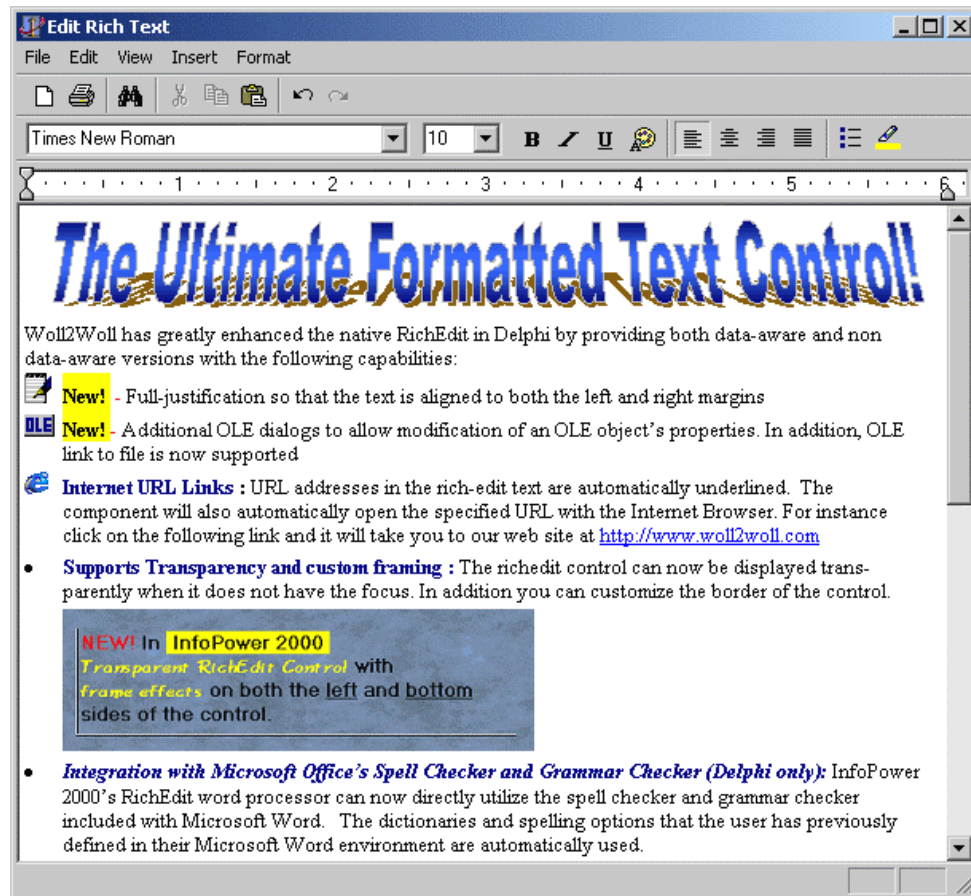


Figure 5.14 – InfoPower's TwwDBRichEdit component allows your end-users to use a full rich text editor to edit richtext fields.

Added Properties

AutoURLODetect

When True, URL addresses in the rich-edit text are automatically underlined. The component will also automatically open the specified URL with the Internet Browser when the user clicks on the link. Use the *OnURLOpen* event to change this default behavior

Data Type: Boolean

DataField

This property defines the name of the field you want displayed in the memo editor window. The default value is blank. If you do not wish to bind the rich-edit control to a table field, then leave both the *DataField* and *DataSource* properties as blank.

Data Type: String

Valid Values: Valid field name where the field is of type TBlobField

DataSource

This property contains the name of a TDataSource component that provides the RichEdit control with data. The default value is blank.

Data Type: TDataSource

Valid Values: Valid DataSource component name

EditorCaption

This property contains a text value that is displayed in the pop-up editor window's title bar. The default value is 'Edit Rich Text'.

Data Type: String

EditorOptions

This property contains a set of Boolean values that control the display of pop-up richtext editor. The user invokes the pop-up richtext editor by right-clicking the control and selecting *Edit*. Alternatively they can invoke the editor by pressing *F2*. Note that these options affect the pop-up richtext editor, and not the pop-up menu.

Data Type: Set of TwwRichEditOption

Valid Values: reoShowLoad, reoShowSaveAs, reoShowSaveExit,

reoShowPrint, reoShowPageSetup, reoShowFormatBar,

reoShowToolBar, reoShowStatusBar, reoShowHints, reoShowRuler,

reoShowInsertObject, reoCloseOnEscape, reoFlatButtons, reoShowSpellCheck,

reoShowMainMenuIcons, reoShowJustifyButton

reoShowSaveExit If True, then the File | Save & Exit menu selection is displayed. This menu selection allows the end-user to save their changes and exit the pop-up editor. Defaults to True

reoShowLoad If True, then the File | Load menu selection is displayed. This menu selection allows the end-user to load text from a file. Defaults to False.

reoShowSaveAs If True, then the File | Save As menu selection is displayed. This allows the end-user to save the richedit's contents to a file. Defaults to False.

reoShowPrint If True, then the File | Print menu selection is displayed. This menu selection allows the end-user to print the richedit's contents. Defaults to True

reoShowPageSetup If True, then the File | Page Setup menu selection is displayed. This menu selection allows the end-user to customize the page-setup in preparation for printing. Defaults to True.

reoShowFormatBar If True, then the FormatBar is displayed. The FormatBar contains icons for formatting the text. Defaults to True.

<i>reoShowToolBar</i>	If True, then the ToolBar is displayed. The ToolBar contains icons for clearing, printing, searching, and clipboard operations. Defaults to True.
<i>reoShowStatusBar</i>	If True, then the StatusBar is displayed at the bottom of the pop-up richtext editor. The StatusBar contains the Keyboard State (Caps and Num Lock) and Hints. Defaults to True.
<i>reoShowHints</i>	If True, then Hints are enabled for the icons and menu selections. Defaults to True.
<i>reoCloseOnEscape</i>	If True, then the Escape key will close the dialog with a ModalResult of mrCancel. Default is True.
<i>reoShowRuler</i>	When True, then the editor displays a ruler to allow the end-user to customize the paragraph indentations.
<i>reoShowInsertObject</i>	When True, then the Insert Object... menu selection is displayed. The menu selection allows the end-user to insert an OLE object .
<i>reoFlatButtons</i>	When True, the editor's buttons appear flat and do not have borders separating them. When False, the buttons are clearly defined.
<i>reoShowSpellCheck</i>	When True, the editor displays a speed button which will invoke the Microsoft Word spell checker to spell check the document. Note: Spell checking is only enabled if you are using Delphi5 and the TwwDBRichEditMSWord component.
<i>reoShowMainMenuIcons</i>	When True, the editor's main menu will display icons for certain menu selections.
<i>reoShowJustifyButton</i>	When True, the editor displays a speed button which when clicked will fully justify the paragraph. Support of this capability is dependent upon your user's environment supporting RichEdit version 3. Right-click the file \windows\system\riched20.dll from windows explorer, and check its version description. It will indicate something like "Rich Text Edit Control v3.0" if it supports version 3. Windows 2000 and Office 2000 should come with version 3 of riched20.dll and update your environment accordingly.
<i>reoUseBackColor</i>	When True, the editor's background color of its embedded richedit control will be the same color as the associated richedit control.
<i>reoNoConfirmation</i>	When True, the editor will not prompt the user with a confirmation dialog, but instead will assume that the editor changes are copied back to the associated richedit control.

EditorPosition

This property allows you to determine the location and size of the pop-up word processor.

Height Height in pixels of the pop-up word processor. Default of 0 means to use the default height.

Left Left position of the pop-up word processor. Default of 0 means to auto-center the editor horizontally.

Top Top position of the pop-up word processor. Default of 0 means to auto-center the editor vertically.

Width Width in pixels of the pop-up word processor. Default of 0 means to use the default width.

EditWidth

This property determines the wrapping boundaries of the pop-up richtext editor. If set to `rewWindowSize` then text is wrapped according to the window boundaries. If set to `rewPrinterSize` then text is wrapped as defined by the page layout settings. Defaults to `rewPrinterSize`.

Data Type: `TwwRichEditWidth`

Valid Values: `rewWindowSize`, `rewPrinterSize`

Frame

See the topic “Key properties and events for custom framing” in chapter 4 for information on this property.


Data Type: `TwwEditFrame`

GutterWidth

Set this property to change the number of pixels of fixed spacing between the text and the edge of the control.

Data Type: Integer

HighlightColor

Set this property to change the color used to highlight text in the word processor. It defaults to `clYellow`. The user can highlight text in the word processor by clicking on the  speed button.

Lines

This property contains the individual lines of text in the rich text edit control. Click on this property at design time to assign rich-edit text to an unbound control.

Use *Lines* to manipulate the text in the rich text edit control on a line by line basis. *Lines* is a `TStrings` object, so `TStrings` methods may be used for *Lines* to perform manipulations such as counting the lines of text, adding lines, deleting lines, or replacing the text in lines.

To work with the text as one chunk, use the `Text` property. To manipulate individual lines of text, the `Lines` property works better.

Data Type: `TStrings`

MeasurementUnits

Unit of measurement to display in the control's dialogs (i.e. PageSetup, Tab Settings, Paragraph Settings). Also determines the units of the PrintMargins property.

Data Type: TwwMeasurementUnits

Valid Values: muInches, muCentimeters

OLEOptions

- reoAdjustPopupMenu* When True, if an OLE is selected, the pop-up menu is adjusted to include the menu selections relating to the selected OLE object. User can click on the *Insert Object...* popup selection to bring up a dialog to insert an OLE object.
- reoDisableOLE* When True, the end-user will not be able to embedding any OLE object.

PrintFooter

Assign this property to integrate headers into the printed output produced from the *Print* method. If you wish to include page numbers in your footer, then assign the footer's text using the *OnPrintFooter* event.

Example: The following code attached to the OnPrintFooter event assigns the text like "Page 1 of 5" to the right-most text in the footer, and assigns the current date to the left-most text.

```
procedure TPrintHeaderForm.wwDBRichEdit1PrintFooter(Sender:
TwwCustomRichEdit;
  DrawRect: TRect; PageNumber: Integer; var LeftText, CenterText,
  RightText: String; var DoDefault: Boolean);
begin
  RightText:= 'Page ' + inttostr(PageNumber) + ' of ' +
    inttostr(Sender.TotalPages);
  LeftText:= datetostr(date);
end;
```

Data Type: TwwRTFHeaderFooter

TwwRTFHeaderFooter is a class defined as follows:

- VertMargin : Double Vertical unit space separating footer from edge of text
- LeftText: string Assign this property to define the text that appears on the left of the footer.
- CenterText: string Assign this property to define the text that appears in the middle of the footer.
- RightText: string Assign this property to define the text that appears on the right of the footer.
- Font: TFont Assign this property to modify the font of the footer
- LineSeparator: Boolean Set to true to include a line separating the footer from the richedit text.

PrintHeader

Assign this property to integrate headers into the printed output produced from the *Print* method. If you wish to include page numbers in your footer, then assign the header's text using the *OnPrintHeader* event. See the *PrintFooter* property for an example of including page numbers, as well as a description of the *TwwRTFHeaderFooter* type.

Data Type: *TwwRTFHeaderFooter*

PrintPageSize

This defines the page size of the richedit control. Some of the possible values are as follows. These values are defined by windows.

Data Type: Integer

Valid Values: See Below

```
DMPAPER_LETTER = 1; { Letter 8 1/2 x 11 in }
DMPAPER_FIRST = DMPAPER_LETTER;
DMPAPER_LETTERSMALL = 2; { Letter Small 8 1/2 x 11 in }
DMPAPER_TABLOID = 3; { Tabloid 11 x 17 in }
DMPAPER_LEDGER = 4; { Ledger 17 x 11 in }
DMPAPER_LEGAL = 5; { Legal 8 1/2 x 14 in }
DMPAPER_STATEMENT = 6; { Statement 5 1/2 x 8 1/2 in }
DMPAPER_EXECUTIVE = 7; { Executive 7 1/4 x 10 1/2 in }
DMPAPER_A3 = 8; { A3 297 x 420 mm }
DMPAPER_A4 = 9; { A4 210 x 297 mm }
DMPAPER_A4SMALL = 10; { A4 Small 210 x 297 mm }
DMPAPER_A5 = 11; { A5 148 x 210 mm }
DMPAPER_B4 = 12; { B4 (JIS) 250 x 354 }
DMPAPER_B5 = 13; { B5 (JIS) 182 x 257 mm }
DMPAPER_FOLIO = 14; { Folio 8 1/2 x 13 in }
DMPAPER_QUARTO = 15; { Quarto 215 x 275 mm }
DMPAPER_10X14 = 16; { 10x14 in }
DMPAPER_11X17 = 17; { 11x17 in }
DMPAPER_NOTE = 18; { Note 8 1/2 x 11 in }
DMPAPER_ENV_9 = 19; { Envelope #9 3 7/8 x 8 7/8 }
DMPAPER_ENV_10 = 20; { Envelope #10 4 1/8 x 9 1/2 }
DMPAPER_ENV_11 = 21; { Envelope #11 4 1/2 x 10 3/8 }
DMPAPER_ENV_12 = 22; { Envelope #12 4 1/4 x 11 }
DMPAPER_ENV_14 = 23; { Envelope #14 5 x 11 1/2 }
DMPAPER_CSHEET = 24; { C size sheet }
DMPAPER_DSHEET = 25; { D size sheet }
DMPAPER_ESHEET = 26; { E size sheet }
DMPAPER_ENV_DL = 27; { Envelope DL 110 x 220mm }
DMPAPER_ENV_C5 = 28; { Envelope C5 162 x 229 mm }
DMPAPER_ENV_C3 = 29; { Envelope C3 324 x 458 mm }
DMPAPER_ENV_C4 = 30; { Envelope C4 229 x 324 mm }
DMPAPER_ENV_C6 = 31; { Envelope C6 114 x 162 mm }
DMPAPER_ENV_C65 = 32; { Envelope C65 114 x 229 mm }
DMPAPER_ENV_B4 = 33; { Envelope B4 250 x 353 mm }
DMPAPER_ENV_B5 = 34; { Envelope B5 176 x 250 mm }
DMPAPER_ENV_B6 = 35; { Envelope B6 176 x 125 mm }
DMPAPER_ENV_ITALY = 36; { Envelope 110 x 230 mm }
DMPAPER_ENV_MONARCH = 37; { Envelope Monarch 3.875 x 7.5 in }
DMPAPER_ENV_PERSONAL = 38; { 6 3/4 Envelope 3 5/8 x 6 1/2 in }
DMPAPER_FANFOLD_US = 39; { US Std Fanfold 14 7/8 x 11 in }
DMPAPER_FANFOLD_STD_GERMAN = 40; { German Std Fanfold 8 1/2 x 12 in }
```

DMPAPER_FANFOLD_LGL_GERMAN = 41; { German Legal Fanfold 8 12 x 13 in }

PopupMenu

Pop-up menu for the rich text control. There already is a built-in pop-up menu for this control, but you can override it to use your own with this property. See also the *PopupMenuOptions* property as this allows you to customize the built-in PopupMenu.

Data Type: TPopupMenu

PopupMenuOptions

This property allows you to select which selections are available to the end-user through the pop-up menu.

Data Type: Set of TwwRichEditPopupMenuOption

Valid Values: rpoPopupMenuEdit, rpoPopupMenuCut, rpoPopupMenuCopy, rpoPopupMenuPaste, rpoPopupMenuBold, rpoPopupMenuItalic, rpoPopupMenuUnderline, rpoPopupMenuFont, rpoPopupMenuBullet, rpoPopupMenuParagraph, rpoPopupMenuTabs, rpoPopupMenuFind, rpoPopupMenuReplace, rpoPopupMenuInsertObject, rpoPopupMenuMSWordSpellCheck

<i>rpoPopupMenuEdit</i>	If True, then the user can click on the <i>Edit</i> popup selection to bring up a pop-up editor window. Defaults to True.
<i>rpoPopupMenuCut</i>	If True, then the user can click on the <i>Cut</i> popup selection to cut the currently selected text. Defaults to True.
<i>rpoPopupMenuCopy</i>	If True, then the user can click on the <i>Copy</i> popup selection to copy the currently selected text to the clipboard. Defaults to True.
<i>rpoPopupMenuPaste</i>	If True, then the user can click on the <i>Paste</i> popup selection to paste the clipboard's contents to the control. Defaults to True.
<i>rpoPopupMenuBold</i>	If True, then the user can click on the <i>Bold</i> popup selection to bold-face the currently selected text. Defaults to False.
<i>rpoPopupMenuItalic</i>	If True, then the user can click on the <i>Italic</i> popup selection to italicize the currently selected text. Defaults to False.
<i>rpoPopupMenuUnderline</i>	If True, then the user can click on the <i>Underline</i> popup selection to underline the currently selected text. Defaults to False.
<i>rpoPopupMenuFont</i>	If True, then the user can click on the <i>Font</i> popup selection to change the font of the currently selected text. Defaults to True.
<i>rpoPopupMenuBullet</i>	If True, then the user can click on the <i>Bullet</i> popup selection to enable or disable bullets for the currently selected text. Defaults to True.
<i>rpoPopupMenuParagraph</i>	If True, then the user can click on the <i>Paragraph</i> popup selection to underline the currently selected text. Defaults to True.
<i>rpoPopupMenuTabs</i>	If True, then the user can click on the <i>Tabs</i> popup selection to customize the tab-stops for the currently selected text. Defaults to True.

<i>rpoPopupFind</i>	If True, then the user can click on the <i>Find</i> popup selection to bring up a dialog to search for text in the edit control. Defaults to True.
<i>rpoPopupReplace</i>	If True, then the user can click on the <i>Replace</i> popup selection to bring up a dialog to search and replace text in the edit control. Defaults to True.
<i>rpoPopupInsertObject</i>	When True, the user can click on the <i>Insert Object...</i> popup selection to bring up a dialog to insert an OLE object.
<i>rpoPopupMSWordSpellCheck</i>	When True, the user can click on the Check Spelling popup selection to bring up a the Microsoft Word spell check to spell check the document. Note: Spell checking is only enabled if you are using Delphi5 and the TwwDBRichEditMSWord component

PrintJobName

Set this property to change the print job name used by the pop-up word processor when the user selects the *File | Print* menu selection.

PrintMargins

This property allows you to define printer margins for when you print the contents of the control. The units of measurement are defined by the TwwDBRichEdit's *MeasurementUnits* property.

- Bottom** Blank space to leave at the bottom of the printout.
Data Type: Double
Valid Values: Valid value for the printer page layout. Value type is defined by the MeasurementUnits property.
- Left** Blank space to leave at the left-edge of the printout.
Data Type: Double
Valid Values: Valid value for the printer page layout. Value type is defined by the MeasurementUnits property.
- Right** Blank space to leave at the right-edge of the printout.
Data Type: Double
Valid Values: Valid value for the printer page layout. Value type is defined by the MeasurementUnits property.
- Top** Blank space to leave at the top of the printout.
Data Type: Double
Valid Values: Valid value for the printer page layout. Value type is defined by the MeasurementUnits property.

UserSpeedButton1

Add additional speedbuttons to the pop-up richedit dialog to integrate other 3rd-party tools such as RTF spell checkers. Set this property to the speedbutton you wish to add to the word-processor, and attach the code to the speedbutton that you wish to execute when it is clicked. For instance the following code inserts the current date into the control

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  with TwwRichEditForm(GetParentForm(Sender as TControl)).richedit1 do
  begin
    sellength:= 0;
    selText:= DateToStr(Date);
  end
end;
```

Note: You will need to add *wwrich* to your form's *uses* clause to resolve the TwwRichEditForm reference.

Data Type: TSpeedButton

Valid Values: Any TSpeedButton control.

UserSpeedButton2

This gives you a 2nd speed button to add to the word processor. See *UserSpeedButton1* for its usage.

Added Events

Some of the following events pass a handle to the form containing the richtext editor. To see what objects are contained within this editing form, open up *wwrich.pas* in the InfoPower source sub-directory. If you do not have the source code version of InfoPower, then perform the steps in Chapter 4's topic "Determining the object names of the controls contained in an InfoPower dialog" on the *wwrich.dfm* file contained in the InfoPower lib directory.

If you want to customize any of the objects contained by the form you can use the *OnInitDialog* event. However if all you are trying to do is to change the labels and hints, then use the *TwwIntl | RichEdit* property.

OnCloseDialog

This event allows you to perform any custom action before the pop-up richtext editor is closed.

The parameters for this event are as follows.

Form: TForm	TForm handle to popup richtext editor
-------------	---------------------------------------

OnCreateDialog

This event allows you to perform any custom action immediately after the pop-up richtext editor is created. You may wish to use this event instead of the *OnCloseDialog* event if you need the code executed before the pop-up editor window is created.

The parameters for this event are as follows.

Form: TForm TForm handle to popup richtext editor

OnInitDialog

This event allows you to perform any custom action before the pop-up richtext editor is initially displayed.

OnMenuLoadClick

This event allows you to perform any custom action when the user selects Load from the pop-up word processor. The parameters for this event are as follows.

Form TForm handle to popup richtext editor
RichEdit RichEdit on the pop-up word processor. Do not confuse this with the richedit on your own form.
DoDefault Set to False to prevent the default action from occurring.

OnMenuPrintClick

This event allows you to perform any custom action when the user selects Print from the pop-up word processor. See the OnMenuLoadClick event for a description of the parameters to this event.

OnMenuSaveAsClick

This event allows you to perform any custom action when the user selects 'Save As' from the pop-up word processor. See the OnMenuLoadClick event for a description of the parameters to this event.

OnMenuSaveAndExitClick

This event allows you to perform any custom action when the user selects 'Save and Exit' from the pop-up word processor. See the OnMenuLoadClick event for a description of the parameters to this event.

OnPrintFooter

This event allows you to customize the footer for each page. The event is fired prior to the formatting of each page for the printer when using the richedit's Print method.. This event is useful for inserting dynamic page information into the printed output, such as page numbers. See the PrintFooter property for an example of using this event.

Sender: TwwCustomRichEdit RichEdit control associated with footer
DrawRect: TRect Printing rectangle for footer/header
var LeftText, CenterText, RightText: string

	Assign these properties to customize the text in the footer/header
PageNumber: integer	Current page of printed output
var DoDefault: Boolean	Set this property to False to disable the default formatting of the footer/header. You may wish to set this to false if you have done your own painting of the footer/header

OnPrintHeader

This event allows you to customize the header for each page. The event is fired prior to the formatting of each page for the printer when using the richedit's Print method. This event is useful for inserting dynamic page information into the printed output, such as page numbers. See the PrintFooter property for an example of using this event.

See the OnPrintFooter event for a description of the parameters.

OnURLOpen

When the end-user clicks on an URL link with the document, the rich edit control will open the default Internet Browser at the specified URL address. If you wish to change this behavior, you can use this event to perform your own custom actions.

The parameters for this event are as follows.

<i>Sender</i> : TwwCustomRichEdit	RichEdit control containing the URL link
<i>URLLink</i> : String;	String containing the text of the URL link
<i>UseDefault</i> : boolean;	Set to True to perform the default behavior of opening the link with the default Internet Browser. Set to False to disable the default behavior.

Added Methods

AppendRichEditFrom

Calling this method appends the contents of SourceRichEdit to the current richedit control.

```
Procedure AppendRichEditFrom(SourceRichEdit: TCustomRichEdit);
```

CanPaste

This method returns true if there is text in the clipboard that can be pasted into the control.

```
Function CanPaste: boolean;
```

CanUndo

This method returns true if the control is capable of undoing the last editing operation.

```
Function CanUndo: boolean;
```

CanCut

This method returns true if there is selected text which can be cut to the clipboard.

```
Function CanCut: boolean;
```

CanFindNext

This method returns true if the control repeats the last search operation. The return value will be false if no search has been previously performed.

```
Function CanFindNext: boolean;
```

CanRedo

This method returns true if the control is capable of redoing the last editing operation.

```
Function CanRedo: boolean;
```

CopyRichEditFromBlob

This method allows you to copy from a Blob Field to the rich edit control.

```
Procedure CopyRichEditFromBlob(Field: TField);
```

CopyRichEditTo

This method allows you to copy the richtext from one rich edit control to another.

```
Procedure CopyRichEditTo(val: TCustomRichEdit);
```

CopyRichEditToBlob

This method allows you to copy from a rich edit control to a Blob Field.

```
Procedure CopyRichEditToBlob(Field: TField);
```

Execute

Calling this method brings up the pop-up richtext editor. A return value of True is returned if the user saved their changes.

```
Function Execute: boolean;
```

ExecuteFindDialog

Calling this method brings up the Find Dialog, where the end-user can search for text in the richedit control.

```
Procedure ExecuteFindDialog; virtual;
```

ExecuteReplaceDialog

Calling this method brings up the Find Dialog, where the end-user can search and replace text in the richedit control.

```
Procedure ExecuteReplaceDialog; virtual;
```

ExecuteFontDialog

Calling this method brings up the Windows Font Dialog

```
Procedure ExecuteFontDialog; virtual;
```

ExecuteParagraphDialog

Calling this method brings up the paragraph dialog, where the end-user can assign paragraph indentation properties.

```
function ExecuteParagraphDialog: boolean; virtual;
```

ExecuteTabDialog

Calling this method brings up the tab dialog, where the end-user can set tab stops.

```
Procedure ExecuteTabDialog; virtual;
```

FindNextMatch

Calling this method repeats the last search performed by the FindDialog

```
Procedure FindNextMatch; virtual;
```

FindReplace

Calling this method repeats the last replace performed by the ReplaceDialog

```
Procedure FindReplace; virtual;
```

FindReplaceText

Calling this method searches for the string defined by *SearchText* and replaces it with *ReplaceText*. The function returns true if a match was found. SearchTypes is a set of TSearchType = (stWholeWord, stMatchCase). The search begins from the current cursor position. If you wish to ensure that the search starts from the beginning of the text, then set the SelStart property to 0.

```
Function FindReplaceText(SearchText, ReplaceText: string;  
    SearchTypes: TSearchTypes): boolean; virtual;
```

Example: The following code replaces all occurrences of the string '\$Company\$' with the string 'Woll2Woll Software'.

```
wwDBRichEdit2.SelStart:= 0;  
while wwDBRichEdit2.FindReplaceText(  
    '$Company$', 'Woll2Woll Software', []) do;
```

GetRTFText

Call this method to get the raw RTF text. This differs from the text property, which returns the unformatted text of the control.

Export

Call this method to export the richedit's contents to a file named *FileName* of the format defined by *Format*. File formats supported are determined by the export filters installed on the

client system. See the \ip4000\demos\richedit\converter.pas file for an example of how to import and export. Also see the *Import* method for further documentation.

```
Procedure Export(Format: string; FileName: string);
```

Import

Call this method to import from a file named *FileName* of the format defined by *Format* into the richedit contents. File formats supported are determined by the import filters installed on the client system. See the \ip4000\demos\richedit\converter.pas file for an example of how to import and export.

```
Procedure Import(Format: string; FileName: string);
```

Example: The following code imports from the HTML file *test.html* into the richedit.

```
Import('HTML', 'TEST.HTML');
```

If you wish to integrate importing and exporting with a TOpenDialog and TSaveDialog, you can use the TwwRTFConverterList to assist you. TwwRTFConverterList is defined in the unit wwrftconverter. Again see \ip4000\demos\richedit\converter.pas for practical usage.

constructor Create(import: boolean)	When constructing the class, pass true for import when generating a list of the client's installed import filters. Use a False value for import when constructing a list of export filters.
LibPath: TStringList	List containing location of each filter
Description: TStringList	List containing descriptions of each filter
FormatClass: TStringList	List containing format class of filters. Elements of this string list can be used as the <i>Format</i> for the <i>Import</i> and <i>Export</i> methods.
Filters: TStringList	List containing names of filters.
FilterList: AnsiString	String containing filters. You can use these filters with the TOpenDialog and TSaveDialog Filter properties.

MSWordSpellChecker

Calling this method invokes Microsoft Word to spell check the text. This method requires Delphi 5 or later versions. See also the properties *PopupOptions.rpoPopupMSWordSpellCheck* and *EditorOptions.reoShowSpellCheck* to allow built-in menus to invoke the spell checker.

```
Function MSWordSpellChecker: boolean; virtual;
```

Print

This method prints the contents of the richedit control. The print job name is defined by the *Caption* property. See also the *PrintMargins*, *PrintFooter*, and *PrintHeader* properties.

```
procedure Print(const Caption: string);
```

Redo

Calling this method asks the richedit control to redo the last operation undone by a Undo.

SetBullet

Calling this method enables or disables the bullet style of the current paragraph

```
Procedure SetBullet(val: boolean);
```

SetBold

Calling this method enables or disables the boldface attribute of the currently selected text.

```
Procedure SetBold(val: boolean);
```

SetItalic

Calling this method enables or disables the italic attribute of the currently selected text.

```
Procedure SetItalic(val: boolean);
```

SetUnderline

Calling this method enables or disables the underline attribute of the currently selected text.

```
Procedure SetUnderline(val: boolean);
```

Undo

Calling this method asks the richedit control to undo the last operation.

How To

Bind the control to a database field

In order to use the rich-edit control with a database field, you will need to attach the control to a TBlobField. You can create a TBlobfield by editing your table structure with a tool such as Database Desktop, and then adding a blob field.

Store RichText into a control at design time

You can store rich-edit text into the control at design time, by dbl-clicking the component and entering your text. This technique is only valid with an unbound control, as bound controls will retrieve their data from the table field.

Display the pop-up richtext editor without displaying the control

In many cases you may just want to display the pop-up richtext editor without displaying the control. This can be accomplished by setting the visible property of the control to False, and then calling its execute method from your button or event.

```
wwDBRichEdit1.execute;
```

Change the richedit's main menu to call your own custom code

The pop-up richtext editor has a menu that can be partially customized through the *EditorOptions* property. However your customization options are limited through this property. For greater flexibility in manipulating the menu, use the *OnInitDialog* event. You can attach code in this event to add additional menu choices to the pop-up editor.

The following example will add a new menu group called *Tools*, and have a menu selection item of *Greeting*. When the user clicks on *Tools | Greeting*, the message “Hello” is displayed. Also make sure that the *menus* unit is added to your form's *uses clause* so that the compiler can recognize the *TMenuItem* component.

```
procedure TForm1.wwDBRichEdit1OnInitDialog(Form: TForm);
var ToolMenuItem: TMenuItem;
function AddMenuItem(Owner: TComponent;
  ACaption: string; event: TNotifyEvent): TMenuItem;
var menuItem: TMenuItem;
begin
  menuItem:= TMenuItem.create(Owner);
  menuItem.caption:= ACaption;
  menuItem.OnClick:= event;
  result:= menuItem;
  if Owner is TMenu then (Owner as TMenu).items.Add(menuItem)
  else (Owner as TMenuItem).Add(menuItem)
end;
begin
  ToolMenuItem:= AddMenuItem(Form.Menu, 'Tools', Nil);
  AddMenuItem(ToolMenuItem, 'Greeting', GreetingClick);
end;
```

In addition to attaching code to the *OnInitDialog* event, you will also need to declare and define the code for your menu item events. The following is the code that is executed when the user selects *Tools | Greeting*. Note that you would also need to add the declaration of *GreetingClick* to your own form.

```
procedure TForm1.GreetingClick(Sender: TObject);
begin
  showmessage('Hello');
end;
```

Embed richedit text in the grid

If you want to see a text representation of your richtext in the grid, the steps are as follows.

1. Dbl-click the grid and select your richedit field.
2. Then choose the Edit Control tab page and change the *Control Type* to RichEdit.

3. Optional - If you want to allow the end-user to edit the richtext from the grid, then select a rich-edit control using the *Control Name* drop-down. The user can then edit the field by pressing the *F2* keystroke or dbl-clicking the grid cell.

Automatically append text when opening the pop-up richtext editor (i.e. add current date/time)

The following code attached to the *OnInitDialog* event will add the current timestamp to the end of the richedit's contents. You will need to add *wwrich* to your uses clause if it is not already there.

```
procedure TForm1.wwDBRichEdit1InitDialog(Form: TForm);
begin
  with (Form as TwwRichEditForm).RichEdit1 do
  begin
    selStart:= length(text);
    lines.add(datetostr(Now));
  end;
end;
```

Force word wrapping to adjust to the printer page size instead of the window size

Set the *EditWidth* property to *rewPrinterSize*.

TwwDBSpinEdit



TwwDBSpinEdit component gives your end-users the ability to easily and quickly increment or decrement formatted numeric and date values by clicking the mouse button or by pressing the up and down cursor arrow keys. You define the data source and field names, along with minimum, maximum and increment values. This component can also be used in an unbound manner (without specifying data source or data field values).

The end-user can modify the contents of the TwwDBSpinEdit by clicking on the up and down icons. Alternatively they can use the UP Arrow and DOWN Arrow keys. On Date, Time, and DateTime fields the spinedit will highlight the text that is changing while spinning, so the user will visually see what is changing.

InfoPower adds the ability to spin formatted numeric text so that the user sees a more meaningful representation of the value. Set the *TField.DisplayFormat* property of the field indicated by the *DataField* property to spin formatted numerics.

Note: See also the TwwDBDateTimePicker component if you wish to edit dates or times.



Figure 5.15 - The TwwDBSpinEdit component

Ancestor

TCustomMaskEdit

└TwwDBCUSTOMEdit

Required supporting components

None

Added Properties

EditorEnabled

When False, the user is not able to type into the TwwDBSpinEdit. The user can still change the value using the Up/Down Arrow keys or by clicking on the spinedit icons.

Data Type: Boolean

Increment

The *Increment* property is the value that the component increments/decrements when spinning the value. This property is ignored when using a date field. Date fields will automatically

determine the increment based on the cursor position within the control. For instance if the cursor is on the Year portion then it will automatically increment/decrement by one year.

Data Type: Double

MinValue

The *MinValue* property is the minimum value allowed by the component. If you are trying to set the *MinValue* of a date field, we recommend you set this property via code to simplify the translation from a date to a double. If the properties *MinValue* and *MaxValue* are both 0, then the component ignores these limits.

Data Type: Double

Example: The following code sets the *MinValue* so that you can't spin to dates less than the current date.

```
wwDBSpinEdit1.MinValue := Now;
```

MaxValue

The *MaxValue* property is the maximum value allowed by the component. If you are trying to set the *MaxValue* of a date field, we recommend you set this property via code to simplify the translation from a date to a double. If the properties *MinValue* and *MaxValue* are both 0, then the component ignores these limits.

Data Type: Double

Example: The following code sets the *MaxValue* so that you can't spin to dates greater than the current date.

```
wwDBSpinEdit1.MaxValue := Now;
```

UnboundDataType

When this component is used without a datasource and datafield, you can still force it to spin as a date or time value by setting this property. This property also determines how the component will auto-fill when the space key is entered by the end-user.

Data Type: TwwEditDataType

Valid Values:

<i>wwDefault</i>	Spin as a numeric
<i>wwEditDate</i>	Spin as a Date
<i>wwEditTime</i>	Spin as a Time
<i>wwEditDateTime</i>	Spin as a date and time

Value

Current value of component.

Data Type: Double

Tips

- ◆ Remember that for a TDateField or a TDateTimeField, the component will automatically handle the spinning increment based on the current cursor position within the date.
- ◆ If you wish to use 4 digit years, then set the Delphi property ShortDateFormat. For instance the following line of code your main form's OnShow event will display years using 4 digits.

```
procedure TMainDemo.FormShow(Sender: TObject);  
begin  
    ShortDateFormat:= 'mm/dd/yyyy';  
end;
```

Similarly if you wish to display and edit time fields without seconds, then set the LongTimeFormat.

```
LongTimeFormat:= 'h:mm AMPM';
```

See the Delphi documentation under date/time formatting for more details on controlling the format of dates and times

TwwExpandButton



Use a TwwExpandButton to embed an expandable grid or inspector inside a grid. This allows you to elegantly display master/detail relationships from a single starting grid. The end-user clicks on the expand button attached to a grid column, and then the detail grid is displayed. The detail grid has the full functionality of a TwwDBGrid, and can be edited.

You can also associate a TwwDataInspector component with the expand button. This is convenient for display concatenated fields in the grid, and then allowing the user to edit the individual fields when he/she clicks on the expand button.

Note: Do not use a TwwExpandButton as a standalone component outside the grid. This component is specifically designed for embedding in the InfoPower grid. In the future this control may be expanded, but at this time it only supports being embedded in the TwwDBGrid.

The basic steps to embed a detail grid or inspector are the following.

1. Drop in a TwwExpandButton and set its Grid property to the detail grid or inspector.
2. Create a calculated or lookup field in your dataset. You will later attach your expand button to this field.
3. Dbl-click the master grid to bring up the *Select Fields Dialog*, and set the Control Type property of the newly created calculated or lookup field to custom. Then choose the TwwExpandButton you previously created as the custom control.

The ExpandButton uses the following rules

- ◆ See the *AutoShrink* property to have the detail grid automatically shrink if there are not enough records to fill the entire detail grid.
- ◆ If the user enters Ctrl-Right or the <space> key while in the control, the associated grid is brought into view.
- ◆ When the user manipulates the appearance of the master grid, the detail grid is automatically collapsed. For instance if the user advances to the next column in the master grid, or changes the column order, the detail grid is collapsed.
- ◆ You can automatically hide the expand/collapse icons by using *Options | AutoHideExpand* to True. The icons are hidden if the value of the field is 0 or null when this property is true.
- ◆ Set *ShowText* to true if you wish to display the text of the associated field within the control.
- ◆ Set *GridIndents* to change the indentation of the expanded grid when it appears.
- ◆ Set *Indents* to change the indentation of the expand/collapse icons as well as any text that appears in the control.

InfoPower 4000 introduces the ability to associate a panel with the expandbutton. Previously you were restricted to using a drop-down grid or drop-down inspector from its clickable expand button. By allowing a panel, your grid's capabilities and display options are dramatically improved. See the how-to topics for information on how-to associate a panel to the control.

Ancestor

TCustomCheckBox

└─TwwCustomCheckBox

└─TwwDBCCustomCheckBox

Added Properties

AutoHideExpandButton

You can automatically hide the expand/collapse icons by using *Options | AutoHideExpand* to True. The icons are hidden if the value of the field is 0 or null when this property is true. This property provides a convenient way of informing the user that there are no detail records for the given expand button. If you also wish to also prevent the drop-down grid from expanding via the <space> or <Ctrl><Right> keystrokes, then add the following line of code to your *OnBeforeExpand* event. The default behavior of the control does not prevent this so that there is still a way to edit/insert records in the detail grid

Data Type: Boolean

Example:

```
procedure TMasterDetailGridForm.wwExpandButton1BeforeExpand(Sender:TObject);
begin
  with (Sender as TwwExpandButton).Field do
    if IsNull or (Text='') then abort;
end;
```

AutoShrink

The design-time height of the detail grid is used as the grid's size when the grid is expanded. If *AutoShrink* property is True, then the expand button will shrink the grid if there are not enough records to fill the entire grid. You may wish to only use AutoShrink for your terminal grid (last detail grid without any contained expand buttons), as with non-terminal nodes you may see some jumpiness as the parent grid tries to ensure that the entire detail grid can fit in its confined grid area.

Data Type: Boolean

ButtonAlignment

Assign this property to change the location of the expand icons with respect to the text. Note: Text only appears in the control when *ShowText* is True. If *ButtonAlignment* is set to

taLeftJustify, then the icons appear to the left of the text. If *ButtonAlignment* is set to *taRightJustify*, then the icons appear to the right of the text. If *ButtonAlignment* is set to *taCenter*, then the icons appear in the middle. Warning: Set *ShowText* to *false* if your *ButtonAlignment* is set to *taCenter*, as otherwise your text and icon will overlap.

Data Type: TAlignment

DataSource

Name of datasource you are filtering or querying

Data Type: TDataSource

Valid Values: Any TDataSource Component

DataField

Name of field whose value is displayed in the control if ShowText is True. Setting this property will not have any effect, as the grid re-assigns this property to be the name of the field that the control is attached to.

Data Type: String

Expanded (Runtime only)

Set this property to True at runtime to force the expand button to expand its associated grid so that it is shown within the parent grid.

Data Type: Boolean

Grid

Assign this property to the grid or inspector you wish to see when the user clicks on the expand button. The grid is automatically shrunk (when AutoShrink is True) if the grid determines that it can display all the records with a smaller height. You can set this to a TwwDBGGrid, TwwDataInspector, or a TPanel Control.

Data Type: TWinControl

GridIndents

Use GridIndents to change the relative placement of the expanded grid.

- | | |
|----------|--|
| X | Assign this property to specify the number of pixels to move the expanded grid to the left (positive value) or right (negative value). |
| Y | Assign this property to specify the number of pixels to move the expanded grid upward (negative value) or downward (negative value). |

Images

Assign this property if you wish to change the expand/collapse icons displayed by the TwwExpandButton. The first image in the imagerlist is used as the expand icon, and the second image is used as the collapse icon.

Data Type: TImageList

Indents

Use Indents to change the relative placement of expand/collapse icon and the text.

- ButtonX** Assign this property to specify the number of pixels to move the expand/collapse icon to the left (positive value) or right (negative value).
- ButtonY** Assign this property to specify the number of pixels to move the expand/collapse icon upward (negative value) or downward (negative value).
- TextX** Assign this property to specify the number of pixels to move the text to the left (positive value) or right (negative value).
- TextY** Assign this property to specify the number of pixels to move the text upward (negative value) or downward (negative value).

ShowAsButton

Set to True to paint the expand/collapse icons as buttons. Instead of being displayed with just the +/- characters, there is a button frame painted around the icons so that they appear more like buttons.

Data Type: Boolean

ShowFocusRect

Set to False to hide the focus rect that would appear around the text. Note: Text only appears in the control when *ShowText* is *True*.

Data Type: Boolean

ShowText

Set *ShowText* to true if you wish to show the text of the calculated field in the column. This value is not editable and is just a visual indicator. For instance if your calculated field computes the number of detail records, you may wish to display this value in the grid in the *TwwExpandButton* column.

Data Type: Boolean

Added Events

OnAfterCollapse

This event is fired after the end-user collapses the expand button, or the detail grid is automatically collapsed. When the user manipulates the appearance of the master grid, the detail grid is automatically collapsed. For instance if the user advances to the next column in the master grid, or changes the column order, the detail grid is collapsed.

OnAfterExpand

This event is fired after the detail grid is expanded.

OnBeforeCollapse

This event is fired immediately before the detail grid is collapsed.

OnBeforeExpand

This event is fired immediately before the detail grid is expanded

How To

Associate a drop-down panel to the ExpandButton

Associating a panel to an `TwwExpandButton` is similar to associating a grid or inspector. However in some cases when using a panel, you will need to trap for the `vk_tab` character to prevent the focus from moving out of the grid that the `expandbutton` is on.

TwwFilterDialog



InfoPower's TwwFilterDialog is one of the most powerful InfoPower components as it gives your *end-users* the ability to visually filter a table or query, or modify the where clause of an existing SQL statement. Even though the dialog is capable of sophisticated SQL generation, the dialog is simple to use as it completely hides the filtering and SQL details from the end-user. All they need to know is what they are looking for. The dialog does the rest for them!

InfoPower includes AND/OR/NULL support on a single field basis, can show nonmatching records, and has some filter optimizations that allow the filterdialog to take greater advantage of indexes while performing the filter. In addition, the user can search on calculated linked fields and lookupfields in the filterdialog. InfoPower also contains support for picture masks and custom combo boxes.

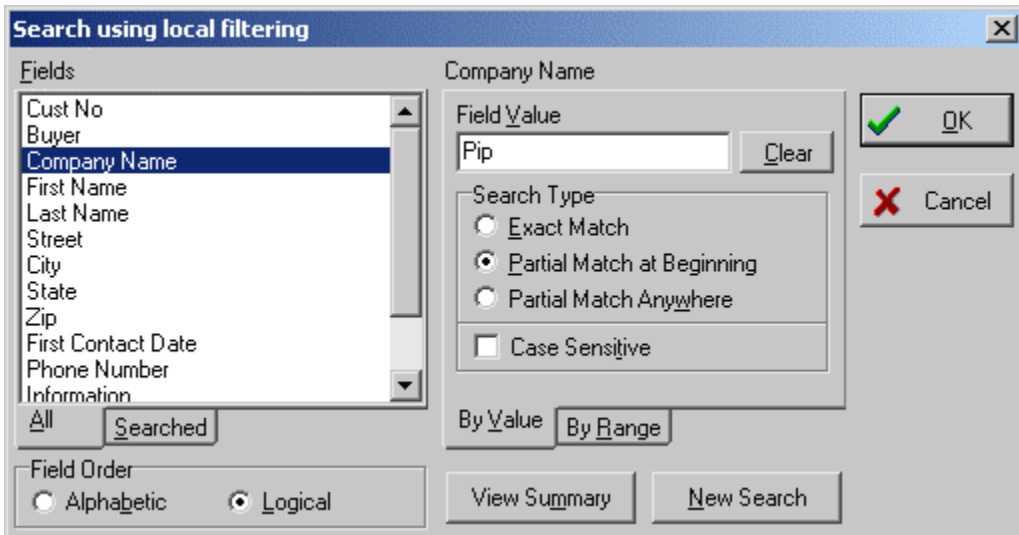


Figure 5.16 - An example of executing the TwwFilterDialog

Ancestor

TComponent
└─TwwCustomDialog

When you execute this dialog box, your end-users can specify a search value, or a range, for any number of fields contained within the table or query referred to by the *DataSource* property. They can also select a specific type of *data match* to be performed within the field, such as “From beginning of field”, “Anywhere within the field”, and “Exact match”. The user can also check the Case Sensitive box to have TwwFilterDialog perform a case-sensitive match.

The following describes the controls on the dialog.

Fields	Displays the list of fields that the user can search on
All Searched	If the <i>All Fields</i> tab page is selected, then the <i>Fields</i> listbox displays all fields. When set to <i>Searched</i> , the <i>Fields</i> listbox displays only fields where the user has assigned a search value.
Field Order	If <i>Alphabetic</i> Field Order is selected then the Fields listbox is ordered alphabetically. If <i>Logical</i> is selected then the fields are displayed in their natural order. If you wish to disable the Field Order Radio Group then set the <i>Options fdShowFieldOrder</i> property for the <i>TwwFilterDialog</i> to <i>False</i> .
View Summary Button	Click this button to view a summary of the current search criteria.
New Search Button	Click this button to start a brand new search.
By Value	Search criteria for currently selected field to be specified by a <i>Field Value</i> . You can clear the <i>Field Value</i> by clicking on the adjacent <i>Clear</i> button. Specify a search type of <i>Exact Match</i> to find exact matches of the <i>Field Value</i> . Specify <i>Partial Match at Beginning</i> and <i>Partial Match Anywhere</i> to find partial matches of the <i>Field Value</i> . Enable the <i>Case Sensitive</i> checkbox to consider case when searching for the <i>Field Value</i> .
By Range	Search Criteria for currently selected field to be specified by a starting and ending range. You can clear the starting and ending ranges by clicking on the adjacent <i>Clear</i> buttons.

Filtering mechanisms used by the TwwFilterDialog

The *TwwFilterDialog*'s allows you to filter the dataset using a wide variety of mechanisms. The filterdialog either uses the dataset's built-in filtering when its *FilterMethod* is set to *fdByFilter*. When the *FilterMethod* is set to *fdByQueryModify*, it will modify the dataset's SQL where clause. For more information on this property see the *TwwFilterDialog FilterMethod* property.

DataSet's Native Filtering

The default filter method is *fdByFilter*. This means the filtering uses the dataset's *Filter* property or its *OnFilterRecord* event to perform the filter, depending upon the *FilterPropertyOptions | DataSetFilterType* property.

The following summarizes the filtering mechanisms when using *fdByFilter*. These mechanisms are controlled by the *FilterPropertyOptions | DataSetFilterType* property

<i>fdUseFilterProp</i>	Uses the dataset's <i>Filter</i> property. The <i>Filter</i> property, although more limited in functionality, can offer significant performance
------------------------	--

benefits when filtering larger datasets. In general this is the fastest type of filter when using `FilterMethod=fdByFilter`.

When set to this value, filtering is not supported on lookupfields, memofields, and calculated fields. In addition it will not support wildcard searches unless the back-end supports the 'like' operator (such as ADO to Microsoft Access which supports the Like keyword in filters).

<code>fdUseOnFilter</code>	Uses the <i>OnFilterRecord</i> event of the dataset. Includes support for lookupfields and memofields. When using ADO or Delphi 5's InterBase objects, it additionally supports filters on calculated fields in your dataset.
<code>fdUseBothFilterTypes</code>	Uses both the dataset's filter property and its <i>OnFilterRecord</i> events. If your back-end supports both filter types, then this should give you the best performance and most capabilities.

Special considerations when filtering on datasets in data modules.

The following warning only applies if you are using `FilterMethod=fdByFilter`, and either `FilterPropertyOptions | DataSetFilterType` of `fdUseOnFilter` or `fdUseBothFilterTypes`.

Warning – If the dataset you are filtering is in a data module, you must place the `TwwFilterDialog` also in the data module, as use of dialog causes the dataset to call a method defined in the filterdialog. Otherwise your filterdialog can be destroyed while your dataset still has an active callback filter. If the filterdialog is destroyed before the dataset, the dataset will be calling random memory and result in runtime access violations.

Special considerations when filtering with Delphi 5's ADO datasets.

Warning – The ADO callback filtering currently has a bug in Delphi 5 when encountering EOF and BOF during the filter. This can manifest itself as a run-time error. In addition, there are also problems when posting a dataset that has been filtered using callback filtering, as the system can enter an infinite loop. The `FilterDialog` uses callback filtering under the following property configuration.

```
FilterMethod = fdByFilter  
FilterOptions.DataSetFilterType = fdUseOnFilter or  
fdUseBothFilterTypes
```

We recommend that you instead use `FilterOptions.DataSetFilterType` of `fdUseFilterProp` when using `TADOTable` or `TADOQuery` to avoid these problems.

Borland/Inprise has been notified of these problems, so we expect that you will be able to use callback filtering in the future once these issues are resolved them.

Remote filtering by modifying the dataset's SQL property

When the `FilterMethod` is set to `fdByQueryModify`, the `TwwFilterDialog` parses the SQL of the dataset, and then replaces its where clause based on the user entered criteria. The revised query is then re-executed so that remote filtering is performed.

Though functionally very similar to local filtering, the actual mechanism of filtering is not performed locally, but instead at the back-end. The back-end can then efficiently perform the search by utilizing available indexes. This filter method also has the advantage of reducing network traffic since the filtering is performed on the same machine that contains the data.

InfoPower's FilterDialog is designed to recognize the new ADO and InterBase dataset objects. However if you are using a 3rd party engine which has different property names or property types for its dataset, then you may need to use the *OnInitTempDataSet* event to initialize any additional properties for your dataset type.

Use the *SQLTables* property to pre-define which tables the filterdialog should extract its list of field names from. If this property is uninitialized then the filterdialog will attempt to parse the SQL to retrieve this information. However the filterdialog's parsing only handles simple SQL statements so it may not be useful for your specified SQL. In general you should always set the *SQLTables* property so that you will not be tied to the filterdialog's limited parsing abilities.

Use the *FieldsFetchMethod* to determine the logic InfoPower uses to retrieve the field information. You may wish to set this property to *fmUseTFields* if you want the FilterDialog to simply gather its field definitions from the dataset's field properties. By doing so, the filterdialog will not need to parse the SQL for its list of tables, nor will it need to query the database for its list of fields for each table. As a result it may improve the speed of the initial display of the dialog. Note that the *SQLTables* property is ignored when *FieldsFetchMethod* is set to *fmUseTFields*.

Added Properties

Caption

This property contains a text value that is displayed in the dialog window's title bar. The default value is blank.

Data Type: String

DataSource

Name of datasource you are filtering or querying

Data Type: TDataSource

Valid Values: Any TDataSource Component

DefaultField

Name of field the dialog initially will select. If the user has previously selected some search criteria, then this property is ignored.

Data Type: String

Valid Values: Valid Field

DefaultFilterBy

The user can filter by specifying a range of values, or filter by a string value. This property controls which search type the dialog initially defaults to.

Data Type: TwwDefaultFilterBy

Valid Values: fdFilterByRange, fdFilterByValue, fdSmartFilter

<i>fdFilterByValue</i>	The dialog defaults to the <i>By Value</i> tab page (search criteria is defined by a string value).
<i>fdFilterByRange</i>	The dialog defaults to the <i>By Range</i> tab page (search criteria is defined by a upper and/or lower range).
<i>fdSmartFilter</i>	The dialog examines the field type of the currently selected field, and automatically uses the most appropriate search type tab page. For numerics and dates it uses the <i>By Range</i> tab page, and for other types it uses the <i>By Value</i> tab page.

DefaultMatchType

Use this property to change the initial selection of the Search Type radio button.

This property defaults to fdMatchStart.

Data Type: TwwDefaultMatchType

Valid Values: fdMatchStart, fdMatchAny, fdMatchExact

DlgHeight

Use this property to change the height (in pixels) of the filter dialog. By increasing the height, the dialog can display more fields at a time.

Data Type: integer

FieldOperators

Use this property to customize the field operators that are used to specify “and”, “or”, and “null” type operations.

Data Type: TwwFieldOperators

There are 3 operators that you can use in the FilterDialog:

AndChar	Use this property to customize the “and” keyword that will be used in the FilterDialog. Default is “and”. Data Type: String
NullChar	Use this property to customize the “null” keyword that will be used in the FilterDialog. Default is “null”. Data Type: String
OrChar	Use this property to customize the “or” keyword that will be used in the FilterDialog. Default is “or”. Data Type: String

FieldsFetchMethod

Use the *FieldsFetchMethod* to determine the logic InfoPower uses to retrieve the field information. This property is only applicable with FilterMethod =fdByQueryModify. By

default InfoPower parses the SQL and searches for the tables referenced by the SQL. You can bypass this parsing by assigning the `SQLTables` property. After the list of tables is generated, it queries the database for a list of fields in the table. If this property is set to `fmUseTTable` then it uses a temporary `TTable` component to retrieve the field information. If set to `fmUseTFields` then it extracts the field information from the datasource associated with the filterdialog. Otherwise it creates a copy of the original dataset to gather the field information.

Data Type: `TwwFilterFieldsFetchType`

Valid Values: `fmUseTTable`, `fmUseSQL`, `fmUseTFields`

fmUseTTable For each table component parsed or defined by the `SQLTables` property, the filterdialog uses a `TTable` component to extract the field information. If using a non BDE dataset, then the component ignores this setting and instead uses `fmUseSQL`.

fmUseSQL Create a copy of the dataset and set its `SQL` property to gather the field information.

fmUseTFields You may wish to set this property to `fmUseTFields` if you want the `FilterDialog` to simply gather its field definitions from the dataset's field properties. By doing so, the filterdialog will not need to parse the SQL for its list of tables, nor will it need to query the database for its list of fields for each table. As a result it may improve the speed of the initial display of the dialog. Note that the `SQLTables` property is ignored when `FieldsFetchMethod` is set to `fmUseTFields`.

FilterMethod

Use this property to change the filtering method used to select the records that matches the user's criteria. The only valid value for filtering a `TwwTable` or `TwwQBE` is the `fdByFilter`. When filtering a `TwwQuery`, you can use either `fdByFilter`, or `fdByQueryModify`. This property defaults to `fdByFilter`.

Data Type: `TwwFilterMethod`

Valid Values: `fdByFilter`, `fdByQueryModify`

fdByFilter When using local filtering on a `Query` or `QBE`, the query is not re-executed, but simply re-filtered. This means that the back-end does not need to do any additional processing. If you using this `FilterMethod` on a `TQuery`, you should set your `RequestLive` property to `False`. Local filtering on tables guarantees a live editable view of the data.

fdByQueryModify Remote filtering is performed by re-executing the SQL in a developer defined `TQuery`. The query's SQL string is modified so that it's `where` clause reflects the user specified criteria. See also the section preceding the `FilterDialog`'s property reference for further information on this filtering method.

FilterOptimization

This property has been added to increase the performance of the FilterDialog when operating against a TwwTable. It will allow the FilterDialog to use no indexes, the active index, or all indexes while performing the filter. FilterOptimization will only improve performance with MatchStart or MatchExact, and not with MatchAny.

Data Type: TwwFilterOptimization

Valid Values: fdNone, fdUseAllIndexes, and fdUseActiveIndex

<i>fdNone</i>	This is the default.
<i>fdUseAllIndexes</i>	This setting allows the component to switch the index of the filtered field to improve performance. The side effect of this is that the order of your displayed data will now be in the order of your filtered field.
<i>fdUseActiveIndex</i>	This property will allow the TwwFilterDialog to use the currently active index while filtering. This will only improve performance when the user searches on the first indexed field.

FilterPropertyOptions

This property defines the filtering mechanism used when *FilterMethod=fdByFilter*. The sub-properties *LikeSupportsUpperKeyword*, *LikeWildcardChar*, *UseBracketsAroundFields*, and *UseLikeOperator* only apply if the DataSetFilterType is set to *fdUseFilterProp* or *fdUseBothFilterTypes*.

Data Type: TwwFilterPropertyOptions

Valid Values: fdNone, fdUseAllIndexes, and fdUseActiveIndex

DataSetFilterType	See the 'Filtering mechanisms used by the TwwFilterDialog' section for a detailed reference of this property. This section appears in the TwwFilterDialog component description. Data Type: TwwDatasetFilterType
LikeSupportsUpperKeyword	Set this property to True if your back-end supports the Upper SQL keyword, and you wish to allow the end-user to select the case sensitivity of their filters. This property is only applicable if <i>UseLikeOperator</i> is set to True. This property defaults to False.
LikeWildcardChar	Assign this property to customize the wildcard character used in the filter. This property defaults to '%'. Modify this property if your back-end uses a different wildcard character. This property is only applicable if <i>UseLikeOperator</i> is set to True Data Type: Character
UseBracketsAroundFields	Set this property to determine if the filter expression generated by the filterdialog should put brackets around the field names. Generally brackets are required by the back-

end. However with Delphi 5's new InterBase data objects, you will need to set this property to False.

UseLikeOperator

Set to True if your database engine supports the *Like* operator when specifying a Filter. The TDataSet Filter property uses a different syntax depending on the back-end. For instance, ADO using the Microsoft 4.0 Jet OLE DB provider will support the like keyword in the filter expression. Thus you should set your *FilterPropertyOptions.UseLikeOperator* property to True. If your back-end or provider does not support the *Like* keyword in the filter expression, then set this property to False. For instance when tied to a TBDEDataSet (i.e. TTable, TQuery), you should set this property false. You may need to experiment to see what your back-end supports. We recommend you first try setting this property to True, as your capabilities will be greater if your back-end supports this.

Note: Some back-ends require a different syntax for the TDataSet filter property. If using Delphi 5's new InterBase objects, you should change the default property values to the following:

LikeSupportsUpperKeyword - True
LikeWildCardChar - %
UseBracketsAroundFields - False
UseLikeOperator - True.

If using Delphi 5 ADO data access components, you should set UseLikeOperator to True.

SupportsUpperKeyword - False
LikeWildCardChar - %
UseBracketsAroundFields - True
UseLikeOperator - True

OnFilterPropertyOptions

This property allows you to customize certain behavior of the callback filtering. Callback filtering is enabled when FilterMethod=fdByFilter, and FilterPropertyOptions | DataSetFilterType is set to fdUseOnFilter or fdUseBothFilterTypes.

fdClearWhenNoCriteria Set this to false if you wish to prevent the automatic canceling of the callback filter when the user has entered no criteria. This can be useful if you are using the OnAcceptFilterRecord, as when a filter is cancelled the OnAcceptFilterRecord event will no longer fire. This property defaults to true.

fdClearWhenCloseDataSet Set this to false if you wish to prevent the clearing of the filter when the dataset is closed. This property defaults to true.

Options

Use this property to change what is displayed shown when the dialog box is executed.

Data Type: Set of *TwwFilterDialogOption*

Valid Values: *fdCaseSensitive*, *fdShowCaseSensitive*, *fdShowOKCancel*, *fdShowViewSummary*, *fdShowFieldOrder*, *fdShowValueRangeTab*, *fdShowNonMatching*, *fdHidePartialAnywhere*, *fdDisableDateTimePicker*, and *fdSizeable*.

<i>fdCaseSensitive</i>	Initial value of dialog's case sensitive checkbox
<i>fdShowCaseSensitive</i>	If True, the case sensitive checkbox will be shown in the dialog.
<i>fdShowOKCancel</i>	If True, the OK and Cancel buttons are displayed in the dialog
<i>fdShowViewSummary</i>	If True, the Show Summary button is shown in the dialog
<i>fdShowFieldOrder</i>	If True, then the Field Order radio button is shown in the dialog
<i>fdShowValueRangeTab</i>	If True, then the tab page that allows the user to select either ByRange or ByValue is shown.
<i>fdShowNonMatching</i>	NOT support. When <i>fdShowNonMatching</i> is set to True, then a checkbox appears in the Filterdialog which will allow the user to find the information that doesn't fit a particular condition.
<i>fdHidePartialAnywhere</i>	When True, the Partial Match Anywhere Radio Button is hidden in the dialog. Defaults to False.
<i>fdDisableDateTimePicker</i>	When True, the <i>TwwDBDateTimePickers</i> controls are not used on date/time fields. Defaults to False.
<i>fdSizeable</i>	When True, the popup filter dialog will be resizable. Defaults to False.

QueryFormatDateMode

This property determines how DateTime values get sent to the SQL back end. For instance if the back-end expects dates to be in the format month,day,year, then set this property to *qfdMonthDayYear*. For further customization then these three values provide, use the *OnEncodeDateTime* event.

Data Type: *TwwQueryFormatDateMode*

Valid Values: *qfdMonthDayYear*, *qfdDayMonthYear*, *qfdYearMonthDay*

Rounding

In some instances when local filtering on numeric values, rounding issues can come into play when doing range filters. To minimize the effect of rounding, you can tell the component to allow some tolerance between the database field value and the ranges that are being checked.

Note:/ Rounding only affects local callback filtering (FilterMethod=fdByFilter, FilterPropertyOptions.DataSetFilterType=fdUseOnFilter). Otherwise these issues are handled by the database driver. The following sub-properties are available:

Epsilon	See the RoundingMethod property for a description of how this property is used. Data Type: Double
RoundingMethod	If set to <i>fdrmNone</i> , then no rounding is performed. If set to <i>fdrmFixed</i> , then the <i>epsilon</i> value is subtracted from the starting range value, and added to the ending range value. Setting this property to <i>fdrmRelative</i> currently has no functionality but is provided for future enhancements. Data Type: TwwFilterDialogRoundingMethod

SelectedFields

This property determines which fields the user can filter on. The TwwFilterDialog uses this property to fill in the Fields listbox. When this property is empty, then all fields are selected.



Figure 5.17 -TwwFilterDialog's Selected Fields property dialog

Click on the *Add Fields* button to add additional fields to display in the filter dialog. Click on *Remove Fields* to remove the selected fields from the filter dialog.

When performing remote filtering a *TwwQuery* (*FilterMethod* = *fdByQueryModify*), the user may wish to specify search criteria for fields that don't actually exist in your *TQuery* field structure definition. In these cases InfoPower will use the field name as the display label. You can override these display labels through this property. To modify the display labels for fields in your field structure definition, use Delphi's *TDataSet* Fields Editor, as InfoPower will not allow you to modify them through this property.

The following is an example of an SQL statement where the field structure definition does not include any fields in IP4INV.DB. The *TwwFilterDialog* however will allow the user to search on these fields when using remote filtering.

```
SELECT DISTINCT
  IP4CUST."Customer No",
  IP4CUST."Buyer" ,
  IP4CUST."Company Name"
FROM "IP4CUST.DB" IP4CUST , "IP4INV.DB" IP4INV
WHERE (IP4CUST."CUSTOMER NO"=IP4INV."CUSTOMER NO")
```

Data Type: TStrings

Valid Values: Array of field names.

SortBy

The list of fields can be sorted alphabetically by field name or logically by field order.

If this property is set to *fdSortByFieldName*, then the dialog initially displays the list of fields alphabetically. If set to *fdSortByFieldNo*, then the dialog displays the fields in their logical order.

Data Type: *TwwFilterDialogSort*

Valid Values: *fdSortByFieldNo*, *fdSortByFieldName*

SQLPropertyName

This property is used when *FilterMethod*=*fdByQueryModify*. In order for the filterdialog to support datasets who define their sql with a different property name, you must set the *SQLPropertyName* to the name of this property.

InfoPower extends the filterdialog to support dataset types that do not use the property name 'SQL' to define their sql. In addition, the data type of the sql property is no longer required to be a *TStringList*. This allows the filterdialog to remotely filter dataset types such as *TADODataSet*, *TClientDataSet*, and other 3rd party dataset types that it did not previously support. This greatly improves the performance in these situations.

See the how-to topic on *Remotely filter a TClientDataSet* if you are using a *TClientDataSet*.

Data Type: String

SQLTables

This property is used when *FilterMethod*=*fdByQueryModify*, and *FieldsFetchMethod* is set to *fmUseTTable* or *fmUseSQL*.

Use the *SQLTables* property to pre-define which tables the filterdialog should extract its list of field names from. If this property is uninitialized then the filterdialog will attempt to parse the SQL to retrieve this information. However the filterdialog's parsing only handles simple SQL statements so it may not be useful for your specified SQL. In general you should always set the *SQLTables* property so that you will not be tied to the filterdialog's limited parsing abilities.

Data Type: TCollection of TwvSQLTablesCollectionItem. Each collection item contains the properties *TableName* and *TableAliasName*. If *TableAliasName* is omitted, then it defaults to the *TableName* property value.

SQLUpperString

Assign this property to change the sql keyword used to upper case a field value. When this property is unassigned, the 'UPPER' keyword is used. This property is only relevant if you are performing case insensitive filtering with the *FilterMethod* set to *fdByQueryModify*. Do not change this value unless you are certain that your back-end supports a different sql keyword for uppercase.

UpperRangePadChar

When searching using the *ByRange* tab on a string field, the upper range field needs to be padded with the highest valid Ascii character internally. So example with this property set to it's default of 122, then if the end-user searches from A to C it will filter on the range from A to Czzzzzz so that all strings that start with A or C are found. You should not normally need to modify this value.

Required property assignments

DataSource

Added Events

Some of the following events pass a handle to the form containing all of the components of the dialog. To see what objects are contained within this editing form, open up *wwfldlg.pas* in the *InfoPower* source sub-directory. If you do not have the source code version of *InfoPower*, then perform the steps in Chapter 4's topic "Determining the object names of the controls contained in an *InfoPower* dialog" on the *wwfldlg.dfm* file contained in the *InfoPower* lib directory.

If you want to customize any of the objects contained by the form you can use the *OnInitDialog* event. However if all you are trying to do is to change the labels and hints, then use the *TwvIntl | FilterDialog* property.

OnAcceptFilterRecord

This event is fired when using *FilterMethod=fdByFilter*, and either *FilterPropertyOptions | DataSetFilterType* of *fdUseOnFilter* or *fdUseBothFilterTypes*. See also the notes in the filterdialog's component description for further note on ADO dataset filtering.

You can use this event for your own needs if you wish to have additional callback filtering performed before the filterdialog's default callback filtering.

<i>Sender</i> : TObject	TwwFilterDialog associated with this event.
<i>DataSet</i> : TDataSet	<i>Dataset that is being filtered</i>
<i>var Accept</i> : boolean	<i>Set to false</i> to reject the current record from being filtered. The filterdialog will not evaluate the record. If you set this property to <i>true</i> and wish for the filterdialog to do no additional examination of the record, then also set the <i>DefaultFiltering</i> property to <i>False</i> .
<i>var DefaultFiltering</i> : boolean	If you wish for the filterdialog to do no additional examination of the record, then set the <i>DefaultFiltering</i> property to <i>false</i> .

OnDialogSummary

This event is fired when the end-user clicks on the View Summary Button in the dialog. Use this event to display your own summary dialog based on the information in the AFieldInfo list.

<i>Sender</i> : TObject	TwwFilterDialog associated with this event.
<i>AFieldInfo</i> : TList	TList of TwwFieldInfo. This list defines the current filter criteria.
<i>var DoDefault</i> : boolean	Set to <i>False</i> when displaying your own custom dialog.

OnEncodeDateTime

This event allows you to change the format of the date within the SQL. InfoPower defaults to formatting based on the *QueryFormatDateMode* property. If you require further customization of the format for your back-end, then use this event to set the *FormattedDateStr* parameter.

OnEncodeValue

This event allows you to change how the value is formatted when the SQL is being generated.

OnExecuteSQL

This method is maintained for backwards compatibility. Use the dataset's BeforeOpen event for new applications.

When using *FilterMethod = fdByQueryModify*, this event is fired before the query is executed. Use this event if you wish to customize the query in some way, such as adding an OrderBy clause. This event only applies if your filterdialog is filtering a TQuery.

Tip: If you wish to view the SQL that was generated (for Debug reasons), then you can put the following code in this event to view the actual SQL string that is generated before it is actually executed.

```
procedure TForm1.wvFilterDialog1ExecutesSQL(Dialog: TwvFilterDlg;
  Query: TQuery);
  var i:integer;
begin
  for i:=0 to Query.SQL.Count-1 do
    ShowMessage('Line #' + IntToStr(i+1) + ': ' + Query.SQL.Strings[i]);
end;
```

OnInitDialog

Allows you to customize the filter dialog box or perform some action during the initialization of the dialog box. For basic customization of the labels and hints, use the TwvIntl component. For all other customization, use this event. This event is fired after the dialog box is created, but before it's displayed on the screen. This gives you access to all the components in the dialog, allowing you to completely customize every aspect of the dialog during program execution. When using this event, your code must reference *wvfltDlg* in your source file's Uses clause. For example, you can modify the control's properties, define custom events, etc.

Example: The following code disables the hint on the user-defined button.

```
procedure TForm1.wvFilterDialog1InitDialog(Dialog: TwvFilterDlg);
begin
  Dialog.ViewButton.ShowHint := False;
end;
```

OnInitTempDataSet

This event is only fired when FilterMethod=fdByQueryModify, and FieldsFetchMethod is set to fmUseSQL. When you call the execute method of the filterdialog,, it creates a temporary dataset and then calls the GetFieldNames method for this temporary dataset. However this temporary dataset may need additional properties set, depending upon if you are using any 3rd party database engines. If you are, then set the properties of the temporary dataset in this event.

<i>Sender:</i> TObject	TwvFilterDialog associated with this event.
<i>OrigDataSet:</i> TDataSet	Dataset that is being <i>filtered</i>
<i>TempDataSet:</i> TDataSet	Temporary dataset created to retrieve the list of fields

OnSelectField

This event allows you to define custom combo-boxes or picture masks within the end-user filtering dialog. The combo-boxes are used when the user is doing an *Exact* or a *Partial Match at Beginning* search.

<i>Sender:</i> TObject	Actual dialog that is retrieving the user input. You can cast this to a TwvFilterDlg to access all of the controls in this dialog. If you cast it then be sure to add <i>wvfltDlg</i> to your
------------------------	---

form's *uses* clause. See the *OnInitDialog* for the names of all the components on this dialog.

<i>FieldName</i> : string	Name of the new field being selected
<i>PictureMask</i> : string	Set this to the picture mask you want to use for the new field.
<i>ComboList</i> : TStringList	Add entries to this TStringList to define a custom combobox for the field.

Example: The following code defines a combo-box list for the 'Buyer' field of the FilterDialog. When the user selects this field doing a value search, a combo-box will appear that has the values "Yes", and "No" in the drop-down list.

```
procedure TFilterDialogForm.wvFilterDialog1SelectField(Sender: TObject;
  FieldName: String; var PictureMask: String; ComboList: TStringList);
begin
  if FieldName = 'Buyer' then begin
    ComboList.Add('Yes');
    ComboList.Add('No');
  end
end;
```

Example: The following code defines a mapped combo-box list, which allows the select from a list of descriptive strings and the filter will automatically filter on the database field value. For instance, consider the case where your database field allowed for 3 possible integer values (0,1,2), and the represented the strings 'Visa', 'MasterCard', 'American Express'. You can use the following code to allow the user to select from the more meaningful descriptive text, but still filter based on the integer value.

```
procedure TFilterDialogForm.wvFilterDialog1SelectField(Sender: TObject;
  FieldName: String; var PictureMask: String; ComboList: TStringList);
var FilterCombo: TwwDBComboBox;
begin
  if (FieldName = 'Buyer') then begin
    FilterCombo:= TwwFilterDialog(Sender).Form.FilterValueCombo;
    FilterCombo.MapList:= True;
    ComboList.Add('Visa' + #9 + '0');
    ComboList.Add('MasterCard' + #9 + '1');
    ComboList.Add('American Express' + #9 + '2');
  end
end;
```

Added Methods

AddFieldInfo

Use this method if you wish to add filter conditions to the FilterDialog using code. This method creates and returns a TwwFieldInfo data type, which is automatically added to the internal *FieldInfo* list. See the example under the *ApplyFilter* method.

```
TwwFieldInfo = class
public
  FieldName: string;
  DisplayLabel: string;
  MatchType: TwwFilterMatchType;
  FilterValue: string;
```



```

        MinValue: string;
        MaxValue: string;
        CaseSensitive: boolean;
        NonMatching: boolean;
    end;

    TwwFilterMatchType = (fdMatchStart, fdMatchAny, fdMatchExact,
        fdMatchRange, fdMatchNone);

```

ApplyFilter

Use this method to apply the filter based on changes that have been made to the FieldInfo criteria of the TwwFilterDialog. See the form savefilt.pas and the unit wwsavflt.pas in the main demonstration program for a complete example of applying filters at runtime without the filter dialog appearing.

Example: This example demonstrates how to change the filter criteria of a TwwFilterDialog at runtime with code.

```

procedure TFilterDialogForm.Button1Click(Sender: TObject);
begin
    with wwFilterDialog1 do begin
        ClearFilter;
        with AddFieldInfo do begin
            FieldName:= 'Last Name';
            Displaylabel:= 'Last Name';
            MatchType:= fdMatchStart;
            FilterValue:= 'R';
            MinValue:= '';
            MaxValue:= '';
            CaseSensitive:= False;
        end;
        ApplyFilter;
    end;
end;

```

ClearFilter

Use this method to clear the filter currently used in the TwwFilterDialog. Remember to use the ApplyFilter method to actually make the changes.

Execute

Display the Visual filtering dialog box to the end-user.

ExecuteDialog

A more flexible version of the TwwFilterDialog Execute method when using FilterMethod=fdByQueryModify. If using FilterMethod=fdByFilter, then it the parameters are ignored.

```

Function ExecuteDialog(ExecuteQuery: boolean = True;
    ReturnWhereClause: TStrings = nil): boolean;

```

<i>ExecuteQuery</i>	When false, the query associated with the filterdialog is not re-executed.
---------------------	--

ReturnWhereClause Assign this parameter if you wish to retrieve the where clause that the filter dialog has computed based on the user's entered criteria.

How To

Remotely filter a TClientDataSet

The following are the steps required to remotely filter a clientdataset so that filter is performed on the server and not locally.

1. Set the following properties for the filterdialog
FilterMethod = fdByQueryModify
FieldsFetchMethod = fmUseTFields
SQLPropertyName = CommandText
2. Set your TClientDataSet so that the CommandText property is set to query the table, such as...
Select * from Employee
3. Set your provider's Options property so that poAllowCommandText to True. This allows the TClientDataSet to modify the provider's SQL with its CommandText property.

Add a Custom Combobox to the FilterDialog

See the documentation and examples under the OnSelectField event.

Invoke the FilterDialog

Call the Execute method to bring up the filter dialog.

```
wwFilterDialog1.Execute;
```

Use the TwwSearchDialog with the wwFilterDialog

When using the FilterDialog on a table, you may wish to apply this same filter to another table. For instance this would be useful when using the SearchDialog with the optional *ShadowSearchTable*, as it would maintain consistency between the search dialog's view and the calling form's view. The following shows you how you can apply the filterdialog's filter to the ShadowSearchTable of the SearchDialog.

1. Add a new TwwFilterDialog component to your form and set the following properties
DataSource = CustomerTableDS
Name = CustomerFilterDlg
FilterMethod = fdByFilter
2. Add a new TButton component to your form and attach code to its *OnClick* event:

```
Procedure TFilterDialogForm.BitBtn1Click(Sender: TObject)  
begin
```

```
CustomerFilterDlg.Execute;  
CustomerShadowTable.OnFilter := CustomerTable.OnFilter;  
end;
```

Loading and saving filters

See the InfoPower demo for an example on how to load and save filters using the `wwFilterDialog`.

Expanding the height of the dialog

Increase the value of the `DlgHeight` property

Determine if the user selected any search criteria

If the `FieldInfo.Count` property is greater than 0, then the user has selected a search criteria.

End-user Resizing of FilterDialog

Set *Options* | *fdSizeable* to True.

Caution

Make sure that your field display labels are unique. InfoPower's `FilterDialog` uses the field titles to fill the fields listbox. If there are duplicates, the user's filtering specification becomes ambiguous and may not function as expected.

TwwIncrementalSearch



TwwIncrementalSearch is a visual interface component that provides your end-users with a means to incrementally search for values. As the end-user enters characters into the edit box, the component performs a search operation based on the characters currently in the edit box, moving to the record that contains the closest match

When searching against a Table, the component automatically uses the index to enhance performance. See the TwwTable property *NarrowSearch* and *SyncSQLByRange* to further customize how InfoPower performs incremental searching on a Table.

When searching against a Query or QBE result, the component performs a sequential search on the field defined by the *SearchField* property. The search is always case insensitive against a query or QBE.



Figure 5.18 - An active TwwIncrementalSearch component looks just like a normal edit box.

Ancestor

TCustomEdit.

Required supporting components

TDataSource

Added properties

DataSource

This property contains the name of a TDataSource component that the search should be applied to. The default value is blank.

Data Type: TDataSource

Valid Values: Valid DataSource component name

Frame

See the topic “Key properties and events for custom framing” in chapter 4 for information on this property.

Data Type: TwwEditFrame

PictureMask

This property allows you to enter a picture mask to validate and convert the entered text to the desired format. For instance you can set the picture mask to “&*?” and the control will only allow letters to be entered, and will automatically capitalize the first letter. Please reference chapter 4, *Selecting a Picture Mask* for details on this property.

Note: If you wish for the control to automatically use the picture mask already defined for the database field, then leave this property blank, and set the *PictureMaskFromField* property to True.

Data Type: String

Valid Values: Valid picture mask string

PictureMaskAutoFill

When you enter a picture mask that consists of auto-fill characters, setting this property to True will activate the auto-fill capabilities of the picture mask. When False, all auto-fill capabilities of the picture masks defined for the field are deactivated.

Data Type: Boolean

PictureMaskFromField

Setting this property to True will allow the control to automatically use the picture mask defined for the database field. The *TwwIncrementalSearch*'s *PictureMask* property must also be set to blank for this property to have an effect, as the component's *PictureMask* property has precedence.

Data Type: Boolean

ShowMatchText

Set this property to True if you want the incremental search component to display the matching field's value directly in this control. By default this property is False.

Data Type: Boolean

SearchDelay

TwwIncrementalSearch supports a search delay for its incremental searching. This property controls how many milliseconds to wait before beginning the search for the user's entered text. The purpose of this property is to reduce the number of searches that are performed as the user enters characters. Setting this to a larger value may improve your performance as fewer searches will need to be performed. Setting this to a smaller value will cause the search to begin more quickly. This property defaults to 0, which tells the control to determine the best delay. Currently 200 milliseconds is used by the control for the delay.

Data Type: Integer

SearchField

In some cases the search component is not able to determine the correct field that it should search on. This can occur when using dBASE expression indexes, or if you are using a multi-field index and you may want to search on the 2nd or 3rd field instead of the default 1st field.

Ambiguity can also occur if you are incrementally searching a dataset that has no indexes, such as a TwwQuery, TwwQBE, or a TwwClientDataSet.

(See TwwDBLookupCombo for an example of how to use this property.)

Data Type: String

Valid Values: Valid field name

Modified properties

Text

No longer published, but is now a runtime only property. You can still use this property in your code.

Required property assignments

DataSource, SearchField (if no index is active, or you are using an xBASE expression index).

Added Events

OnAfterSearch

This event is executed after the incremental search has repositioned the table according to the user's entered text.

Added Methods

Clear

Clear the incremental search component back to blank.

FindValue

Call this method if you wish to force the incremental search component to use the current text value and perform the search. The component automatically performs incremental searching when the end-users types in text, but if you explicitly set the text value the search is not performed.

Tips

- ◆ This component is especially useful when combined with the TwwKeyCombo component, which allows end-users to select the active table index.
- ◆ If you have a choice, use case insensitive indexes in your tables to make incremental searching more user-friendly. If you do not have a choice use picture masks to automatically convert the input to the proper case.

TwwIntl



The TwwIntl component is a non-visual component that provides you with a centralized way of changing the attributes of InfoPower's end-user dialog boxes. You can control the captions, hints, and even the style of buttons to use. This component will greatly assist those internationalizing their application into other languages.

Ancestor

TComponent.

Required supporting components

None.

Added Properties

ADO

With ADO Datasets when locating on partial match or exact match searches the locate method is much faster. So setting UseLocateWhenFindingValue to true will result in performance enhancements for search type operations in InfoPower components.

BtnCancelCaption

Caption to use in Cancel buttons

BtnOKCaption

Caption to use in OK buttons

CheckBoxInGridStyle

Use this property to choose the style of the checkboxes that appear in all the grids.

Data Type: TwwCheckBoxInGridStyle

Valid Values: cbStyleAuto, cbStyleCheckmark, or cbStyleXmark

<i>cbStyleAuto</i>	Style of checkbox is dependent on operating system.
<i>cbStyleCheckmark</i>	Checkmark style of checkbox.
<i>cbStyleXmark</i>	X style of checkbox.

Connected

Toggle this property to True to activate this component's properties so that they are used by the InfoPower dialogs.

Data Type: Boolean

DefaultEpochYear

This property defines the default epoch date when creating new TwwDateTimePicker controls. The property is used to determine how 2 digit years resolve to 4 digit years. This property defaults to 1950, which will translate 2 digit years (xx) less than 50 to 20xx', and years greater than 50 to 19xx.

Data Type: integer

Valid Values: Valid year greater than 1900

DialogFontStyle

Use this property to choose the style of the fonts that appear in InfoPower's dialogs. If you need to change any other attributes of a given dialog's Font, then use the corresponding dialog's OnInitDialog event.

Data Type: TFontStyles

Valid Values: fsBold, fsItalic, fsUnderline, or fsStrikeOut

<i>fsBold</i>	Font style of bold.
<i>fsItalic</i>	Font style of italics.
<i>fsUnderline</i>	Font style of underline.
<i>fsStrikeOut</i>	Font style of strikeout.

FilterDialog

Captions and Hints for the TwwFilterDialog component

FilterMemoSize

This property defines the number of bytes to allocate for temporary data storage when performing callback filtering. This property is only relevant if using TwwTable, TwwQuery, or TwwStoredProc.

Data Type: Integer

IniFileName

This property defines the value use when a grid's IniAttributes | FileName property is blank. This property is ignored if the IniAttributes | FileName property is already assigned for the grid control. If IniFileName is also blank, the grid will compute the FileName based on the application's executable name.

Data Type: String

LocateDialog

Captions and Hints for the TwwLocateDialog component

MonthCalendar

Captions and Dialog messages for the TwwDBMonthCalendar

Navigator

Captions and Hints for the TwwDBNavigator

OKCancelBitmapped

True if the OK and Cancel buttons use a bitmap

Data Type: Boolean

RegistrationNo

Displays your registration number.

RichEdit

Captions, Menu items, and Hints for the TwwDBRichEdit component.

SearchDialog

Captions and Hints for the TwwSearchDialog, TwwLookupDialog, and TwwDBLookupComboDlg components

UseLocateMethodForSearch

Setting this property to False, will make the component use the FindNearest method instead of the Locate method when incremental searching on tables. Because Delphi recommends using the Locate method over the FindNearest method, we recommend leaving this property True.

UserMessages

User messages generated by InfoPower

VersionInfoPower

Displays the current version of InfoPower

Required property assignments

None

Added Events

OnPerformCustomSearch

When using a large search table from a remote server, the performance of the incremental searching can significantly degrade. To resolve this issue, InfoPower adds a new event where you can control the specific action that takes place after the user types a character. In particular the custom action can update the query to only return the records that you are interested in. When using this event, your code is responsible for manipulating the lookup table based on the parameter values passed in. See the TwwDBLookupCombo OnPerformCustomSearch event for a description of the events parameters.

OnValidationErrorUsingMask

Write an OnValidationErrorUsingMask event handler to perform any custom action after the user tries to leave a cell or edit control with a value that does not satisfy the picture mask constraints assigned for the cell. The default behavior is to raise an exception with the message “Invalid input value. Use escape key to abandon changes”.

<u>Parameter</u>	<u>Description</u>
<i>Sender</i> :TObject	The TwwDataInspector associated with this event.
<i>Field</i> :TField	TwwInspectorItem whose edited value does not satisfy the picture mask constraints.
<i>Msg</i> :String	You can set this value to change the actual message used by the default error handler.
<i>var DoDefault</i> :Boolean	Set this to False to prevent the default handler from executing. The default error handler raises an exception with the message defined by Msg.

Here is an example of code attached to the TwwIntl | *OnValidationErrorUsingMask* event. Note that you will need to also set the TwwIntl *Connected* property to true, and place this component in your main form.

```
procedure TForm1.wwIntl1ValidationErrorUsingMask(Sender: TObject;  
  Field: TField; var Msg: String; var DoDefault: Boolean);  
begin  
  Msg:= 'Invalid input for field ' + Field.fieldname;  
  DoDefault:=True;  
end;
```

How To

Modify the labels and hints in an InfoPower dialog

Customizing the dialog labels and hints is accomplished via the following steps:

1. Drop an InfoPower TwwIntl component on your main form
2. Modify the properties you wish to change.
3. Set the component's Connect property to True.

TwwKeyCombo



TwwKeyCombo is a visual interface component that provides your end-users with an easy means of changing the current display order of data being retrieved from an indexed table. When the user selects the TwwKeyCombo component, its drop-down selection list is populated with the DisplayNames of all indexes available for the table it's assigned to. Index fields that are not selected for display in the target visual interface component are not shown in the selection list. When the user selects one of the available table indexes, the table's access path index is changed to the user-selected index.

If there is more than one index that corresponds to a given field, this component places a priority on using a case insensitive index.

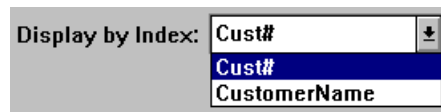


Figure 5.19 - The visual interface portion of a live TwwKeyCombo component with the drop-down list activated.

Ancestor

TwwDBCUSTOMCOMBOBOX.

Required supporting components

TDataSource.

Added Properties

ButtonEffects

See the topic “Key properties for enabling custom button effects in the edit controls” in chapter 4 for information on this property.

Data Type: TwwButtonEffects

ButtonGlyph, ButtonStyle, ButtonWidth

See the TwwDBCUSTOMCOMBOBOX component for a description of these properties.

DataSource

This property contains the name of a TDataSource component that is to be used for the acquisition of data. The default value is blank.

Data Type: TDataSource

Valid Values: Valid DataSource component name

Frame

See the topic “Key properties and events for custom framing” in chapter 4 for information on this property.

Data Type: TwwEditFrame

PrimaryKeyName

Used only when *ShowAllIndexes* is True. This property contains the text that is displayed for the primary index name. The default value is “PrimaryKey”.

Data Type: String

ShowAllIndexes

When True, all indexes are included in the list, and the index names are used instead of the field names. The default value is False.

Data Type: Boolean

Modified properties

None.

Required property assignments

DataSource.

Added Events

None.

Added methods

InitCombo

Refreshes wwKeyCombo to reflect the table’s current settings.

Tips

- ◆ If you have a choice use case insensitive indexes in your tables to make incremental searching more user-friendly.
- ◆ To display the drop-down list via keyboard, press the **Alt+down cursor arrow** keys when the component has focus.

TwwLocateDialog



The TwwLocateDialog component itself is non-visual but when executed it provides your end-users with a dialog box that allows them to search for a value within any field, including Memo fields. You specify the default values for case sensitivity (True or False), match type (exact, match any or match starting characters), search field name and the order in which the fields are sorted in the drop-down selection list (by field name or field number). Each of these options are also end-user selectable, except for the field list sort order.

The built-in *FindFirst* and *FindNext* methods can be executed under program control without displaying the dialog box. This lets the developer assign any end-user keyboard key, button or icon to these methods, allowing the user to repeat the last locate command or start over from the beginning of the table without having to re-display the dialog box or re-enter their search criteria.

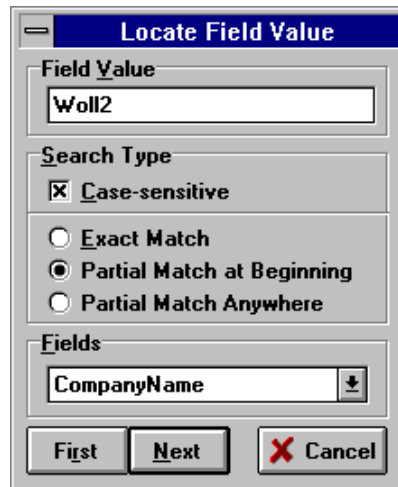


Figure 5.20 - The TwwLocateDialog in action.

Ancestor

TComponent
└─TwwCustomDialog

Required supporting components

TDataSource

Added Properties

Caption

This property contains a text value that is displayed in the dialog box title bar. The default value is "Locate Field Value".

Data Type: String

CaseSensitive

This property defines whether or not the search is to be case sensitive. If the user changes this value, the new value is used the next time the dialog box is displayed. The default value is False.

Data Type: Boolean

DataSource

This property contains the name of a TDataSource component that provides the table name to be searched. The default value is blank.

Data Type: TDataSource

DefaultButton

This property controls which button is the default button when the dialog appears. The default value is dbFindNext.

Data Type: Constant

Valid Values: dbFindFirst *or* dbFindNext

FieldSelection

This property determines which fields are used to fill the available field list. When set to fsAllFields, all fields, excluding non-searchable fields such as Graphic, Blob, etc. are used. When set to fsVisibleFields then only fields whose visible property is True are used. The default value is fsAllFields.

Data Type: Constant

Valid Values: fsAllFields *or* fsVisibleFields

FieldValue

This is a run-time only property. This property allows you to set the search value that the *FindFirst* and *FindNext* methods use. Thus, you can completely bypass the InfoPower locate dialog when performing a locate field value. The default value is an empty string.

Example: The following example finds the next occurrence of 'Port' in the 'City' field. The match can be contained in any part of the 'City' field's value.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with wwLocateDialog1 do begin
    SearchField := 'City';
    FieldValue := 'Port';
    MatchType := mtPartialMatchAny;
    FindNext;
  end;
```

end;

MatchType

This property contains a constant value that defines the type of match to be used. This value is used the first time the dialog box is displayed. If the user changes this value, the new value is used the next time the dialog box is displayed. The default value is `mtPartialMatchStart`.

Data Type: Constant

Valid Values: `mtExactMatch`, `mtPartialMatchAny`, `mtPartialMatchStart`

Options

Use this property for further customizations of appearance and behavior when the dialog box is executed.

Data Type: Set of `TwwLocateDlgOption`

Valid Values: `ldoCaseSensitiveBelow`, `ldoCloseOnMatch`

<i>ldoCaseSensitiveBelow</i>	When True the CaseSensitive checkbox will appear below the match type radio buttons. Otherwise it will appear above.
<i>ldoCloseOnMatch</i>	If False, the dialog will remain open when a match is found. Default is True, in which case the dialog will close as soon as a match is found.

SearchField

This property contains the name of the table's field to be searched against. If the user changes this value, the new value is used the next time the dialog box is displayed. The default value is blank—no *SearchField* specified.

Data Type: String

Valid Values: Existing field name in the table being searched.

ShowMessages

When True, informational messages, such as "No more matches found", are displayed to the user if the locate does not find a match. To disable messages, set this property to False. The default value is True.

Data Type: Boolean

SortFields

This property determines the sort order of the field names that appear in the drop-down field name list. The default is `fsSortByFieldName`.

Data Type: Constant

Valid Values: `fsSortByFieldName` and `fsSortByFieldNo`

Tag

This is the standard Delphi *Tag* property that you can use for your own internal processing needs. The default value is 0.

Data Type: Long

UseLocateMethod

This property is for ADO DataSet performance optimization when locating on a field that is not the active index or the index's case-sensitivity does not match the end-users selections in the dialog. If this is the case, then you can set this property to True so that the ADO Datasets's Locate method is called in partial match or exact match searches.

Data Type: Boolean

Modified properties

None.

Required property assignments

DataSource.

Added Events

Some of the following events pass a handle to the form containing all of the components of the dialog. To see what objects are contained within this editing form, open up wwlocate.pas in the InfoPower source sub-directory. If you do not have the source code version of InfoPower, then perform the steps in Chapter 4's topic "Determining the object names of the controls contained in an InfoPower dialog" on the wwlocate.dfm file contained in the InfoPower lib directory.

If you want to customize any of the objects contained by the form you can use the OnInitDialog event. However if all you are trying to do is to change the labels and hints, then use the TwwIntl | LocateDialog property.

OnInitDialog

Allows you to completely customize every aspect of the dialog box or perform some action during the initialization of the dialog box. For example, you can modify the grid's properties, define custom events, etc.

Note: See the *OnInitDialog* event for TwwLookupDialog or the How To section of the TwwMemoDialog for an example on how to use this event.

Added Methods

Execute

Display the Locate Field Value dialog box to the end user. The dialog box remains displayed until either a First or Next match is found, the user clicks the Cancel button or the user closes the dialog box via the control menu. False is returned when the user clicks Cancel or closes the dialog box. If a First or Next match is found, the dialog box closes and True is returned.

Example: If your `TwwLocateDialog` component's Name is `wwLocateDialog1`, you could attach the following code to a button on your form with a Caption of "Locate" to demonstrate when True and False are returned after displaying the Locate dialog box...

```
if wwLocateDialog1.execute then
    MessageDlg('(True) A match was found. Dialog now closed.',
               mtInformation, [mbOk], 0)
else
    MessageDlg('(False) User clicked Cancel '
               + 'or closed the dialog box.',
               mtInformation, [mbOk], 0);
```

FindFirst

Simulates the user clicking the First button of the dialog box. Searches the selected field in the associated table, *from the top of the table*, for the first occurrence of the Field Value previously entered by the user. The dialog box is displayed only if the Field Value is blank. Returns True if the Field Value was located, or False if it was not.

Example: If your `TwwLocateDialog` component's Name is `wwLocateDialog1`, you could attach the following code to a button on your form with a Caption of "First" to demonstrate when True and False are returned after executing the `FindFirst` method...

```
if wwLocateDialog1.FindFirst then
    MessageDlg('(True) Located first match.', mtInformation, [mbOk], 0)
else
    MessageDlg('(False) No match found.', mtInformation, [mbOk], 0);
```

FindNext

Simulates the user clicking the Next button of the dialog box. Searches the selected field in the associated table, *from the current record location*, for the next occurrence of the Field Value previously entered by the user. The dialog box is displayed only if the Field Value is blank. Returns True if a next match was found, or False if another match was not found.

Example: If your `TwwLocateDialog` component's Name is `wwLocateDialog1`, you could attach the following code to a button on your form with a Caption of "Next" to demonstrate when True and False are returned after executing the `FindNext` method...

```
if wwLocateDialog1.FindNext then
    MessageDlg('(True) Located next match.', mtInformation, [mbOk], 0)
else
    MessageDlg('(False) No more matches found.', mtInformation, [mbOk], 0);
```

How To

Refer to the sample application in the `\ip3000\demos\locate` directory to see an example of using the `TwwLocateDialog` component. This sample application also shows how to make the dialog box set the default search field based on the currently active database field.

Tips

- ◆ To set the `SearchField` to the field that currently has input focus, add code to accomplish this just before the `Execute` method in your program.

TwwLookupDialog



InfoPower's non-visual TwwLookupDialog component is similar to TwwDBLookupComboDlg but does not include the visual interface portion of the component. This component, which is *not* bound to any other visual interface component, displays a dialog box (as described in TwwDBLookupComboDlg) to the user whenever you execute it from within your code. This gives you complete control and flexibility over not only *how* the lookup dialog box is displayed and what it contains, but also *when* it's displayed. You can enable up to two optional developer-controlled buttons in this dialog box and define what actions take place when the user clicks on either button.

Ancestor

```
TComponent
├─TtwCustomDialog
│   └─TtwCustomLookupDialog
```

Required supporting components

None.

Added Properties

Caption

This property contains a text value that is displayed in the dialog box's title bar. The default value is "Lookup".

Data Type: String

CharCase

This property defines what case the characters typed in by the user are to take. ecLowerCase converts all characters to lower case. ecNormal allows the user to type both upper and lower case characters. ecUpperCase converts all characters to upper case. The default value is ecNormal.

Data Type: Constant

Valid Values: ecLowerCase, ecNormal, ecUpperCase

GridColor

This property defines the background color of the grid. The default value is clWhite. (When the first column of a grid is fixed, it's colors are the same colors used for the grid's column titles as defined by the *TitleColor* property.)

Data Type: Constant

Valid Values: Valid Delphi color.

GridOptions

This property contains a set of standard Delphi grid options.

Data Type: TSet()

Valid Values: Valid Delphi grid options

GridTitleAlignment

Determines the text alignment of titles in popup-dialog's grid. The default value is taLeftJustify.

Data Type: Constant

Valid Values: taCenter, taLeftJustify *or* taRightJustify

LookupTable

This property defines the TDataSet component to be used for performing the lookup. The default value is blank. This property must be assigned. In order for the dialog to display the search-by combo, the dataset must have its IndexDefs property published. This allows the component to use the indexes.

Data Type: TDataSet

MaxHeight

Defines the maximum Height of the grid in the related dialog. Use this property to control the height of the popup-dialog. The default value for a standard VGA display (640 x 480) is 209.

Data Type: Integer

Valid Values: Positive integer value

MaxWidth

This property defines how wide the dialog box is allowed to grow, in pixels. The default value is 0, which allows the dialog box to grow to the entire width of the screen.

Data Type: Integer

Valid Values: Depends on your screen's display resolution

Options

This property contains a set of Boolean values that control the appearance of the dialog box, as described below. The default values are opShowOK and opShowSearchBy.

Data Type: TSet()

Valid Values: opShowOK, opShowSearchBy, opGroupControls, opFixFirstColumn and opShowStatusBar (described below)

opShowOKCancel When True, the OK and Cancel buttons are displayed in the dialog box. When False these buttons are not displayed—OK can be simulated by double-clicking an entry or by selecting it and then pressing the Enter key. Cancel can be simulated by pressing the Esc key or by closing the dialog box window. The default is True.

<i>opShowSearchBy</i>	When True, the Search By drop-down control is displayed in the dialog box. When False, this control is not visible. The default value is True.
<i>opGroupControls</i>	When True, the Search Characters and Search By controls are displayed side-by-side above the grid. When False, the Search Characters control is displayed above the grid and the Search By control is displayed below the grid. The default value is False.
<i>opFixFirstColumn</i>	When True, the left-most column of the grid is fixed (non-scrollable). When False, the left-most column can be scrolled out of view. The default value is True.
<i>opShowStatusBar</i>	<i>For use with Paradox tables only.</i> When True, a status bar is added to the dialog box that displays the table name, current record number and total number of records in the table.

PictureMaskFromDataSet

This property is only relevant if your datasource is attached to a TwwTable, TwwQuery, TwwQBE, or TwwClientDataset component, as it is always treated as false in other cases.

When customizing the picture masks through the select fields dialog (invoked by clicking on the selected property at design time), the mask information is stored in the related dataset if this property is True. Otherwise the mask information is stored as a property in the related visual component. By storing the mask information in the dataset, you do not need to re-enter the picture mask for other visual controls attached to this same database field.

Data Type: boolean

PictureMaskFromField

Setting this property to True will allow the dialog to automatically use the picture mask defined for the database field when the end-user is entering the text to search for.

Data Type: Boolean

PictureMasks

The assigned picture mask information is stored in this property if PictureMaskFromDataset is false. See the *PictureMaskFromDataSet* property.

Data Type: TStrings

Selected

Clicking the “...” button or double-clicking the SearchDialog component displays the Select Fields dialog box. This dialog box allows you to select the fields you want displayed in the grid, their titles, widths, control types and link information. (See *Using the Select Fields Dialog Box* at the beginning of Chapter 4.) The default value is *all* fields selected, using the *field name* as it’s title, displayed as a *Field* control for a width equal to the number of characters in the field or the title, whichever is longer.

Data Type: (Internal to InfoPower)

Tag

This is the standard Delphi *Tag* property that you can use for your own internal processing needs. The default value is 0.

UserButton1Caption

When you want to display this button on the dialog box, enter the caption text for the button here and then add code to the *OnUserButton1Click* event. The default value is blank.

Data Type: String

UserButton2Caption

When you want to display this button on the dialog box, enter the caption text for the button here and then add code to the *OnUserButton2Click* event. The default value is blank.

Data Type: String

UseTFields

When the UseTFields property is set to true the *Selected* properties Display settings information will be stored and retrieved from the *LookupTable*. When it is set to False the *Selected* properties Display settings information is stored with the *TwwLookupDialog*. The default is True.

Data Type: Boolean

Required property assignments

LookupTable, LookupField

Added Events

Some of the following events pass a handle to the form containing all of the components of the dialog. To see what objects are contained within this editing form, open up *wwidlg.pas* in the InfoPower source sub-directory. If you do not have the source code version of InfoPower, then perform the steps in Chapter 4's topic "Determining the object names of the controls contained in an InfoPower dialog" on the *wwidlg.dfm* file contained in the InfoPower lib directory.

If you want to customize any of the objects contained by the form you can use the *OnInitDialog* event. However if all you are trying to do is to change the labels and hints, then use the *TwwIntl | SearchDialog* property.

OnInitDialog

Allows you to completely customize every aspect of the dialog box or perform some action during the initialization of the dialog box. When using this event, your code must reference **wwidlg** in your source file's Uses clause. This gives you access to all the components in the dialog. For example, you can modify the grid's properties, define custom events, etc.

Example: The following code tells the first user-defined button to show a hint when the user moves the mouse pointer over the button:

```
procedure TForm1.wvLookupDialog1InitDialog(  
    Dialog: TwvLookupDlg);  
begin  
    with Dialog do begin  
        UserButton1.hint := 'Hint for user button 1';  
        UserButton1.showHint := True;  
    end  
end;
```

OnCloseDialog

This event allows you to perform any custom action before the dialog is actually closed

OnPerformCustomSearch

When using a large lookuptable from a remote server, the performance of the dialog's incremental searching can significantly degrade. To resolve this issue, InfoPower adds a new event where you can control the specific action that takes place after the user types a character. In particular the custom action can update the query to only return the records that you are interested in. When using this event, your code is responsible for manipulating the lookuptable based on the parameter values passed in. See the TwvDBLookupCombo OnPerformCustomSearch event for a description of the events parameters.

OnSortChange

If you want to perform some custom action when the end-user makes a selection from the SortBy combo, then place your custom code here.

OnUserButton1Click

When you want to display developer-defined button #1 on the dialog box, enter the caption text for the button in the *UserButton1Caption* property and then add code to this event that will be executed when the end-user clicks the button.

Tip: If you wish for this dialog to immediately close after executing your code, assign the *ModalResult* property of the dialog. The Sender parameter is cast to a TForm to get a handle to the actual dialog on the screen.

```
procedure TForm1.wvLookupDialog1UserButton1Click(  
    Sender: TObject; LookupTable: TDataSet);  
begin  
    (Sender as TForm).ModalResult:= mrOK;  
end;
```

OnUserButton2Click

When you want to display developer-defined button #2 on the dialog box, enter the caption text for the button in the *UserButton2Caption* property and then add code to this event that will be executed when the end-user clicks the button.

Added Methods

Execute

Display the lookup dialog box to the end-user. This method returns a value of False if the user cancelled the dialog, and True if the user did not cancel.

How To

Perform a lookup and fill:

The following example performs a lookup and fill when the end-user clicks on a button. You can easily attach this code to any event you desire (i.e. pull-down menus, keypress, etc.). In this example, the Zip field of table CustomerTable is looked up in a separate zip code table. After the end-user selects the zip code, the City and State fields are filled into the corresponding CustomerTable fields:

```
procedure TGridDemo.Button1Click(Sender: TObject);
begin
  with wwLookupDialog1 do begin
    { Reset back to primary index in case user changed }
    { the index the last time the dialog was called. }
    (LookupTable as TwwTable).IndexName := '';

    { Pre-select record in lookup table }
    (LookupTable as TwwTable).wwFindKey([CustomerTableZip.AsString]);

    if Execute then begin
      { Fill the corresponding fields in CustomerTable }
      CustomerTable.edit;
      CustomerTableZip.AsString :=
        LookupTable.fieldByName('Zip').AsString;
      CustomerTableCity.AsString :=
        LookupTable.fieldByName('City').AsString;
      CustomerTableState.AsString :=
        LookupTable.fieldByName('State').AsString;
    end
  end;
end;
```

Using Multiselect with the TwwLookupDialog Component:

The following example demonstrates how one can multiselect from a TwwLookupDialog component and then iterate through the selections and perform whatever action is necessary on the lookup table. In this example the TwwLookupDialog will fill a listbox with the *Customer No* of all the selected records. The table used in this example is IP4CUST.DB and is located in the InfoDemo5 *DatabaseName* alias.

1. Add a TwwTable component to your form and set the following properties:
 - Active = True (after defining the DatabaseName and TableName)
 - DatabaseName = InfoDemo5
 - Name = CustomerTable
 - TableName = IP4CUST.DB

2. Add a TListBox component to your form and set the following properties:
Name = ListBox1
3. Add a TwwLookupDialog component form and set the following properties:
LookupTable = CustomerTable
GridOptions | dgMultiSelect = True
4. Drop a TButton on the form and put the following code in the TButton's OnClick event.

```

Procedure TForm1.Button1Click(Sender: TObject)
begin
    wwLookupDialog1.Execute;
end;

```

5. Attach the following code to the OnCloseDialog event. This code fills the listbox named ListBox1 with the *Customer No* of each selected record.

```

procedure TForm1.wwLookupDialog1CloseDialog(Dialog: TwwLookupDlg);
var i:integer;
begin
    with dialog.wfdbgrid1,dialog.wfdbgrid1.datasource.dataset do
    begin
        for i:= 0 to selectedlist.count-1 do begin
            GotoBookmark(selectedlist[i]);
            Listbox1.items.add(FieldByName('Customer No').asString);
        end
    end
end;

```

6. Attach the following code to the OnInitDialog event. This code pre-selects records in the LookupDialog based on the entries in ListBox1.

```

procedure TForm1.wwLookupDialog1InitDialog(Dialog: TwwLookupDlg);
var i: Integer;
begin
    with Dialog.wfdbgrid1, CustomerTable do
    begin
        for i:= 0 to ListBox1.items.count-1 do begin
            if FindKey([ListBox1.items[i]]) then
                Selectedlist.add(Getbookmark);
        end;
        First;
    end
end;

```

Tips

- ◆ TwwLookupDialog is great general purpose component because it can be displayed at any time, from any source code or attached to any event of any component.
- ◆ You can use this component to access lookup tables even when they use multi-field lookup indexes. When pre-selecting the record, just remember to specify all the fields. The following specifies two lookup values.

```

LookupTable.SetKey;
LookupTable.FieldByName('Field1').AsString := LookupValue1;
LookupTable.FieldByName('Field2').AsString := LookupValue2;
LookupTable.GotoKey;

```


TwwMemoDialog



The TwwMemoDialog component itself is non-visual but when executed it provides your end-users with a pop-up, resizable window where they can view or edit the data stored in memo fields. This is the same dialog box that appears when you double-click a memo field from within a running TwwDBGrid component. Options include giving the user the ability to resize the dialog window, whether or not to automatically word-wrap the memo text to fit within the window's boundaries, whether or not to display the memo field in the TwwDBGrid component, if one is present, whether or not to allow the user to edit the memo, and providing the user with 0, 1 or 2 custom buttons that you define the action for.

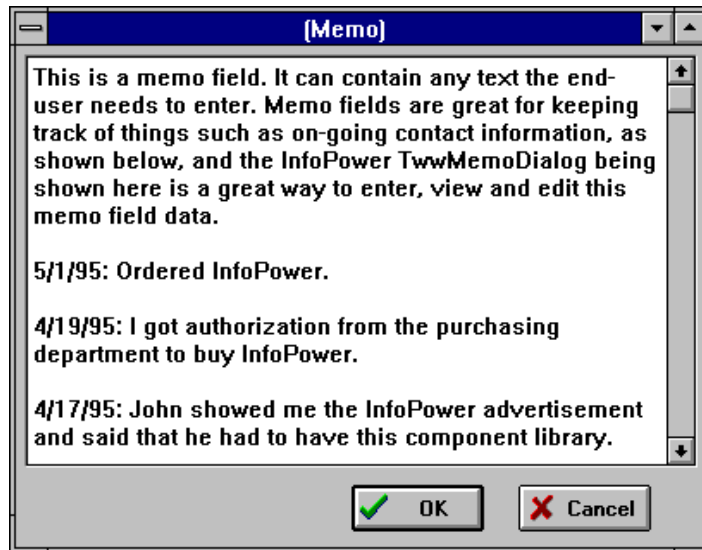


Figure 5.21 - InfoPower's TwwMemoDialog component allows your end-users to enter, view and edit data stored in a memo field.

Ancestor TComponent

Added properties

Caption

This property contains a text value that is displayed in the editor window's title bar. The default value is blank.

Data Type: String

DataField

This property defines the name of the field you want displayed in the memo editor window. The default value is blank.

Data Type: String

Valid Values: Valid field name

DataSource

This property contains the name of a TDataSource component that provides the memo editor with data. The default value is blank.

Data Type: TDataSource

Valid Values: Valid DataSource component name

DlgHeight

This property defines the height of the dialog box in pixels. The default value for a standard VGA screen driver is 396.

Data Type: Integer

Valid Values: Depends on your screen resolution.

DlgLeft

This property defines the left-most position of the dialog box in pixels. If this value is 0, then the dialog is automatically centered horizontally. The default value for a standard VGA screen driver is 0.

Data Type: Integer

Valid Values: Depends on your screen resolution.

DlgTop

This property defines the top-most position of the dialog box in pixels. If this value is 0, then the dialog is automatically centered vertically. The default value for a standard VGA screen driver is 0.

Data Type: Integer

Valid Values: Depends on your screen resolution.

DlgWidth

This property defines the width of the dialog box in pixels. The default value for a standard VGA screen driver is 561.

Data Type: Integer

Valid Values: Depends on your screen resolution.

Font

This is the standard Delphi *Font* property that allows you to define the font and its attributes used to display the memo data in the editor window.

Data Type: TFont

Lines

Use this property if you do not wish to bind your `TwwMemoDialog` to a database field, but instead want to preset its contents. Upon return of the `execute` method, the `lines` property is updated. This property is ignored if you have assigned the component's `datasource` and `datafield` properties.

Data Type: `TStrings`

MemoAttributes

This property contains a set of Boolean values that control the display of Memo data, as described below. The default values are `mSizable` and `mWordWrap`.

Data Type: `TSet()`

Valid Values: `mSizable`, `mWordWrap`, `mGridShow`, `mViewOnly` (described below)

<i>mSizable</i>	When True, the end-user is allowed to resize the pop-up memo editor window. When False, the pop-up editor is displayed as a dialog box. The default value is True.
<i>mWordWrap</i>	When True, word wrapping is automated by adding wrapped words onto additional lines in a vertical manner as necessary. When False, the entire display scrolls horizontally, to the left and right, as words are added. The default value is True.
<i>mGridShow</i>	When True, the memo data is displayed in the grid. When False, memo data is not displayed in the grid. The default value is False. (Warning: Enabling this option dramatically slows down the grid display since memo data must be retrieved from a file other than the table being accessed.)
<i>mViewOnly</i>	When True, the user may not edit the contents of the memo and only the OK button is displayed. When False, the user may edit the memo data and both OK and Cancel buttons are displayed. The default value is False. (Note: If the grid or field <i>ReadOnly</i> property is set to True, this property is automatically set to True.)
<i>mDisableDialog</i>	This property does not have any effect in the stand-alone <code>TwwMemoDialog</code> component. This property is only used in the <i>MemoAttributes</i> property of the <code>TwwDBGrid</code> .

Tag

This is the standard Delphi *Tag* property that you can use for your own internal processing needs. The default value is 0.

UserButton1Caption

This property serves two purposes. First, when the value is blank, no extra button is created on the memo dialog box. Second, when the value is non-blank, a button is created on the memo dialog box with the caption specified in this property. The default value is blank. To define an

action that should take place when the user clicks this button, refer to the *OnUserButton1Click* event described below.

Data Type: String

UserButton2Caption

This property serves two purposes. First, when the value is blank, no extra button is created on the memo dialog box. Second, when the value is non-blank, a button is created on the memo dialog box with the caption specified in this property. The default value is blank. To define an action that should take place when the user clicks this button, refer to the *OnUserButton2Click* event described below.

Modified properties

None.

Added Events

Some of the following events pass a handle to the form containing all of the components of the dialog. To see what objects are contained within this editing form, open up `wwmemo.pas` in the InfoPower source sub-directory. If you do not have the source code version of InfoPower, then perform the steps in Chapter 4's topic "Determining the object names of the controls contained in an InfoPower dialog" on the `wwmemo.dfm` file contained in the InfoPower lib directory.

If you want to customize any of the objects contained by the form you can use the `OnInitDialog` event.

OnCloseDialog

Allows you to define an action when the user closes the MemoDialog. When using this event, your code must reference the file `wwmemo` in your source file's uses clause.

OnInitDialog

Allows you to customize the memo dialog box or perform some action during the initialization of the dialog box. This event is fired after the dialog box is created, but before it's displayed on the screen. When using this event, your code must reference the file `wwmemo` in your source file's Uses clause. For an example of how to use this event, see the *How-to* section below.

OnUserButton1Click

Allows you to define an action when the user clicks on UserButton1. (*See also: UserButton1Caption property above and How-to section below.*)

When using this event, your code must reference the file `wwmemo` in your source file's uses clause.

OnUserButton2Click

Allows you to define an action when the user clicks on UserButton2. (*See also: UserButton2Caption property above and How-to section below.*)

Added Methods

Execute

Display the memo dialog box to the end-user. False is returned if the user clicks the Cancel button or the user closes the dialog box via the control menu. True is returned otherwise. The following example will invoke the memo dialog

```
wwMemoDialog1.Execute;
```

How To

Pre-define the size and position of a memo dialog box:

Set the *DlgHeight*, *DlgLeft*, *DlgTop* and *DlgWidth* properties as described in the *Added properties* section above.

Change the background color of a memo dialog box to red, using the OnInitDialog event:

First, add *wwmemo* to the Uses clause in your source code file. Second, add the following line of code to the *OnInitDialog* event:

```
Dialog.Memo.Color := clRed;
```

Add a programmer-defined button to a memo dialog box:

First, enter the caption text you want to appear on the button in the *UserButton1Caption* or *UserButton2Caption* property value. Second, add the code you want executed when the user clicks the button, to either the *OnUserButton1Click* or *OnUserButton2Click* event.

Close the MemoDialog with a user button:

First, enter the caption text you want to appear on the button in the *UserButton1Caption*. Second, add the following lines of code to use the *UserButton1* to execute your code and then close the form using the OK button.

```
procedure TForm1.wwMemoDialog1UserButton1Click(  
  Dialog: TwwMemoDlg; Memo: TMemo);  
begin  
  Dialog.OkBtnClick(Dialog);  
  Dialog.ModalResult := mrOk;  
end;
```

When using this event, your code must reference the file *wwmemo* in your source file's uses clause.

Add a TimeStamp When Editing Memos:

First, add a *TwwMemoDialog* to your form named *wwMemoDialog1* and assign the *DataSource* and *DataField* properties. Second, add the following lines of code to the *OnInitDialog* event of the *wwMemoDialog*. If you want the *TwwDBGrid*'s *Memo* to have a timestamp, then you just need to assign this routine in the *TwwDBGrid*'s *OnMemoOpen*

event. See *Create an OnInitDialog event for a memo dialog box that is embedded in a TwwDBGrid.*

```
procedure TForm1.wwMemoDialog1OnInitDialog(Dialog: TwwMemoDlg);
begin
    Dialog.Memo.Lines.Add(DateTimeToStr(Now));
end;
```

TwwQBE



The non-visual TwwQBE component allows you to specify Paradox-style Query-By-Example query statements that are used to supply data to one or more of the other InfoPower visual interface components placed on your form. Multi-table queries and special queries such as Insert, Delete and ChangeTo queries are also supported. You can even create a linked, editable live view of one of the tables used to produce the resulting answer table. Since InfoPower's TwwQBE component is inherited from Delphi's TDataSet component, the Delphi Fields editor is still available within this component.

When defining QBE statements, it's a very good idea to create, fully test and then save your query via the Database Desktop or the native application (Paradox or dBASE for Windows). Then, click the Load button of the QBE string editor to load this saved query file into the editor. These environments make it much easier to create, test and validate the results of a QBE than the services provided in Delphi.

Before you add a TwwQBE component to your form, please read about and fully understand how the *AnswerTable* and *AuxiliaryTables* properties function. Also, be aware of the fact that the Borland Database Engine, which is used to process the QBE, by default uses the drive and directory where your program file (.exe) is located as its *working* directory. If this drive does not have sufficient free space for all of the work files created by the QBE processor, including the resulting answer table, you will receive errors and the query will not complete.

If you know more about SQL than Paradox-style QBE, or if your tables are from an SQL database, you might want to consider using the TwwQuery component instead.

Ancestor

TDBDataSet.

Required supporting components

No other Delphi components are required to use a TwwQBE component, but you need to have an existing database table that can be accessed with this component. To add a data-aware component that is connected to the resulting answer table, you need to first add a TDataSource component that uses the TwwQBE component as its DataSet.

Added Properties

AnswerTable

This property allows you to override the default database alias and table name used for a resulting table. The value must be specified in the following format:

```
:<DatabaseName>:<TableName>.db
```

Where <DatabaseName> is an existing database alias and <TableName> is the name of the physical result table to be created. Notice the **required** use of the colon “:” character before and after the <DatabaseName> value, and also the “.db” value at the end of the <TableName> value. This component currently supports only the Paradox style result table (.db). When a physical result table is created, it remains on your disk until it is deleted. If the specified *AnswerTable* already exists, it will be overwritten without warning.

When this property is left blank, the default value, the Borland Database Engine (BDE) creates the query result table as a hidden work file in the directory pointed to by Delphi’s Session.PrivateDir setting, which is the directory where your program is being executed from. If you don’t need to create a physical result table (the default hidden work table is fine), but the default drive and directory don’t contain enough free space for all the necessary work files, you can manually define the working directory, to any drive and directory that contains sufficient free space, by adding the following line of code to this component’s **BeforeOpen** event: (the drive and directory must be specified within double quote marks):

```
Session.PrivateDir := 'd:\private';
```

Data Type: String

Valid Values: Valid DatabaseName (alias) and file name

AuxiliaryTables

This property tells the QBE processor whether or not to create the standard auxiliary tables (keyviol, changed, inserted, etc.). When an auxiliary table is created, it remains on your disk until it is deleted. If an auxiliary table already exists, it will be overwritten without warning. The default value is True.

Caution: If you set this property to False, you may not be able to fully check the results of your QBE since no auxiliary tables will be created for manual verification.

Data Type: Boolean

BlankAsZero

When True blanks in numeric fields in a table are treated equivalent to a zero. The default value is False.

Data Type: Boolean

ControlType

This property holds information about the type of control used to display a field if the field is contained within a grid component. The default value is Field.

Data Type: (Internal to InfoPower)

Valid Values: (Internal to InfoPower)

LookupFields

Maintained for backwards compatibility with earlier versions of InfoPower.

Data Type: (Internal to InfoPower)

Valid Values: (Internal to InfoPower)

LookupLinks

Maintained for backwards compatibility with earlier versions of InfoPower.

Data Type: (Internal to InfoPower)

Valid Values: (Internal to InfoPower)

OnFilterOptions

See the documentation for *OnFilter* under the TwwTable component

PictureMasks

This property holds information about a field's picture mask. See *Using InfoPower's Picture Masks* in Chapter 4 for more details.

Data Type: TStrings

Valid Values: (Internal to InfoPower)

QBE

This property holds the actual QBE query statements. After creating a query via the Database Desktop, or from within dBASE or Paradox, save the query as a .QBE file (*File | Save* menu options). You can then load this previously saved file into the Delphi String list editor window for the *QBE* property by clicking on the Load button and selecting the file you previously saved. Optionally, you can specify the QBE statements interactively. The default value is blank - no QBE defined.

Notes:

- 1) The contents of the "ANSWER:" QBE statement generated when you save a query to a file are ignored by the BDE QBE processor. Use the *AnswerTable* property to specify a database name (alias) and table name if you want to create a physical query result table on your disk.
- 2) The contents of the Selected Fields list box of the Select Fields dialog box default to all fields that are selected via the QBE statements contained in the *QBE* property.

Data Type: TStrings

Valid Values: Valid Query By Example code (see example in "How to" below).

Modified properties

None.

Required property assignments

DatabaseName and *QBE* (valid Paradox-style QBE code).

Added Events

OnFilter

See the documentation for *OnFilter* under the *TwwTable* component

OnFilterEscape

See the documentation for *OnFilterEscape* under the *TwwTable* component.

OnInvalidValue

See *Using InfoPower's Picture Masks* in chapter 4.

Added Methods

ClearParams

This method clears the current parameter definitions for the related *TwwQBE* component. Refer to the *SetParam* method for more details on using parameters in a *TwwQBE* component. Its calling syntax is as follows:

```
Procedure ClearParams;
```

SetParam

This method allows you to substitute tilde (~) variables in your QBE. *ParamName* specifies the name of the tilde variable (exclude the ~), and *ParamValue* specifies the value it should be. Its calling syntax is as follows:

```
Procedure SetParam(paramName: string; paramValue: string);
```

Example: Given the following QBE stored in a *TwwQBE* component, the following code replaces the “~LastName” variable with the string ‘Woll’, and then re-executes the QBE:

```
Query
EZCUST.DB | CustomerNo | CompanyName | FirstName | LastName |
          | Check      | Check      | Check      | ~LastName |
EndQuery

procedure TForm1.Button1Click(Sender: TObject);
begin
  wwQBE1.Active := False;
  wwQBE1.ClearParams;
```

```

wwQBEL.SetParam('LastName','Woll');
wwQBEL.Active := True;
end;

```

wwFilterField

See the documentation for wwFilterField under the TwwTable component

How To

Define QBE statements:

If you want to query the Customer table of your CustData database, selecting only records where the State is equal to CO and the Status is equal to P, retrieving the CustNum, CompanyName, City, State, Zip, Status and FullName fields, create the query by using the Database Desktop and then save it to a .QBE file via the *File | Save* menu options. The resulting query definition would look like the following:

```

Query
ANSWER: ~:PRIV:ANSWER.DB

:CustData:CUSTOMER.DB | CustNum | CompanyName | City | State |
| Check | Check | Check | Check CO |

:CustData:CUSTOMER.DB | Zip | Status | FullName |
| Check | Check P | Check |

EndQuery

```

Then, open the QBE String list editor by clicking on the “...” button of the *QBE* property, click the Load button, select the .QBE file you just saved, click the OK button of the String list editor.

Next, set the *DatabaseName* property of the TwwQBE to the alias referenced in your QBE. The following example references an alias named CustData.

```

DatabaseName := CustData

```

Heterogeneous QBEs (Using more than one alias)

On occasion you may have a QBE which references more than one alias. InfoPower requires that each alias of a query be opened prior to executing the QBE. By specifying the *DatabaseName* property, you can open one alias. To open additional aliases you will need to drop TDatabase components for each alias, assign their *AliasName* property, and set their *connect* property to True.

Using parameters in your QBE statement

You can use parameters to modify your QBE statement at runtime. The TwwQBE component will interpret names preceded by a tilde (~) character as a parameter that you can assign at runtime. Use the method *ClearParams* to reset your tilde parameters and *SetParam* to assign them a value. To re-execute your QBE after assigning the parameters, toggle the QBE’s *active* property from False back to True. See the method *SetParam* for an example.

Tips

- ◆ If you want the answer table written to your hard drive, specify a database name (alias) and table name in the *AnswerTable* property.
- ◆ If you want the normal auxiliary QBE tables written to your hard drive, set the *AuxiliaryTables* property to True.
- ◆ The Delphi Fields editor is also available within this component.
- ◆ If you receive the error message “Error creating cursor handle” when executing your QBE, it means that your query was unable to be executed. Likely causes are incorrect syntax in your QBE specification, unopened aliases, or missing tables.

TwvQuery



The non-visual TwvQuery component allows you to define a query with SQL statements that supplies data to one or more of the other InfoPower visual interface components placed on your form. InfoPower allows you to use TQuery instead of TwvQuery, but this component is provided for backward compatibility. The TwvQuery additionally will allow you to filter on lookupfields.

Since InfoPower's TwvQuery component is inherited from Delphi's TQuery component, all standard Delphi TQuery component properties and functionality are available, such as Delphi's built-in Fields editor and the SQL Query Builder (if using the Client/Server version of Delphi).

Ancestor

TQuery.

Required supporting components

None.

Added Properties

ControlType

This property holds information about the type of control used to display a field if the field is contained within a grid component. The default value is Field. (See *Using the Select Fields Dialog Box* at the beginning of Chapter 4.) To change this property at runtime, see the *SetControlType* method of the wwDBGGrid component.

Data Type: (Internal to InfoPower)

LookupFields

Maintained for backward compatibility with earlier versions of InfoPower.

Data Type: (Internal to InfoPower)

LookupLinks

Maintained for backward compatibility with earlier versions of InfoPower.

Data Type: (Internal to InfoPower)

OnFilterOptions

See the documentation for *OnFilter* under the TwvTable component

PictureMasks

This property holds information about a field's picture mask. See *Using InfoPower's Picture Masks* in Chapter 4 for more details.

Data Type: TStrings

Valid Values: (Internal to InfoPower)

ValidateWithMask

See the documentation for *ValidateWithMask* under the TwwTable component

Modified properties

None.

Required property assignments

1) *DataSource* or *DatabaseName*, and 2) *SQL*—valid query via SQL statements.

Added Events

OnFilter

See the documentation for *OnFilter* under the TwwTable component

OnFilterEscape

This event is fired after the end-user has cancelled a filter in progress by pressing the <Esc> key. You may wish to use this event to display an informational message to the user so that they are aware they have cancelled the filter. See also the *onFilter* event.

OnInvalidValue

See *Using InfoPower's Picture Masks* in Chapter 4.

Added Methods

SetLookupField

See the documentation for *SetLookupField* under the TwwTable component

wwFilterField

See the documentation for *wwFilterField* under the TwwTable component

How To

The TwwQuery component is inherited from Delphi's TQuery, so please refer to your Delphi manual for more information about this component.

Since InfoPower's TwwQuery component is inherited from Delphi's TQuery component, you are provided with 100% backward compatibility. Thus, you can safely replace your use of TQuery with TwwQuery at any time. In addition, all standard Delphi component properties and functionality are still available, such as the Fields editor, and the SQL Query Builder if you use the C/S version of Delphi.

TwwRadioButton

The TwwRadioButton is not a standalone control, but a control that is created for each item in a TwwRadioGroup.

Ancestor

TRadioButton

└TwwCustomRadioButton

Added Properties

Alignment

Assign this property to change the location of the text within the control. If Alignment is set to taRightJustify, then the text is aligned on the right-hand side of the control. If Alignment is set to taLeftJustify, then the text is aligned on the left-hand side of the control.

AlwaysTransparent

Set this to true if you wish for the radio button to be transparent even when it has the focus. Normally when Frame.Enabled and Frame.Transparent are both true, the control is only transparent when it does not have the focus. Note: This property has no effect unless Frame.Enabled and Frame.Transparent are both true.

Caption

Assign a string to this property to assign the label that appears next to the radio button. The TwwRadioGroup sets the caption of each radio button based on its Items property.

Checked

This value indicates whether or not the radio button is selected or not.

Frame

See the topic “Key properties and events for custom framing” in chapter 4 for more information on this property.

Data Type: TwwEditFrame

Images

Assign this property if you wish to change the icons displayed by the radio button. The first image in the imagelist is used as the unselected icon and the second image is used as the selected icon.

Data Type: TImageList

Indents

Use Indents to change the relative placement of the icons and the text.

- ButtonX** Assign this property to specify the number of pixels to move the radio button icon to the left (positive value) or right (negative value).
- ButtonY** Assign this property to specify the number of pixels to move the radio button icon upward (negative value) or downward (negative value).
- TextX** Assign this property to specify the number of pixels to move the text to the left (positive value) or right (negative value).
- TextY** Assign this property to specify the number of pixels to move the text upward (negative value) or downward (negative value).

ShowFocusRect

When true, a focus rectangle is drawn around the text. You may wish to set this property to false when using custom framing, as this can already give the end-user a visual cue to when the checkbox has the focus.

ValueChecked

This is the value that is stored into the database when the radio button is selected.

ValueUnchecked

This is the value that is stored into the database when the checkbox is not selected.

TwvRadioGroup

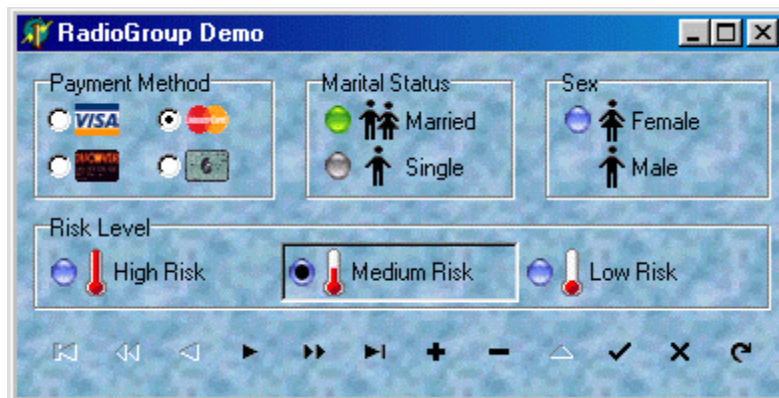


InfoPower integrates a versatile new radio group control into its suite. A radio group contains a set of radio buttons that are grouped in such a way that only one item can be selected. This provides you with an easy way to ensure that the enduser enters one and only one of the options presented. This control is especially useful since it can be bound to a database field that contains a small amount of possible values.

To add radio buttons to a TwvRadioGroup, edit the items property in the Object Inspector. Each string in Items makes a radio button appear in the group box with the string as its caption. Use the values to map the displayed value to a different stored value when the control is attached to a database field.

Some additional powerful features include the following:

- ◆ Support for your own custom bitmaps for the radio button glyphs selected and unselected states.
- ◆ Integration with the InfoPower grid, record-view components, and data inspector.
- ◆ Show glyphs in addition to the text or glyphs only as in the Payment Method radio group example below.
- ◆ Custom framing and transparency support for a consistent look with other InfoPower edit controls.



Ancestor

TCustomGroupBox

└TwvCustomTransparentGroupBox

└TwvCustomRadioGroup

Added Properties

ButtonFrame

This property allows you to set the framing style properties of the generated radio buttons in the `TwwRadioGroup`. See the topic “Key properties and events for custom framing” in chapter 4 for more information on this property.

Data Type: `TwwEditFrame`

Frame

To enable custom framing with this control, you must set `ShowBorder` to `False`. See the topic “Key properties and events for custom framing” in chapter 4 for more information on this property.

Data Type: `TwwEditFrame`

GlyphImages

Assign this property if you wish to display a glyph in place of or next to the radio buttons caption text. The glyphs in the `GlyphImages` properties needs to be in the same order as the `items` property. Set `ShowText` to `False` to show only glyphs and no text in the radio group.

Data Type: `TImageList`

Images

Assign this property if you wish to change the selection icons displayed by the radio group. The first image in the `imagelist` is used as the unselected icon, and the second image is used as the selected icon.

Data Type: `TImageList`

Indents

Use `Indents` to change the relative placement of the icons and the text.

- ButtonX** Assign this property to specify the number of pixels to move the radio button icon to the left (positive value) or right (negative value).
- ButtonY** Assign this property to specify the number of pixels to move the radio button icon upward (negative value) or downward (negative value).
- TextX** Assign this property to specify the number of pixels to move the text to the left (positive value) or right (negative value).
- TextY** Assign this property to specify the number of pixels to move the text upward (negative value) or downward (negative value).

ItemIndex

This runtime only property allows you to get programmatically set the selected item for an unbound `TwwRadioGroup` control. When the control is bound to a database field, there is no need to initialize it since it will display the value associated with that field automatically.

Data Type: `Integer`

Items

Items holds a TStrings object that lists the captions of the radio buttons in the group. These captions become the values of the radio buttons, unless overridden by the Values property.

Data Type: TStrings

ShowBorder

When true, a standard radiogroup engraved border will appear around the outside edge of the *TwvRadioGroup* control. To enable custom framing as in other InfoPower controls you can set this property to False and use the *Frame* property.

Data Type: Boolean

ShowFocusRect

When true, a focus rectangle is drawn around the text. You may wish to set this property to false when using custom framing, as this can already give the end-user a more elegant visual cue to when the radio button has the focus.

Data Type: Boolean

ShowGroupCaption

Set this property to false to hide the main caption of the *TwvRadioGroup*. This may be useful to you if your radio group is embedded in the grid or datainspector control.

Data Type: Boolean

ShowText

Set this property to false to hide the text of the radio buttons in the *TwvRadioGroup*

Data Type: Boolean

Transparent

Set this property to make the *TwvRadioGroup* paint transparently.

Data Type: Boolean

TransparentActiveItem

Set this property to make the active item in the radio group also paint transparently.

Data Type: Boolean

Value

Holds the value of the *TwvRadioGroup* control based on the selected radio button.

Data Type: String

Values

By default the value of the radio buttons in the group are determined by the items property. However, often you may want the values to differ from the captions. For example, if you use radio buttons to represent a database field whose content can be 'Y' or 'N', you may want the radio button's caption to have more descriptive text like 'Yes' or 'No'. So you would enter 'Y' and 'N' in the *Values* list, and 'Yes' and 'No' in the *Items* list.

Data Type: TStrings

Added Events

OnCreateRadioButton

Use this event to customize the dynamically generated *TwwRadioButtons* that are created based on the *Items* property. See *TwwRadioButton*.

Parameters

Sender : *TwwCustomRadioGroup* RadioGroup that is creating the buttons.

RadioButton : *TwwRadioButton* DataSet being looked up

How To

Setting Items and Values at Runtime

This example uses a database radio group box connected to field in a dataset. The actual field can contain one of the values 'Y', 'N', or 'M'. However, you want the user to see the following captions on the radio buttons: 'Yes', 'No', or 'Maybe'.

When the code runs, three radio buttons appear in the group box. If the current record in the dataset contains any of the values contained in the *Values* property, the appropriate radio button is automatically checked. When the user selects a radio button with the mouse or the keyboard, then the corresponding string in the *Values* property is stored into the field.

```
with wwRadioGroup1 do
begin
  Items.Clear;
  Items.BeginUpdate;
  Items.Add('Yes');
  Items.Add('No');
  Items.Add('Maybe');
  Items.EndUpdate;
  Values.Clear;
  Values.Add('Y');
  Values.Add('N');
  Values.Add('M');
end
```

Make a Transparent *TwwRadioGroup*.

Set the *Transparent* property to True in order to make the Radio Group Transparent. However, the active item of the *TwwRadioGroup* is a *TwwRadioButton* that will not by default be transparent. If you want a completely transparent control, then set *TransparentActiveItem* to True as well.

Warning: Do not use transparency if there is no background image.

TwwRecordViewDialog



The TwwRecordViewDialog component is a non-visual component that provides a convenient way to view or edit a record's contents. The component dynamically creates a form based on your DataSet's field properties. This component removes the necessity of building custom record editing forms for each table. InfoPower's RecordView supports embedded controls, picture masks, horizontal or vertical display, custom menus, modal or non-modal display, grid integration, and detailed display options.

The screenshot shows a window titled "Record View" with a menu bar (File, Edit, Record) and a toolbar with navigation and editing icons. The form contains the following fields and values:

348	No	Giant Plumbing
Customer No	Buyer	Company Name
John	Schultz	
First Name	Last Name	
Gregersensvej, PO 141		
Street		
32720	Fernandina Beach	
Zip	City	
FL		
State		
This is some Rich Edit Text		
Rich Edit		
01/04/1994	452-773-0444	Yes
First Contact Date	Phone Number	Requested Demo

At the bottom of the dialog are "OK" and "Cancel" buttons.

Figure 5.22 - InfoPower's TwwRecordViewDialog component allows your end-users a convenient way to view or edit a record's contents. The above uses the record-view's horizontally layout style with the labels displayed underneath the controls.

Ancestor

TComponent
└─TwwCustomDialog

Required supporting components

TDataSource.

Added Properties

BorderStyle

This property defines the record-view form's BorderStyle. The default is bsSizeable.

Data Type: TFormBorderStyle

Valid Values: Standard Delphi FormBorderStyle

Caption

This property contains a text value that is displayed in the editor window's title bar. The default value is 'Record View'

Data Type: String

ControlInfoInDataset

Set this property to False if you wish for the record view dialog to store the information about the embedded controls into its ControlType property . You may wish to set this property to *False* if you want the record view dialog to have no dependency upon the embedded control information stored in the dataset. By default this property is *True*, which means that information about the embedded controls is stored in the related *TDataSet*.

Note: Normally you will want to leave this property as *True*. Set this property to False if you have more than one IP container control such as a grid or record-view attached to the same dataset, and do not wish for them to share the same custom control.

ControlOptions

This property contains a set of Boolean values that control the display of embedded controls.

Data Type: Set of TwwRecordViewControlOption

Valid Values: rvcTransparentLabels, rvcTransparentButtons, rvcFlatButtons

<i>rvcTransparentLabels</i>	If True, then the labels in the TwwRecordViewDialog will be transparent. This is useful when you have a background image being used in the dialog. This property defaults to True.
<i>rvcTransparentButtons</i>	If True, then controls with buttons in the dialog will have their buttons be displayed transparently.
<i>rvcFlatButtons</i>	If True, then controls with buttons in the dialog will have their buttons be displayed as flat.

ControlType

This property is equivalent to the *ControlType* property (See *TwWTable ControlType*). InfoPower stores the control information into this property if the *ControlInfoInDataSet* property is *False*. Otherwise this property is not used.

DataSource

This property contains the name of a TDataSource component that provides the record view form with data. The default value is blank.

Data Type: TDataSource

EditFrame

See the topic “Key properties and events for custom framing” in chapter 4 for information on this property. Set the EditFrame property to customize how the contained edit control’s borders and background are painted. For instance, set the EditFrame.Enabled and EditFrame.Transparent properties to True to display the edit controls transparently. See also the *OnSetControlEffects* to individually customize the borders of a control.

Data Type: TwWEditFrame

EditSpacing | HorizontalView

This property defines the spacing values for the controls in the record view form when using Style=rvsHorizontal

BetweenEditsInRow	Horizontal Space between the edit controls Data Type: Integer
BetweenLabelEdit	Vertical Space between the label and the related edit control Data Type: Integer
BetweenRow	Vertical Space between the edit control and the top of the label on the next row Data Type: Integer
LabelIndent	Horizontal indentation of label with respect to its related edit control Data Type: Integer

EditSpacing | VerticalView

This property defines the spacing values for the controls in the record view form when using Style=rvsVertical

BetweenLabelEdit	Horizontal Space between the label and the related edit control Data Type: Integer
BetweenRow	Vertical Space between the edit controls Data Type: Integer

Font

This is the standard Delphi *Font* property that allows you to define the font and its attributes used to display the controls in the record-view form. See also the *LabelFont* property.

Data Type: TFont

Valid Values: Standard Delphi Font options

FormPosition

Placement and size of the record-view form

Height	Height of the record-view form. Defaults to 0 which means to grow as needed. Data Type: Integer
Left	Left position of the record-view form. Default of 0 means to auto-center the form Data Type: Integer
Top	Top position of the record-view form. Default of 0 means to auto-center the form Data Type: Integer
Width	Width of the record-view form. Defaults to 0 which means to grow as needed. Data Type: Integer

LabelFont

This is the standard Delphi *Font* property that allows you to define the font and its attributes used to display the labels in the record-view form. See also the *Font* property.

Data Type: TFont

Valid Values: Standard Delphi Font options

LinesPerMemoControl

This property determines the height of memo controls in the dialog. The default is 2 lines per memo control.

Data Type: Integer

Margin

Spacing between panels and controls.

BottomOffset	Space between the bottom edit control and the bottom of the record-view panel Data Type: Integer
LeftOffset	Space between the left-most edit control and the left side of the record-view panel Data Type: Integer
RightOffset	Space between the right-most edit control and the right side of the record-view panel. Data Type: Integer

TopOffset Space between the top edit control and the top of the record-view panel
Data Type: Integer

Menu

This property allows you to attach your own custom menu to the record-view form. See the How-to documentation later in this section for an example.

Data Type: TMainMenu

Navigator

Set this property to change the default navigator used by the RecordViewDialog. The object inspector will display all TwwDBNavigator controls in your current form. When this property is assigned the *NavigatorVisibleButtons* property is ignored.

NavigatorButtons

This property defines which buttons in the record-view navigator are visible. To make the navigator invisible, set the Option | HideNavigator property to True.

Data Type: Set of TNavigatorBtn

Valid Values: Set of Delphi TNavigatorBtn values

NavigatorFlat

If True, then the navigator buttons are displayed as flat icons. Defaults to False.

Data Type: Boolean

OKCancelOptions

This property contains a set of Boolean values that control the display of the OK and Cancel buttons in the record-view form

Data Type: Set of TwwRecordViewOKCancelOption

Valid Values: rvokShowOKCancel, rvokAutoPostRec, rvokAutoCancelRec

rvokShowOKCancel If True, then the OK and Cancel buttons are displayed in the record-view form. This property defaults to True.

rvokAutoPostRec If True, then when the OK button is clicked the record-view form automatically posts the record. This property defaults to True.

If the OK button is not visible, then auto-posting only occurs under the following conditions:

1. The user closes the record-view form and Property *rvoCloseIsCancel* is False
2. The dialog is closed with the Enter key. Note: The dialog can only be closed with the enter key if the property *rvoEnterToTab* is False.

rvoAutoCancelRec If True, then when the Cancel button is clicked the record-view form automatically cancels the record's changes by calling the TDataSet cancel method. This property defaults to True.

If the Cancel button is not visible, then auto-cancel only occurs under the following conditions:

1. The user closes the record-view form and the Property *rvoCloseIsCancel* is True.
2. The dialog is closed with the Escape key.

Options

This property contains a set of Boolean values that control options in the record-view form.

Data Type: Set of TwwRecordViewOption

Valid Values: (*rvoHideReadOnly*, *rvoHideCalculated*, *rvoHideNavigator*, *rvoUseCustomControls*, *rvoShortenEditBox*, *rvoModalForm*, *rvoStayOnTopForm*, *rvoConsistentEditWidth*, *rvoEnterToTab*, *rvoConfirmCancel*, *rvoCloseIsCancel*, *rvoMaximizeMemoWidth*, *rvoUseDateTimePicker*, *rvoLabelsBeneathControl*)

rvoHideReadOnly If True, then readonly fields are not displayed in the record-view form. Defaults to False.

rvoHideCalculated If True, then calculated fields are not displayed in the record-view form. Defaults to False.

rvoHideNavigator If True, then the navigator control is not displayed Defaults to True.

rvoUseCustomControls If True, then embedded controls are used by the record-view form. If False, then the fields use a regular edit control for editing. Defaults to True.

rvoShortenEditBox If True, then edit controls that exceed the width of the record-view form are resized to fit into the form. Defaults to True.

rvoModalForm If True, then the record-view form is displayed as a modal dialog. If False, the form is displayed as a non-modal form. When used non-modally, the end-user can switch to another form while the record-view form is displayed. Defaults to True.

rvoStayOnTopForm If True, then the record-view form stays on top of all other forms. This is useful when *rvoModalForm* is set to False as it allows the record-view form to not be hidden when moving to other forms. Defaults to False.

rvoConsistentEditWidth If True, then edit controls are all the same size. This property is only used when the *Style=rvsVertical*. Defaults to False.

<i>rvoEnterToTab</i>	If True, then carriage returns are converted to a tab.
<i>rvoConfirmCancel</i>	If True, then a confirmation dialog appears when the user cancels the dialog. Defaults to True.
<i>rvoCloseIsCancel</i>	If True, then when the user closes the dialog the control bar, the record-view form treats it as a cancel operation. Defaults to True.
<i>rvoMaximizeMemoWidth</i>	If True, then the record-view form will maximize the width of the memofields by placing their related edit control on their own row, and be sized to fit the entire width of the form. If False, then the design time settings are used.
<i>RvoUseDateTimePicker</i>	If True, then the record-view form will automatically create and use the TwwDBDateTimePicker control to edit dates or time fields.
<i>rvoLabelsBeneathControl</i>	If True, then the record-view form will place the labels underneath the control. This property is ignored if Style is rvsHorizontal.

PictureMaskFromDataSet

This property is only relevant if your datasource is attached to a TwwTable, TwwQuery, TwwQBE, or TwwClientDataset component, as it is always treated as false in other cases.

When customizing the picture masks through the select fields dialog (invoked by clicking on the selected property at design time), the mask information is stored in the related dataset if this property is True. Otherwise the mask information is stored as a property in the related visual component. By storing the mask information in the dataset, you do not need to re-enter the picture mask for other visual controls attached to this same database field.

Data Type: boolean

PictureMasks

The assigned picture mask information is stored in this property. See the *PictureMaskFromDataSet* property.

Data Type: TStrings

ReadOnlyColor

This property determines the color of the read-only fields in the record-view form. Defaults to *clInactiveCaptionText*.


Data Type: TColor

ReadOnlyColor

This property determines the color of the read-only fields in the record-view form. Defaults to *clInactiveCaptionText*.

Data Type: TColor

Selected

This property determines the field layout of the record-view form. It determines the field order, the display labels, and the edit control width. Clicking in this property brings up a specialized form of the *Select Fields Dialog*, which allows line-breaks to be inserted. You can insert line breaks by clicking on the  icon.

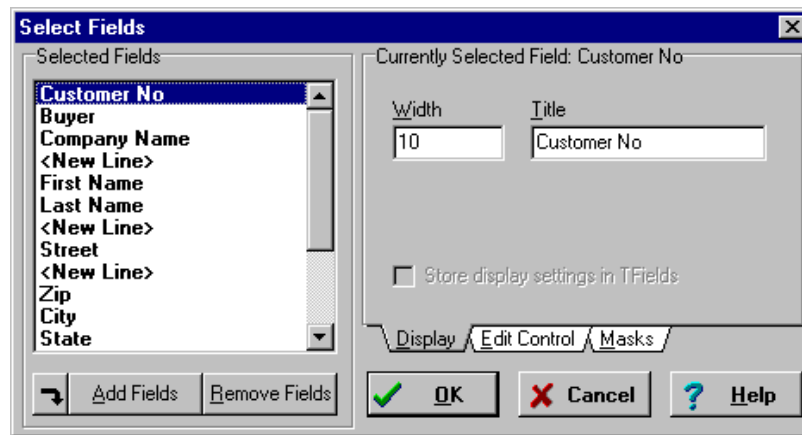


Figure 5.23 - Select Fields Dialog for TwwRecordViewDialog

Using this dialog you can set picture masks, attach edit controls, select fields, etc. See *Using the Select Fields Dialog Box* at the beginning of Chapter 4. The default value is all fields selected.

Style

This property determines the style of the record-view form's field display. If set to *rvsHorizontal*, then each succeeding field is displayed on the same line until it reaches the right edge of the form. You can force line-breaks using the *Selected* property. If set to *rvsVertical*, then only one field is displayed per line.

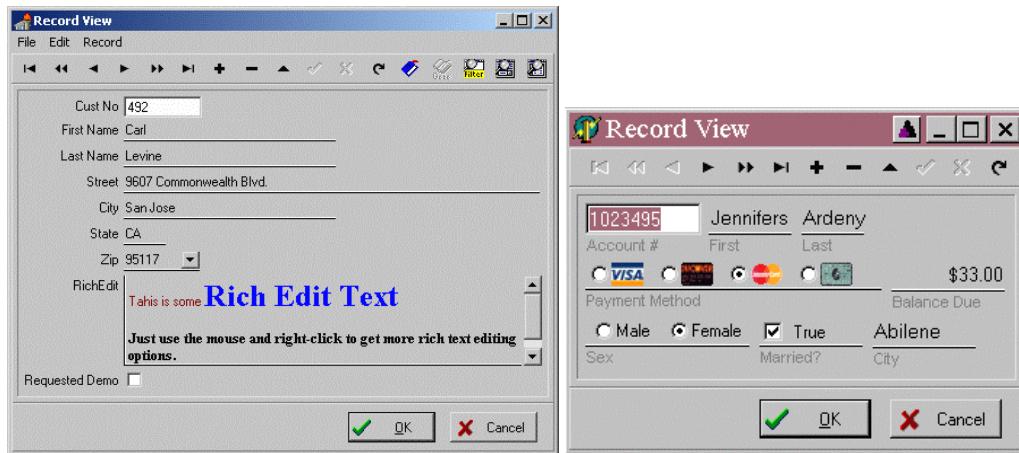


Figure 5.24 - InfoPower's TwwRecordViewDialog component using Vertical and Horizontal layout styles

Data Type: TwwRecordViewStyle

Valid Values: rvsHorizontal, rvsVertical

Required property assignments

DataSource

Added Events

The following events pass a handle to the record-view form. The following objects are contained within the record-view form.

<u>Component Name</u>	<u>Component Type</u>	<u>Description</u>
RecordPanel	TPanel	Panel containing the scrollbox which contains the edit controls.
NavigatorPanel	TPanel	Panel containing the navigator
ButtonPanel	TPanel	Panel containing the OK and Cancel buttons
Navigator	TDBNavigator	Record-view Navigator
ScrollBox	TScrollBox	Scrollbox containing the edit controls

OnAfterCreateControl

This event allows you to customize the edit control after the record-view form has created the control.

Note : Embedded controls on the record-view form can be customized outside the scope of this event since the control already exists before the record-view form is shown. However normal edit controls do not exist until the record-view form is shown, and thus to customize you will need to use this event.

The parameters for this event are as follows.

<i>Form</i> : TwwRecordViewForm	Record-view form
<i>CurField</i> : TField	Field that is related to the edit control
<i>Control</i> : TControl	Control to customize

Example: The following example changes the field 'Last Name' so that it's color is clYellow.

```
procedure TRecordViewDemoForm.wvRecordViewDialog1AfterCreateControl(  
  Form: TwwRecordViewForm; curField: TField; Control: TControl);  
begin  
  if (curField.FieldName='Last Name') and  
    (Control is TCustomEdit) then TEdit(Control).color:= clYellow;  
end;
```

OnBeforeCreateControl

This event allows you to evaluate the control about to be created and either accept or reject its placement into the record-view form.

The parameters for this event are as follows.

<i>Form</i> : TwwRecordViewForm	Record-view form
<i>CurField</i> : TField	Field that is related to the edit control
<i>Accept</i> : boolean	Set to false to reject the component. Defaults to True.

Example: The following example will reject the last name field from being included in the record-view form. Note: You can also remove a field at design time by clicking in the selected property. However if the fields that are to be included are not known until runtime, then you can resort to this technique to remove fields.

```
procedure TRecordViewDemoForm.wvRecordViewDialog1BeforeCreateControl(  
  Form: TwwRecordViewForm; curField: TField; var Accept: Boolean);  
begin  
  if curField.FieldName='Last Name' then Accept:= False;  
end;
```

OnCancelWarning

This event allows you to display your own cancel confirmation dialog. This event is fired only when the user cancels the dialog and the Options | rvoConfirmCancel is True.

The parameters for this event are as follows.

<i>Form</i> : TForm	Record-view form
<i>CanClose</i> : boolean	Set to False to leave the record-view form open.

Example: The following example displays the message *Are you sure you wish to cancel?* The record-view form is closed if the user clicks the Yes button

```
procedure TRecordViewDemoForm.wvRecordViewDialog1CancelWarning(  
    Sender: TForm; var CanClose: Boolean);  
begin  
    CanClose:= MessageDlg('Are you sure you wish to cancel?',  
        mtConfirmation, [mbYes, mbNo], 0)=mrYes;  
end;
```

OnCloseDialog

This event allows you to perform any custom action before the record-view form is closed.

The parameters for this event are as follows.

Form : TwwRecordViewForm Record-view form

OnInitDialog

This event allows you to perform any custom action before the record-view form is initially displayed.

The parameters for this event are as follows.

Form : TwwRecordViewForm Record-view form

OnResizeDialog

This event allows you to perform any custom action when the dialog is being resized.

The parameters for this event are as follows.

Form : TwwRecordViewForm Record-view form

OnSetControlEffects

Use the *OnSetControlEffects* event to override the RecordView's *EditFrame* settings for an individual or selected control.

The parameters for this event are as follows.

Form : TwwRecordViewForm Record-view form

CurField: TField Field that is related to the edit control

Control: TControl Control to customize

Frame: TwwEditFrame Assign this property to change the frame properties for a control.

ButtonEffects: TwwButtonEffects Assign this property to change the button effects properties for a control. Warning: You should first verify that ButtonEffects is not nil before referencing it.

Example: The following code in this event will place a left-border when the edit control is tied to a TBlobField.

```
procedure TRecordViewDemoForm.wvRecordViewDialog1SetControlEffects (
  Form: TwwRecordViewForm; curField: TField; Control: TControl;
  Frame: TwwEditFrame; ButtonEffects: TwwButtonEffects);
begin
  if curfield is TBlobfield then
  begin
    Frame.NonFocusBorders:=
      Frame.NonFocusBorders + [efLeftBorder];
  end
end;
```

Added Methods

Execute

Display the record-view form to the end-user. If used modally, False is returned if the user clicks the Cancel button or the user closes the dialog box via the control menu. If used non-modally, True is always returned. See also the OnCloseDialog event to execute custom code when the dialog is closed.

How To

Customize the selection and order of fields in the record-view form

Click on the *selected* property to invoke the *Select Fields Dialog*. From here you can select which fields should be visible in the record view, as well as determine the order of fields. You can also force line breaks when using style=rvsHorizontal by inserting a <New line>.

Create accelerators for the controls in the record-view form

Click on the *selected* property to invoke the *Select Fields Dialog*. From here you can select the field title. Use the & symbol in the title of the field to create an accelerator in the record-view form for the field. For instance if the field title was &Last Name, then the field label would appear as Last Name, and entering Alt | L would move to that control.

Test the record view layout during design time

After customizing your display options for the recordview, you do not need to execute your program to test the appearance of the record-view form. You can simply dbl-click the component and it will display the record-view exactly as it would appear at runtime.

Embed custom controls in the record-view form

The record-view form is capable of displaying fields as CheckBoxes, SpinEdits, ComboDlgs, ComboBoxes, Memo fields, RichEdit fields, and LookupCombos. The control information is stored at the dataset level, so if you already have attached controls to the dataset using another component such as the TwwDBGrid you do not need to redefine this control information. To

define the control information, click on the *selected* property and select the *Edit Control* tab page. See using the *Select Fields Dialog* for more information on embedding controls.

Warning- You cannot attach the same custom control to multiple fields. For instance if you want to use the same drop-down list for Field1 and Field2, you should use 2 separate TwwDBComboBox's. This is necessary since both combobox's would need to be visible at the same time within the record-view form.

Integrate the TwwRecordViewDialog with the grid.

The record-view dialog is a convenient way of displaying all the details for a particular record. The TwwDBGrid is a convenient way to scan multiple records at once. You can enable end-user access to both views from the grid by enabling the icon in the indicator column, and having it bring up the record-view form when it is clicked. The following steps will accomplish this:

1. Drop a TwwDBGrid (wwDBGrid1) and a TwwRecordViewDialog (wwRecordViewDialog1) into your form, and assign the datasource property for each to the same datasource.
2. Click on the TwwDBGrid's IndicatorButton property to define the indicator button. The object inspector will automatically display the newly created indicator button
 - a) Click on the TwwIButton Glyph property and select a bitmap.
 - b) Click on TwwIButton OnClick event and attach the following code.

```
wwDBGrid1.FlushChanges;  
wwRecordViewDialog1.execute;
```

- c) The call to FlushChanges ensures that the record-view form sees any changes made to the grid. If you do not call this method, then the record-view form will not be able to see the changes made to the currently active field in the grid.
3. *Optional* - Use the RecordViewDialog's OnInitDialog to set the active control to the same field that corresponds to the grid's active field. This allows the active field to remain consistent between the record-view and the grid.

```
procedure TRecordViewDemoForm.wwRecordViewDialog1OnInitDialog(  
    Form: TwwRecordViewForm);  
begin  
    wwdbgrid1.GetActiveField.FocusControl;  
end;
```

Attach your own custom menu to the record-view dialog

You can attach your own custom menu to the record-view dialog with the following steps.

1. Add a new TMainMenu component to your form and set the following properties.

Name=RecordViewMenu
Items=Your menu design

2. Check the Menu property of your form and make sure it is not set to RecordViewMenu. If it is then clear it.
3. Set the Menu property of your TwwRecordViewDialog to RecordViewMenu

See the demonstration program `prcdvw.dpr` for a complete example of using a custom menu in the record-view dialog.

Customize the spacing between controls

The *EditSpacing* property allows you to fine-tune the edit-control spacing of your record-view form to your preferences. See the property *EditSpacing* for details.

Defining the default behavior when the record-view form is closed

The default behavior of the record-view form is to post the record when the OK button is clicked and to call the dataset's cancel method when the Cancel button is clicked. If the user closes the form using the control menu, then the record-view form treats this as a cancel operation. If the user closes the form without saving the changes, the record-view form displays a confirmation dialog. The following are common customizations you may wish to make to change the default closing behavior.

- In cases where you do not have the OK and cancel buttons visible you may want to set the *Options | rvoCloseIsCancel* to False. This treats closing the form via the control menu equivalent to the OK button being clicked.
- If you do not want the form to automatically post the changes to the record when the OK button is clicked, set the *OKCancelOptions | rvokAutoPostRec* to False.
- If you do not want the form to automatically cancel the record's changes when the Cancel button is clicked, set the *OKCancelOptions | rvokAutoCancelRec* to False.
- If you wish to use your own confirmation dialog when closing the form, use the *OnCancelWarning* event.

Defining picture masks for fields in the record-view form

Click on the selected property and define the masks using the Masks tab page. Picture masks are stored at the dataset level, so if you have already defined picture masks with another control (such as the TwwDBGrid), you will not need to redefine them.

TwwRecordViewPanel



This InfoPower component provides a convenient way to view or edit a record's contents. Similar in functionality to the `TwwRecordViewDialog`, this component differs in that the record-view can be embedded on your own form instead of a pop-up dialog. The edit controls are constructed during program execution to allow the panel to adapt itself to any table. Thus with just a few lines of code, you can have a generic form that can edit or view any table.

InfoPower's `RecordViewPanel` supports embedded controls, picture masks, horizontal or vertical display, and detailed display options.

Customer No	348	Buyer	No	Company Name	Giant Plumbing	First Name	John
Last Name	Schultz	Street	Gregersensvej, PO 141				
City	Fernandina Beach	State	FL	Zip	66534	First Contact Date	1/4/94
Phone Number	452-773-0444	Requested Demo	No				

Figure 5.25 - InfoPower's TwwRecordViewPanel component with horizontal view

Required supporting components

`TDataSource`.

Added Properties

ControlInfoInDataset

Set this property to `False` if you wish for the record view panel to store the information about the embedded controls into its `ControlType` property. You may wish to set this property to `False` if you want the record view panel to have no dependency upon the embedded control information stored in the dataset. By default this property is `True`, which means that information about the embedded controls is stored in the related `TDataSet`.

Notes: Normally you will want to leave this property as `True`. Set this property to `False` if you have more than one IP container control such as a grid or record-view attached to the same dataset, and do not wish for them to share the same custom control.

ControlOptions

This property contains a set of Boolean values that control the display of embedded controls.

Data Type: Set of `TwwRecordViewControlOption`

Valid Values: `rvcTransparentLabels`, `rvcTransparentButtons`, `rvcFlatButtons`

<i>rvcTransparentLabels</i>	If True, then the labels in the <i>TwwRecordViewDialog</i> will be transparent. This is useful when you have a background image being used in the dialog. This property defaults to True.
<i>rvcTransparentButtons</i>	If True, then controls with buttons in the dialog will have their buttons be displayed transparently.
<i>rvcFlatButtons</i>	If True, then controls with buttons in the dialog will have their buttons be displayed as flat.

ControlType

This property is equivalent to the *ControlType* property (See *TwwTable ControlType*). *InfoPower* stores the control information into this property if the *ControlInfoInDataSet* property is *False*. Otherwise this property is not used.

DataSource

This property contains the name of a *TDataSource* component that provides the record view form with data. The default value is blank.

Data Type: *TDataSource*

EditFrame

See the topic “Key properties and events for custom framing” in chapter 4 for information on this property. Set the *EditFrame* property to customize how the contained edit control’s borders and background are painted. For instance, set the *EditFrame.Enabled* and *EditFrame.Transparent* properties to True to display the edit controls transparently. See also the *OnSetControlEffects* to individually customize the borders of a control.

Data Type: *TwwEditFrame*

EditSpacing | HorizontalView

This property defines the spacing values for the controls in the record view form when using *Style=rvsHorizontal*

BetweenEditsInRow	Horizontal Space between the edit controls Data Type: Integer
BetweenLabelEdit	Vertical Space between the label and the related edit control Data Type: Integer
BetweenRow	Vertical Space between the edit control and the top of the label on the next row. Data Type: Integer
LabelIndent	Horizontal indentation of label with respect to its related edit control Data Type: Integer

EditSpacing | VerticalView

This property defines the spacing values for the controls in the record view form when using `Style=rvsVertical`

BetweenLabelEdit	Horizontal Space between the label and the related edit control Data Type: Integer
BetweenRow	Vertical Space between the edit controls Data Type: Integer

Font

This is the standard Delphi *Font* property that allows you to define the font and its attributes used to display the controls in the record-view panel. See also the *LabelFont* property.

Data Type: TFont

Valid Values: Standard Delphi Font options

LabelFont

This is the standard Delphi *Font* property that allows you to define the font and its attributes used to display the labels in the record-view form. See also the *Font* property.

Data Type: TFont

Valid Values: Standard Delphi Font options

LinesPerMemoControl

This property determines the height of memo controls in the panel. The default is 2 lines per memo control.

Data Type: integer

Margin

Space between panel and controls.

BottomOffset	Space between the bottom edit control and the bottom of the record-view panel Data Type: Integer
LeftOffset	Space between the left-most edit control and the left side of the record-view panel Data Type: Integer
RightOffset	Space between the right-most edit control and the right side of the record-view panel. Data Type: Integer
TopOffset	Space between the top edit control and the top of the record-view panel Data Type: Integer

Options

This property contains a set of Boolean values that control options in the record-view form.

Data Type: Set of TwwRecordViewOption

Valid Values: (rvopHideReadOnly, rvopHideCalculated, rvopUseCustomControls, rvopShortenEditBox, rvopConsistentEditWidth, rvopMaximizeMemoWidth, rvopUseDateTimePicker, rvopLabelsBeneathControl)

<i>rvopHideReadOnly</i>	If True, then readonly fields are not displayed in the record-view form. Defaults to False.
<i>rvopHideCalculated</i>	If True, then calculated fields are not displayed in the record-view form. Defaults to False.
<i>rvopUseCustomControls</i>	If True, then embedded controls are used by the record-view form. If False, then the fields use a regular edit control for editing. Defaults to True.
<i>rvopShortenEditBox</i>	If True, then edit controls that exceed the width of the record-view form are resized to fit into the form. Defaults to True.
<i>rvopConsistentEditWidth</i>	If True, then edit controls are all the same size. This property is only used when the Style=rvpsVertical. Defaults to False.
<i>rvopMaximizeMemoWidth</i>	If True, then the record-view form will maximize the width of the memofields by placing their related edit control on their own row, and be sized to fit the entire width of the form. If False, then the design time settings are used.
<i>rvopUseDateTimePicker</i>	If True, then the record-view panel will automatically create and use the TwwDBDateTimePicker control to edit dates or time fields.
<i>rvopLabelsBeneathControl</i>	If True, then the record-view will place the labels underneath the control. This property is ignored if Style is rvpsHorizontal.

PictureMaskFromDataSet

This property is only relevant if your datasource is attached to a TwwTable, TwwQuery, TwwQBE, or TwwClientDataset component, as it is always treated as false in other cases.

When customizing the picture masks through the select fields dialog (invoked by clicking on the selected property at design time), the mask information is stored in the related dataset if this property is True. Otherwise the mask information is stored as a property in the related visual component. By storing the mask information in the dataset, you do not need to re-enter the picture mask for other visual controls attached to this same database field.

Data Type: boolean

PictureMasks

The assigned picture mask information is stored in this property if PictureMaskFromDataset is false. See the *PictureMaskFromDataSet* property.


Data Type: TStrings

ReadOnlyColor

This property determines the color of the read-only fields in the record-view form. Defaults to *clInactiveCaptionText*.

Data Type: TColor

Selected

This property determines the field layout of the record-view panel. It determines the field order, the display labels, and the edit control width. Clicking in this property brings up a specialized form of the *Select Fields Dialog*, which allows line-breaks to be inserted. You can insert line breaks by clicking on the  icon.

Using this dialog you can set picture masks, attach edit controls, select fields, etc. *See Using the Select Fields Dialog Box* at the beginning of Chapter 4.) The default value is all fields selected.

Data Type: (Internal to InfoPower)

Valid Values: (Internal to InfoPower)

Style

This property determines the style of the record-view panel's field display. If set to *rvpsHorizontal*, then each succeeding field is displayed on the same line until it reaches the right edge of the panel. You can force line-breaks using the *Selected* property. If set to *rvpsVertical*, then only one field is displayed per line.

Data Type: TwwRecordViewPanelStyle

Valid Values: rvpsHorizontal, rvpsVertical

Required property assignments

DataSource

Added Events

OnAfterCreateControl

This event allows you to customize the edit control after the record-view form has created the control.

Note : Embedded controls on the record-view form can be customized outside the scope of this event since the control already exists before the record-view form is shown. However normal edit controls do not exist until the record-view form is shown, and thus to customize you will need to use this event.

The parameters for this event are as follows.

<i>Sender</i> :	TObject	Record-view Panel
<i>CurField</i> :	TField	Field that is related to the edit control

Control: TControl

Control to customize

Example: The following example changes the field 'Last Name' so that it's color is clYellow.

```
procedure TForm1.wvRecordViewPanel1AfterCreateControl(  
  Sender: TObject; curField: TField; Control: TControl);  
begin  
  if (curField.FieldName='Last Name') and  
    (Control is TCustomEdit) then TEdit(Control).Color:= clYellow;  
end;
```

OnBeforeCreateControl

This event allows you to evaluate the control about to be created and either accept or reject its placement into the record-view form.

The parameters for this event are as follows.

<i>Sender:</i> TObject	Record-View Panel
<i>CurField:</i> TField	Field that is related to the edit control
<i>Accept:</i> boolean	Set to false to reject the component. Defaults to True.

Example: The following example will reject the last name field from being included in the record-view panel. Note: You can also remove a field at design time by clicking in the selected property. However if the fields that are to be included are not known until runtime, then you can resort to this technique to remove fields.

```
procedure TForm1.wvRecordViewPanel1BeforeCreateControl(  
  Sender: TObject; curField: TField; var Accept: Boolean);  
begin  
  if curField.FieldName='Last Name' then Accept:= False;  
end;
```

OnSetControlEffects

Use the *OnSetControlEffects* event to override the RecordView's *EditFrame* settings for an individual or selected control. See the TwwRecordViewDialog OnSetControlEffects event for further information.

How To

Customize the selection and order of fields in the record-view form

Click on the *selected* property to invoke the *Select Fields Dialog*. From here you can select which fields should be visible in the record view, as well as determine the order of fields. You can also force line breaks when using style=rvpsHorizontal by inserting a <New line>.

Create accelerators for the controls in the record-view form

Click on the *selected* property to invoke the *Select Fields Dialog*. From here you can select the field title. Use the & symbol in the title of the field to create an accelerator in the record-view form for the field. For instance if the field title was &Last Name, then the field label would appear as Last Name, and entering Alt | L would move to that control.

Embed custom controls in the record-view panel

The record-view panel is capable of displaying fields as CheckBoxes, SpinEdits, ComboDlgs, ComboBoxes, Memo fields, DateTimePickers, RichEdit fields, and LookupCombos. The control information is stored at the dataset level, so if you already have attached controls to the dataset using another component such as the *TwwDBGrid* you may not need to redefine this control information. However if your grid's embedded controls are not on the same form as the record view panel, the record-view panel will not find the embedded controls. In this case you should set the *ControlInfoInDataset* property to False, and attach new custom controls to the record-view panel. To define the control information, click on the *selected* property and select the *Edit Control* tab page. See using the *Select Fields Dialog* for more information on embedding controls.

Warning: You cannot attach the same custom control to multiple fields. For instance if you want to use the same drop-down list for Field1 and Field2, you should use 2 separate *TwwDBComboBox*'s. This is necessary since both comboboxes would need to be visible at the same time within the record-view form.

Customize the spacing between controls

The *EditSpacing* property allows you to fine-tune the edit-control spacing of your record-view form to your preferences. See the property *EditSpacing* for details.

TwwSearchDialog



TwwSearchDialog is a highly functional incremental search dialog box component that provides your end-users with a means to incrementally search for values against any of the table's index fields you allow the user to select. The dialog box includes a developer-controlled table grid, search criteria edit box and optionally displays a combo box containing the table's indexes you allow the user to select from. Any of the fields available in the associated search table can be selected for display in the table grid, regardless if the field is displayed on your form or not. You can enable up to two optional developer-controlled buttons in this dialog box and define what actions take place when the user clicks on either button.

Call the component's *Execute* method to bring up the search dialog. You can customize which fields are shown in the dialog by dbl-clicking the component at design time.

Assign the *SearchTable* property to define which dataset to search. You can optionally assign the *ShadowSearchTable* property as well if you wish to prevent the searching process from changing the dataset record pointer.

If you are not using a searching on a TTable component and you are using a ShadowSearchTable, you must write code in the OnSyncDataSet event to synchronize the record pointers in your SearchTable and ShadowSearchTable. See the OnSyncDataSet event for more information.

Just like the TwwIncrementalSearch component, as the end-user enters characters into the edit box, the component performs a "locate index value" operation based on the characters currently in the edit box, moving to the record that contains the closest match. (If you are using SQL tables and TwwTables, refer to the TwwTable component's *SyncSQLByRange* property and also the SQL entries in the Troubleshooting section for some very useful information that will assist you in working with SQL tables and InfoPower.)

The current search field is always located in the first column of the grid display. When the first column is fixed, it is non-scrollable, cannot be resized by the user, and is displayed using the same colors as the column heading titles (*TitleColor* property). The remainder of the grid is displayed in the color specified in the *GridColor* property. If the user is allowed to change the *Search By* field, the selected field is automatically moved to the first column of the grid display

Ancestor

```
TComponent
├─ TwwCustomDialog
│   └─ TwwCustomLookupDialog
```

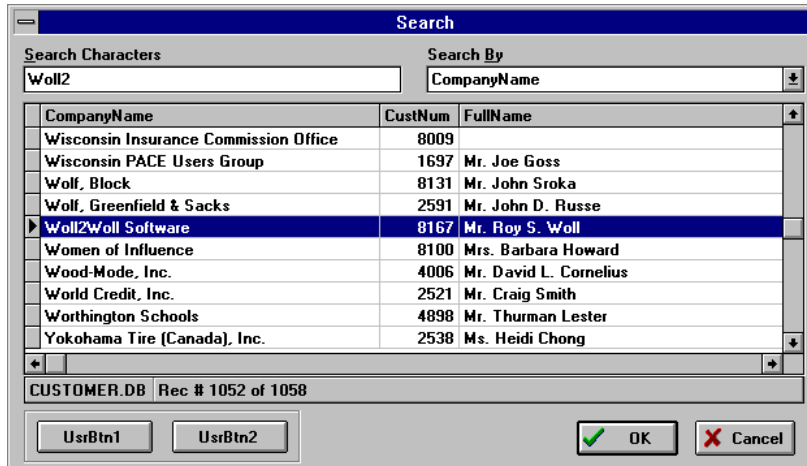


Figure 5.26 - An example TwwSearchDialog box in action, where the user has already selected one of the table's secondary Indexes (CompanyName) and entered the Search Characters "Woll2". Optional developer-defined buttons are also shown along with the opShowStatusBar option set to True.

Required supporting components

At least one TwwTable component—two TwwTable components if a shadow table is used.

Added Properties

Caption

This property contains a text value that is displayed in the search dialog box's title bar. The default value is "Search".

Data Type: String

CharCase

This property defines what case the characters typed in by the user are to take. ecLowerCase converts all characters to lower case. ecNormal allows the user to type both upper and lower case characters. ecUpperCase converts all characters to upper case. The default value is ecNormal.

Data Type: Constant

Valid Values: ecLowerCase, ecNormal, ecUpperCase

GridColor

This property defines the background color of the grid. The default value is clWhite. (When one or more columns of a grid are *fixed*, their colors are the same colors used for the grid's column headings defined by *TitleColor*.)

Data Type: TColor

GridOptions

This property contains a set of standard Delphi grid options.

Data Type: TSet()

Valid Values: Valid Delphi grid options

GridTitleAlignment

Determines the text alignment of titles in popup-dialog's grid. The default value is taLeftJustify.

Data Type: Constant

Valid Values: taCenter, taLeftJustify *or* taRightJustify

MaxHeight

Defines the maximum Height of the grid in the related dialog. Use this property to control the height of the popup-dialog. The default value for a standard VGA display (640 x 480) is 209.

Data Type: Integer

Valid Values: Positive integer value

MaxWidth

This property defines how wide the dialog box is allowed to grow, in pixels. The default value is 0, which allows the dialog box to grow to the entire width of the screen.

Data Type: Integer

Valid Values: Depends on your screen's display resolution

Options

This property contains a set of Boolean values that control the appearance of the dialog box. See the TwwLookupDialog *Options* property for a detailed description.

Data Type: TSet()

Valid Values: opShowOK, opShowSearchBy, opGroupControls, opFixFirstColumn and opShowStatusBar

PictureMaskFromDataSet

This property is only relevant if your datasource is attached to a TwwTable, TwwQuery, TwwQBE, or TwwClientDataset component, as it is always treated as false in other cases.

When customizing the picture masks through the select fields dialog (invoked by clicking on the selected property at design time), the mask information is stored in the related dataset if this property is True. Otherwise the mask information is stored as a property in the related visual component. By storing the mask information in the dataset, you do not need to re-enter the picture mask for other visual controls attached to this same database field.

Data Type: boolean

PictureMaskFromField

Setting this property to True will allow the dialog to automatically use the picture mask defined for the database field when the end-user is entering the text to search for.

Data Type: Boolean

PictureMasks

The assigned picture mask information is stored in this property if `PictureMaskFromDataset` is false. See the `PictureMaskFromDataSet` property.

Data Type: TStrings

SearchTable

This property defines the TDataSet component that the search should use for synchronization (the table used to retrieve data displayed to the user). The `ShadowSearchTable` property, described below, is the TDataSet component used internally to actually perform the search operation.

Using a different table component internally allows you to select the fields to be displayed in the grid along with allowing your end-users to navigate within the SearchDialog grid and even Cancel the search operation. All without affecting the visual interface component's record position. The default value is blank.

Data Type: TDataSet

Selected

Clicking the "..." button or double-clicking the SearchDialog component displays the Select Fields dialog box. This dialog box allows you to select the fields you want displayed in the grid, their titles, widths, control types and link information. (See *Using the Select Fields Dialog Box* at the beginning of Chapter 4.) The default value is *all* fields selected, using the *field name* as it's title, displayed as a *Field* control for a width equal to the number of characters in the field or the title, whichever is longer.

Data Type: (Internal to InfoPower)

Valid Values: (Internal to InfoPower)

ShadowSearchTable

Optional This property defines the TDataSet component to be used internally by the SearchDialog component. If not specified then all navigational operations are applied directly to the table specified in the `SearchTable` property, as well as all field selections originating from the SearchTable. This should be a different TwwTable component than the one specified in SearchTable, but still point to the same physical table.

Using a different table component internally allows the developer to select the fields displayed in the grid, along with allowing the user to navigate within the SearchDialog grid and even Cancel the search operation without affecting the visual interface component's record position. The default value is blank.

Data Type: TDataSet

Tag

This is the standard Delphi *Tag* property that you can use for your own internal processing needs. The default value is 0.

Data Type: Long

UserButton1Caption

When you want to display this button on the dialog box, enter the caption text for the button here and then add code to the *OnUserButton1Click* event. The default value is blank.

Data Type: String

Valid Values: Any text value that fits within the width of the button.

UserButton2Caption

When you want to display this button on the dialog box, enter the caption text for the button here and then add code to the *OnUserButton2Click* event. The default value is blank.

Data Type: String

Valid Values: Any text value that fits within the width of the button.

UseTFields

When the UseTFields property is set to true the *Selected* property's display settings are stored and retrieved from the ShadowSearchTable or the SearchTable. When it is set to False the *Selected* property's display settings are stored with the TwwSearchDialog. The default is True.

Data Type: Boolean

Modified properties

None.

Required property assignments

SearchTable

Added Events

Some of the following events pass a handle to the form containing all of the components of the dialog. To see what objects are contained within this editing form, open up *wwidlg.pas* in the InfoPower source sub-directory. If you do not have the source code version of InfoPower, then perform the steps in Chapter 4's topic "Determining the object names of the controls contained in an InfoPower dialog" on the *wwidlg.dfm* file contained in the InfoPower lib directory.

If you want to customize any of the objects contained by the form you can use the *OnInitDialog* event. However if all you are trying to do is to change the labels and hints, then use the *TwwIntl | SearchDialog* property.

OnInitDialog

Allows you to completely customize every aspect of the dialog box or perform some action during the initialization of the dialog box. When using this event, your code must reference

wwidl in your source file's Uses clause. This gives you access to all the components in the dialog. For example, you can modify the grid's properties, define custom events, etc.

Example: The following code tells the first user-defined button to show a hint when the user moves the mouse pointer over the button:

```
procedure TForm1.wwDBLookupComboDlg1InitDialog(
  Dialog: TwwLookupDlg);
begin
  Dialog.UserButton1.Hint := 'Hint for user button 1';
  Dialog.UserButton1.ShowHint := True;
end;
```

OnCloseDialog

This event allows you to perform any custom action before the dialog is actually closed.

OnPerformCustomSearch

When using a large dataset from a remote server, the performance of the dialog's incremental searching can significantly degrade. To resolve this issue, InfoPower adds a new event where you can control the specific action that takes place after the user types a character, or when the control needs to look up a value. In particular the custom action can update the query to only return the records that you are interested in. When using this event, your code is responsible for manipulating the lookuptable based on the parameter values passed in. See the TwwDBLookupCombo OnPerformCustomSearch event for a description of the events parameters.

OnSortChange

If you want to perform some custom action when the end-user makes a selection from the SortBy combo, then place your custom code here.

OnSyncDataSets

Use this event to synchronize the search table and shadow table to point to the same record. When using TwwTable components, synchronization is automatically performed. However when using other TDataSet components, you must attach code to this event to perform the synchronization. The parameters for this event are as follows.

Sender: TObject TwwSearchDialog associated with event

MoveDataSet: TDataSet DataSet that needs to be synchronized with the *BaseDataSet*

BaseDataSet : TDataSet DataSet that the MoveDataSet needs to move to.

Example: The following example synchronizes the datasets used by the SearchDialog when going against an TADOTable component. In this example OrderID is a unique field in the table.

```
procedure TForm1.wwSearchDialog1SyncDataSets(Sender: TObject;
  MoveDataSet,
  BaseDataSet: TDataSet);
begin
```

```
TADOTable(MoveDataSet).Locate('OrderID',  
    TADOTable(BaseDataSet).FieldByName('OrderID').asString, []);  
end;
```

OnUserButton1Click

When you want to display developer-defined button #1 on the dialog box, enter the caption text for the button in the *UserButton1Caption* property and then add code to this event that will be executed when the end-user clicks the button.

OnUserButton2Click

When you want to display developer-defined button #2 on the dialog box, enter the caption text for the button in the *UserButton2Caption* property and then add code to this event that will be executed when the end-user clicks the button.

Added Methods

Execute

Display the search dialog box to the end-user. False is returned if the user clicks the Cancel button or the user closes the dialog box via the control menu

How To

Use the SearchDialog with non-TTable components

See the example under the *OnSyncDataSets* event.

Use ADOTables with the SearchDialog.

To initialize the default value of the search-by combo in the dialog, you will need to preset the *IndexFieldNames* property of your *ShadowSearchTable* component. The *SearchDialog* will automatically default to this field when it is displayed. In order for the *SearchDialog* to display the sort-by control, you need to also set your *TADOTable*'s *CursorLocation* property to *clUseClient*.

Tips

- ◆ If you want to give your users some special functionality while the *Search* dialog box is being displayed, define one or both of the optional developer-defined buttons via the *UserButton1Caption* and *UserButton2Caption* properties. Then define your actions via the *OnUserButton1Click* and *OnUserButton2Click* events.
- ◆ If you are using SQL tables, refer to the *TwwTable* component's *SyncSQLByRange* property, and the SQL entries in the *Troubleshooting* section, for some very useful information that will assist you in working with SQL tables and *InfoPower*.
- ◆ If you have a choice use case insensitive indexes in your tables to make incremental searching more user-friendly.

- ◆ If you are using the `wwfilterdialog1` on the `SearchTable` and you wish the *ShadowSearchTable* to be filtered the same, then on the *OnInitDialog* event of the `TwwSearchDialog` you should put in the following code:

```
procedure TForm1.wwSearchDialog1OnInitDialog(  
    Dialog: TwwLookupDlg);  
begin  
    wwSearchDialog1.ShadowSearchTable.OnFilter :=  
        wwSearchDialog1.SearchTable.OnFilter;  
end;
```


TwwStoredProc



The non-visual TwwStoredProc component allows you to execute server stored procedures. Stored procedures can return either a singleton result or a multiple row result set to one or more of the other InfoPower visual interface components placed on your form.

Since InfoPower 2000, you can safely use the native TStoredProc instead of TwwStoredProc. Since InfoPower's TwwStoredProc component is inherited from Delphi's TStoredProc component, all standard Delphi component properties and functionality are still available, such as Delphi's built-in Fields editor and the Parameters Editor.

Ancestor

TStoredProc.

Required supporting components

None.

Added Properties

ControlType

This property holds information about the type of control used to display a field if the field is contained within a grid component. The default value is Field. (See *Using the Select Fields Dialog Box* at the beginning of Chapter 4.) To change this property at runtime, see the *SetControlType* method of the wwDBGrid component.

Data Type: (Internal to InfoPower)

Valid Values: (Internal to InfoPower)

LookupFields

Maintained for backward compatibility with earlier versions of InfoPower.

Data Type: (Internal to InfoPower)

Valid Values: (Internal to InfoPower)

LookupLinks

Maintained for backward compatibility with earlier versions of InfoPower.

Data Type: (Internal to InfoPower)

Valid Values: (Internal to InfoPower)

OnFilterOptions

See the documentation for *OnFilter* under the TwwTable component

PictureMasks

This property holds information about a field's picture mask. See *Using InfoPower's Picture Masks* in Chapter 4 for more details.

Data Type: TStrings

Valid Values: (Internal to InfoPower)

ValidateWithMask

See the documentation for *ValidateWithMask* under the TwwTable component

Modified properties

None.

Required property assignments

1) *DataSource* or *DatabaseName*, and 2) *StoredProcName*.

Added Events

OnFilter

See the documentation for *OnFilter* under the TwwTable component.

OnFilterEscape

See the documentation for *OnFilterEscape* under the TwwTable component.

OnInvalidValue

See *Using InfoPower's Picture Masks* in chapter 4.

Added Methods

wwFilterField

See the documentation for *wwFilterField* under the TwwTable component

How To

The TwwStoredProc component is inherited from Delphi's TStoredProc, so please refer to your Delphi manual for more information about this component. Since InfoPower's TwwStoredProc component is inherited from Delphi's TStoredProc component, you are provided with 100% backward compatibility. Thus, you can safely replace your use of TStoredProc with TwwStoredProc at any time.

TwwTable



The non-visual TwwTable component allows you to define the database table that supplies data to one or more of the other InfoPower visual interface components placed on your form. Since InfoPower's TwwTable component is inherited from Delphi's TTable component, all standard Delphi component properties and functionality are still available, such as double-clicking the component to invoke Delphi's built-in Fields editor. All data access for the TwwTable is still performed by the Delphi TTable.

Several properties, events and methods were added to TwwTable. For example, the Pack method allows you to *pack* any Paradox or dBASE table by adding only a single line of code to your program! InfoPower also includes picture masks at the table field level so invalid field values will be detected before your records are posted.

Ancestor

TTable.

Required supporting components

None.

Added Properties

ControlType

This property holds information about the type of control used to display a field if the field is contained within a grid component. The default value is Field. (See *Using the Select Fields Dialog Box* at the beginning of Chapter 4.) To change this property at runtime, see the *SetControlType* method of the wwDBGrid component.

Data Type: (Internal to InfoPower)

Valid Values: (Internal to InfoPower)

FilterCount

This property is maintained for backwards compatibility with earlier versions of InfoPower. This property is identical to the RecordCount property.

LookupFields

Maintained for backward compatibility with earlier versions of InfoPower.

Data Type: (Internal to InfoPower)

Valid Values: (Internal to InfoPower)

LookupLinks

Maintained for backward compatibility with earlier versions of InfoPower.

Data Type: (Internal to InfoPower)

Valid Values: (Internal to InfoPower)

NarrowSearch

This property affects how InfoPower's incremental searching is performed. When this property is True, incremental searching will show only those entries that match what has been typed in so far. For example, if the user types in the letter C, then all records starting with C are shown and no other records. When the user enters another character, the search becomes even narrower, or more specific.

When this property is False and using local tables, incremental searching moves to the record that most closely matches the letters typed in. If this property is False and you are using SQL tables, then incremental searching is dependent upon the *SyncSQLByRange* property setting. The default value of this property is False. See also the *NarrowSearchUpperChar* property.

Note: When this property is True and you are using a *TwwSearchDialog*, you should use a shadow table.

Data Type: Boolean

NarrowSearchUpperChar

This property is defines the behavior of linked field lookups on SQL tables, as well as the behavior of the incremental searching with the *NarrowSearch* property set to True. You should set this property to the highest ASCII value that your back-end recognizes.

In general you will never need to change this property unless your back-end does not understand the ASCII value of 255 for its data values. SQL server does not recognize the ASCII value of 255 in its SQL select statements, so you need to change this value to something SQL server recognizes, such as 122 (ASCII value for 'z'). The default value is 255.

Data Type: Word

Valid Values: Valid ASCII Ord value

OnFilterOptions

This property contains a set of boolean values that control the behavior of the *onFilter* event.

Data Type: set of *TwwOnFilterOption*;

Valid Values: *ofEnabled*, *ofShowHourGlass*, *ofCancelOnEscape*

<i>ofEnabled</i>	When False, the <i>onFilter</i> callback is ignored.
<i>ofShowHourGlass</i>	When True, this property will enable the HourGlass cursor to appear whenever an <i>OnFilter</i> is being applied on the dataset. For example when the user is scrolling in a grid against a filtered table, the hourglass will be displayed while the dataset is being searched and return to an Arrow when the search is complete.
<i>ofCancelOnEscape</i>	When True, this property allows the end-user to cancel a filter that is currently being applied. For instance if a filter is taking a long to

be applied, the end-user can abort the process by entering the <Esc> key. See also the OnFilterEscape event.

PictureMasks

This property holds information about a field's picture mask. See *Using InfoPower's Picture Masks* in Chapter 4 for more details.

Data Type: TStrings

Valid Values: (Internal to InfoPower)

Query

This property is preserved for backward compatibility with earlier versions of InfoPower. Previously this property was necessary if you wanted to fill a drop-down list from a query result. Since IP 3000 supports TwwQuery for its LookupTable in its lookup components, you should use the TwwQuery component instead of this property.

Data Type: TStrings

Valid Values: Any SQL or QBE query that produces a result set.

SyncSQLByRange

This property affects how InfoPower synchronizes two SQL tables and performs an incremental search when using the TwwSearchDialog, TwwLookupCombo, or TwwIncrementalSearch. By default, *SyncSQLByRange* is *False*, which tells the search components to use the Delphi Locate method, and the SearchDialog to use the GotoCurrent method to synchronize two tables.

By setting *SyncSQLByRange* to *True*, you may experience much faster table-to-table synchronization and incremental searching, even for very large tables, because ranges are used for searching. However the navigating range of the applied table starts at the first displayed record (the sync value), instead of at the first record in the table. *Thus, you cannot navigate or scroll backwards to a record before the first displayed record.*

As a rule of thumb, set *SyncSQLByRange* to *True* when searching large tables.

Data Type: Boolean

ValidateWithMask

When *True*, assigned picture masks are used to validate field values before they are posted. When *False*, this validation is skipped. You may want to set this property to *False* when using the TField's *OnGetText* and *OnSetText* events to map the stored value to a different displayed value.

Data Type: Boolean

wwFilter

Maintained for backwards compatibility with earlier versions of InfoPower. For expression filtering see the Delphi Filter and Filtered properties.

Modified properties

None.

Required property assignments

DatabaseName and TableName.

Added Events

OnFilter

This event is executed for records in the related table which allows you to have practically unlimited filtering capabilities. Via this callback filter function, you simply set the parameter *Accept* to *False* if you want the record to be excluded. In addition, this event respects other ranges and filters that might currently be set on the related table (i.e. *SetRange* method and *Filter* property). You can also filter on InfoPower linked fields or Delphi's lookupfields with this event.

Parameters

<i>Table</i> : TwwTable	Table being filtered
<i>Accept</i> : Boolean	Set to <i>False</i> to exclude a record. Defaults to <i>True</i> .

Delphi has the event named *OnFilterRecord*, which has similar functionality. However Delphi's *OnFilterRecord* does not work with lookup fields or memo fields. We recommend you use the *OnFilter* event if you need to filter on lookup or memo fields.

With the InfoPower *OnFilter* event, you can compare two database fields with each other (for example: *Field1 < Field2*), do bitwise comparisons in fields, have filters dependent upon other related tables, or anything else you can express in code. This event is most practical when used against local tables (Paradox or dBASE) because when used against SQL tables, the back-end database system is prevented from optimizing the filter, since every record is passed to this event.

Disabling the onFilter event: You can disable a previously defined *OnFilter* event during program execution by setting the table's *OnFilterOptions* | *ofoEnabled* property to *False*. For example:

```
wwtable1.onFilterOptions := wwtable1.onFilterOptions - [ofoEnabled];
```

Refreshing the filtered table: If you change a variable that your *OnFilter* event depends upon, then you should also call the table's refresh method. In general, call the table's refresh method to force the *OnFilter* to be re-processed.

Allowing the end-user to cancel a filter in progress: By setting the *onFilterOptions* | *ofoCancelOnEscape* property to *True*, the end-user can cancel a filter in progress by entering the <ESC> key. You can also display an informational message after the filter has been cancelled by using the *onFilterEscape* event.

Notes: The related *wwFilterField* method should be used to extract the contents of the fields you use. Do **not** use the *FieldByName* method to access the field data within this event because it will **not** contain the correct data while this event is being executed.

Example 1: The following example performs a pattern search. The code shows only the records that contain the string 'System' in the "Company Name" field:

```
procedure TForm1.wwTable1Filter(table: TwwQBE; var Accept: Boolean);
begin
  with table do
    accept := pos('System',wwFilterField('Company Name').AsString) <> 0;
end;
```

Example 2: The following example shows all records that have an odd number for their Customer No.:

```
procedure TForm1.wwTable1Filter(table: TwwTable; var Accept: boolean)
begin
  with table do
    accept := odd(wwFilterField('Customer No').AsInteger);
end;
```

Example 3: The following example shows all employees who started a job and quit before 30 days had elapsed:

```
procedure TForm1.wwTable1Filter(table: TwwTable; var Accept: boolean):
begin
  with table do
    accept := wwFilterField('DateStartedJob').AsDate
      - wwFilterField('DateQuitJob').AsDate > 30
end;
```

OnFilterEscape

This event is fired after the end-user has cancelled a filter in progress by pressing the <Esc> key. You may wish to use this event to display an informational message to the user so that they are aware they have cancelled the filter. See also the *onFilter* event.

OnInvalidValue

See *Using InfoPower's Picture Masks* in chapter 4.

Added Methods

FilterActivate

Maintained for backward compatibility with earlier versions of InfoPower. For expression filtering see the Delphi *Filter* and *Filtered* properties.

Pack

Allows you to *pack* both Paradox and dBASE tables interactively from within your program. The function definition is:

```
Function Pack(var statusMsg: string): Boolean;
```

Example: The following source code example demonstrates how to pack the table defined in a TwwTable component named InvoiceTable...

```
String rtnMsg;  
  
if not InvoiceTable.Pack(rtnMsg) then  
  {Your error processing code goes here...}  
  MessageDlg('Pack failed with message: '+rtnMsg+'.', mtError, [mbOK], 0);
```

RefreshLinks

Maintained for backward compatibility.

SetLookupField

This method allows you to update lookup fields in the InfoPower grid.

```
Function SetLookupField(Field: TField): boolean;
```

This method should be called in the lookup TField.OnChange event to provide an updateable 1 to 1 relationship. You will also need to set the Grid's EditCalculated property to True so that the grid will allow edits in the lookup field. See the example in the TwwDBGrid how-to section, under editing a Lookup Field.

wwFilterField

This method must be used only during processing of the OnFilter event. wwFilterField returns the contents of any field, contained in the currently filtered record, in the format you specify via the following properties:

<u>Property</u>	<u>Return Value</u>	<u>Rtn Value Data Type</u>
AsBoolean	Field's contents as a boolean	Boolean
AsFloat	Field's contents as a float	Double
AsInteger	Field's contents as an integer	LongInt
AsString	Field's contents as a Pascal string	String
AsDate	Date portion of the field's value	TDateTime
AsTime	Time portion of the field's value	TDateTime
AsDateTime	Field's value as a TDateTime	TDateTime
AsCurrency	Field 's value as currency	Double
IsNull	True if the field is Null	Boolean

Example: Please refer to the OnFilter event examples.

wwFindKey

Maintained for backwards compatibility. Use the Delphi FindKey method instead.

How To

Performing case-insensitive filters, or substring filters.

If you need this ability then you should use the TwwTableOnFilter event. Similarly, use the onFilter event if you need your filter to search on a substring anywhere in the field

Partially simulate the Paradox “..” wildcard characters in a filter criteria:

See example 1 in the *onFilter* event documentation.

Tips

- ◆ To extract only specific fields from the table, double-click the TwwTable component to invoke the Delphi Fields editor and select only those fields you want to extract.
- ◆ To change the *display order* of records associated with this table, use the *IndexName* property to select a valid secondary index. For SQL servers, use the *IndexFieldNames* property. Records are now displayed in the order of the Index you chose.
- ◆ To modify field-level properties that are not included in the Select Fields dialog box, such as alignment, display format, edit mask, etc., use the Object Inspector. If the field is not listed in the Object Inspector, select it via Delphi’s Fields editor window (double-click the TwwTable component).

By default all fields in a table are selected for retrieval, but they are not listed in the Object Inspector where you can modify their properties. To add all fields to the Object Inspector, click the Add button of the Fields editor, make sure all fields are highlighted and then click the OK button. You will now be able to select an individual field in the Object Inspector and modify its properties.

- ◆ Since InfoPower’s TwwTable component is a direct descendent of Delphi’s TTable component, you are provided with 100% backward compatibility. Thus, you can safely replace your use of TTable with TwwTable at any time.

Troubleshooting

We recommend you visit our newsgroups at <http://www.woll2woll.com>, as the newsgroup contains thousands of messages discussing InfoPower, getting the most out of InfoPower, and troubleshooting InfoPower. Also be sure to check the useful sites at <http://www.tamaracka.com/search.htm> and <http://www.mers.com/searchsite.html>, as it contains a database of InfoPower newsgroup threads that provide useful tips and solutions.

This chapter provides troubleshooting assistance when you are having problems working with specific InfoPower components. Please check through this list before calling our technical support department because it could save you some time and the cost of a long-distance phone call. Components in this section are listed in alphabetical order.

General

- ◆ **Problem:** I am getting some consistent unexplainable GPFs or system crashes when I run my application.
Solution: Make sure your project stack size is sufficiently high. We deem Delphi's 16K default to be inadequate in most cases and strongly recommend that you raise this value to 24K (6000 Hex).

Options | Project | Linker | Min Stack Size 0x00006000

SQL Tables - Performance

- ◆ **Problem:** When using a TwwSearchDialog, TwwLookupDialog or a TwwDBLookupComboDlg component, it's taking a long time to open the dialog and a long time to return from the dialog.
Solution: Set the TwwTable's *SyncSQLByRange* property to True. When False, table synchronization on large SQL tables is very slow. See *SQL Tables - Navigating* below for more details about this property.

SQL Tables - Navigating/Backward scrolling

- ◆ **Problem:** After locating a record via the `TwwSearchDialog`, `TwwIncrementalSearch`, or `TwwDBLookupCombo`, I'm not allowed to scroll back to prior records or use the `TDBNavigator` to move to prior records.
Solution: If the `TwwTable`'s `SyncSQLByRange` property is `True`, SQL table synchronization will move to the selected record by using the `TwwTable`'s `SetRange` method. This allows for very fast SQL table synchronization, but you cannot scroll backwards to a record prior to the first one located. If you really need the ability to move to prior records, set the `SyncSQLByRange` property to `False`. However, be aware that SQL table synchronization will become quite slow.

SQL Tables - When using a TwwDBGrid

- ◆ **Problem:** My `TwwDBGrid`'s display attributes seem to get reset whenever I run the program.
Solution: Set your `Grid`'s `UseTFields` property to `False`

TwwDBGrid

- ◆ **Problem:** I added a `TwwDBLookupCombo` to my grid. When the `LookupCombo` is active (currently selected cell), it displays the correct data from the lookup table, but the rest of the rows in the grid (non-active) display the lookup value instead. How do I get the grid to display the field from the other table in all the rows and not just for the row where the `LookupCombo` is active.
Solution: Define a Delphi lookup field as described in the Delphi documentation, and then bind your `lookupcombo` control to this `lookupfield` column.
- ◆ **Problem:** During program execution I am assigning the `DataSet` property of a `TDataSource`. However after I assign the `datasource`, the related `TwwDBGrid` seems to have lost most of its fields.
Solution: Before assigning the `DataSet` property of the `TDataSource`, first clear the grid's `selected` property. The grid then will automatically display the visible fields of the `NewTable`.

```
wwDBGrid1.Selected.Clear;  
wwDBGrid1.DataSource.DataSet := NewTable;
```

TwwDBLookupCombo

- ◆ **Problem:** The drop-down list does not seem to be able to scroll backwards when going against SQL or ODBC tables.
Solution: Change your `lookuptable`'s `SyncSQLByRange` property to `False`. See the documentation for `SyncSQLByRange` for more details.

- ◆ **Problem:** No data is being displayed in the drop-down list.
Solution: Select the fields to be displayed by using the Select Fields dialog box—click the “...” button in the *Selected* property (refer to *Using the Select Fields Dialog Box* at the beginning of Chapter 4 for details about using this dialog box).

TwwKeyCombo

- ◆ **Problem:** I’m getting a “DataSource cannot be a child table” Warning box.
Solution: It’s telling you that the *DataSource* property of this component is trying to reference a child table of a Master/Child relationship, which is not legal. You can assign a TwwKeyCombo’s *DataSource* only to a Master table. If you get this error when using the SearchDialog or LookupDialog you can avoid this problem by setting the property *Option | OpShowSearchBy* to *False*.
- ◆ **Problem:** The TwwKeyCombo component appears to only display indexes based on fields that are currently set to 'Visible = True'. I regularly combine several fields into a single calculated field for display in a grid and make the original fields invisible.
Solution: You can add indexes that InfoPower didn't choose by overriding the *OnEnter* event and adding all other fields the user should be able to select the index for. For example, the following code adds the index associated with the field 'City' to wwKeyCombo1 and makes sure that the index field is only added one time:

```

Procedure TIncrSearch.wwKeyCombo1Enter(Sender: TObject)
begin
  if (wwKeyCombo1.Items.IndexOf('City') < 0) then
    wwKeyCombo1.Items.Add('City');
end;

```

TwwIncrementalSearch

- ◆ **Problem:** When typing characters to be located, the value is not found, even though I know it exists.
Solution: This component’s case sensitivity is dependent upon the table index’s case sensitivity. If the index is defined as Case Sensitive, then the user needs to enter case-sensitive characters, both upper and lower case characters, in the exact order in which they appear in the value of the index. For example, if the user wants to locate a value of “San Jose” in the City field of their Customer table, they must enter an upper case “S” and then a lower case “a” in order to locate “Sa”, and then continue entering each character with its proper capitalization. To assist the user, you may wish to assign a picture mask to the field so that the first letter is automatically converted to uppercase

TwwSearchDialog

- ◆ **Problem:** I'm getting an error message telling me that a DataSource or DataSet is required or that the DataSource has no DataSet.
Solution: Make sure you have placed two separate TwwTable components on your form and assigned them to the *SearchTable* and *ShadowSearchTable* properties, as described in the *Added properties* section of this component.

TwwDBLookupComboDlg, TwwSearchDialog, TwwLookupDialog

- ◆ **Problem:** I am trying to have the user button terminate the dialog.
Solution: Set the ModalResult in your user button to mrOK as in the following.

```
(Sender as TForm).ModalResult := mrOK;
```
- ◆ **Problem:** When using the TwwSearchDialog, how can I access the search text in order to be able to automatically (or by pressing one of the user buttons) add a record to the lookup table based on that string.
Solution: Your code must reference wwIdlg in your Uses clause. This gives you access to all the SearchDialog's components. For instance the following code shows the current text in the incremental search component of the TwwSearchDialog.

```
procedure TSearchForm.wwSearchDialog1UserButton1Click(  
  Sender: TObject;  
  LookupTable: TDataSet);  
begin  
  with (Sender as TwwLookupDlg) do  
    ShowMessage(wwIncrementalSearch1.text);  
end;
```
- ◆ **Problem:** I'm using TwwLookupDialog and the TwwSearchDialog and can't find out how I can tell if the end-user clicked on OK or Cancel.
Solution: The dialog's *execute* method returns a Boolean (True or False). True means the user clicked OK, and False means Cancel.
- ◆ **Problem:** When using the *OnUserButton1Click* event of the TwwSearchDialog or TwwLookupDialog, how do I set the focus so it is on a different control than the button.
Solution: You can access all the components in the TwwSearchDialog, TwwLookupDialog, or TwwDBLookupComboDlg by doing the following.
 1. Include wwIdlg in your uses clause
 2. Cast the sender parameter as TwwLookupDlg
The following example uses UserButton1 to insert a new record into the search table and then sets the focus to the grid in the related wwSearchDialog.

```

procedure TGridDemo.wvSearchDialog1UserButton1Click(
  Sender: TObject;
  LookupTable: TDataSet);
begin
  with Sender as TwvLookupDlg do begin
    LookupTable.insert;
    wvdbgrid1.setFocus;
  end
end;

```

For more information on the component names within the dialog, please refer to the *OnInitDialog* event.

- ◆ **Problem:** I've been looking at the *OnInitDialog* method of the *TwvDBLookupComboDlg*, *TwvLookupDialog*, and *TwvSearchDialog*, but can't seem to find the right code to make the memo field in the lookup grid shows.

Solution: The following example sets the *MemoAttributes* property of the grid on the dialog so that memos appear in the pop-up search dialog. First, add *wvMemo* to your form's *uses* clause.

```

procedure TForm1.wvDBLookupComboDlg1OnInitDialog(
  Dialog: TwvLookupDlg);
var grid: TwvDBGrid;
begin
  grid := Dialog.wvDBGrid1;
  grid.MemoAttributes := grid.MemoAttributes + [mGridShow];
  grid.MemoAttributes := grid.MemoAttributes + [mDisableDialog];
end;

```

TwvTable

- ◆ **Problem:** I'm trying to use the *Pack* method, but keep getting the message "Pack failed. Table is busy." even with the table's *Active* property set to *False*. What else do I need to do?
Solution: Packing requires that the table's *active* property be *False*, so it can gain exclusive access to the table. You will need to make sure that the table's *active* property is set to *False* at design time so that the Delphi IDE does not open the table. Setting *active* to *False* at run-time isn't equivalent, as the Delphi IDE itself may already have this table open.
- ◆ **Problem:** I'm currently running into a problem using *GotoCurrent* on filtered tables, I'm getting the message 'no current record'.
Solution: No current record means that you are referencing a record that no longer appears in the filtered table. Make sure you reference a record that would be in the filtered table.

Index

- ..
- "Error creating cursor handle", 252
- 1**
- IstClass
 - advantages, 61
 - integration, 61
- A**
- ActiveEdit property
 - TwwDataInspector, 54
- ActiveItem property
 - TwwDataInspector, 54
 - TwwInspectorItem, 69
- ActiveRecordColor property
 - TwwDataInspector, 59
- ActiveRows property
 - TwwDataInspector, 54
- Add method
 - TwwInspectorCollection, 71
 - TwwNavButtons, 177
- Add property
 - example, 71
- AddFieldInfo method
 - TwwFilterDialog, 219
- AddInfoPowerDialogs
 - TwwNavButtons, 177
- AddItem method
 - TwwDBComboBox, 86
- ADO property
 - TwwIntl, 225
- Alignment property
 - TwwCheckBox, 44, 45
 - TwwDBDateTimePickerCalendarAttributes, 94
 - TwwInspectorItem, 73
 - TwwRadioButton, 255
- All | Searched,
 - TwwFilterDialog, 206
- AllowClearKey property
 - TwwDBComboBox, 81
 - TwwDBLookupCombo, 152
 - TwwDBLookupComboDlg, 162
 - TwwInspectorItem.PickList, 76
- AllowGrayed property
 - TwwCheckBox, 44
 - AllowInvalidExit property, 35
 - AlternatingRowColor property
 - TwwDataInspector, 59
 - AlternatingRowRegions property
 - TwwDataInspector, 59
 - AlwaysTransparent property
 - TwwCheckBox, 45
 - TwwRadioButton, 255
 - AndChar property
 - TwwFilterDialog, 209
 - AnswerTable property
 - TwwQBE, 248
 - Appending to a rich edit, 191
 - AppendRichEditFrom
 - TwwDBRichEdit, 191
 - ApplyFilter method
 - TwwFilterDialog, 219
 - ApplyFrame
 - TwwController, 50
 - ApplyList method
 - TwwDBComboBox, 86, 88
 - ApplySelected method
 - TwwDBGrid, 136
 - attaching icon to indicator button, 145
 - AutoDropDown property
 - TwwDBComboBox, 81
 - TwwDBLookupCombo, 152
 - TwwDBLookupComboDlg, 162
 - AutoEnableEdit property
 - TwwDBComboDlg, 90
 - AutoFill property
 - PictureMasks, 35
 - AutoFillDate property
 - TwwDBEdit, 99
 - AutoHideExpandButton property
 - TwwExpandButton, 201
 - AutoShrink property
 - TwwExpandButton, 201
 - AutoSizeHeightAdjust property
 - EditFrame, 23
 - Frame, 23
 - AutoSizeStyle,
 - TwwDBNavigator, 173
 - Auto-sizing columns
 - TwwDBGrid, 102
 - AutoURLDetect property
 - TwwDBRichEdit, 181
 - AuxiliaryTables property
 - TwwQBE, 249
 - B**
 - BackgroundBitmap property
 - TwwDataInspector, 59
 - BackgroundDrawStyle property
 - TwwDataInspector, 59
 - BackgroundOptions property
 - TwwDataInspector, 59
 - BeginUpdate method
 - TwwDataInspector, 67
 - BetweenEditsInRow property
 - TwwRecordViewDialog, 263
 - TwwRecordViewPanel, 276
 - BetweenLabelEdit property
 - TwwRecordViewDialog, 263
 - TwwRecordViewDialog, 263
 - TwwRecordViewPanel, 276
 - BetweenRow property
 - TwwRecordViewDialog, 263
 - TwwRecordViewPanel, 276, 277
 - Bitmap in grid, 30, 31
 - Bitmaps in a TwwDBRichEdit, 179
 - bitmaps in dialog buttons
 - TwwIntl, 227
 - bitmaps in grids, 148
 - BlankAsZero property
 - TwwQBE, 249
 - BorderStyle property
 - TwwRecordViewDialog, 262
 - Bottom margin
 - TwwDBRichEdit, 188
 - BottomOffset property
 - TwwRecordViewDialog, 264
 - TwwRecordViewPanel, 277
 - building custom packages in Delphi 5, 14
 - button effects
 - Flat property, 25
 - key properties, 25
 - supporting components, 22
 - Transparent property, 25
 - ButtonAlignment property

- TwwExpandButton, 201
 - ButtonEffects property
 - TwwController, 50
 - TwwDBComboBox, 81
 - TwwDBComboDlg, 90
 - TwwDBDateTimePicker, 94
 - TwwDBLookupCombo, 152
 - TwwKeyCombo, 229
 - ButtonFrame property
 - TwwRadioGroup, 258
 - ButtonGlyph property
 - TwwDBComboBox, 81
 - TwwDBComboDlg, 90
 - TwwDBDateTimePicker, 94
 - TwwDBLookupCombo, 152
 - ButtonOptions property
 - TwwDataInspector, 54
 - Buttons property,
 - TwwDBNavigator, 173
 - ButtonStyle property
 - TwwDBComboBox, 81
 - TwwDBComboDlg, 90
 - TwwDBDateTimePicker, 94
 - TwwDBLookupCombo, 152
 - TwwInspectorItem.PickList, 76
 - ButtonWidth property
 - TwwDBComboBox, 81
 - TwwDBComboDlg, 91
 - TwwDBDateTimePicker, 94
 - TwwDBLookupCombo, 153
 - By Range, TwwFilterDialog, 206
 - By Value, TwwFilterDialog, 206
- C**
- CalcCellCol property
 - TwwDBGrid, 108
 - CalcCellRow property
 - TwwDBGrid, 108
 - CalColors property
 - TwwDBMonthCalendar, 169
 - CalendarAttributes property
 - TwwDBDateTimePicker, 94
 - calling recordview from the grid, 146
 - Cancel buttons
 - Setting the caption, 225
 - CanCut method
 - TwwDBRichEdit, 191
 - CanFindNext method
 - TwwDBRichEdit, 191
 - CanPaste method
 - TwwDBRichEdit, 191
 - CanRedo method
 - TwwDBRichEdit, 191
 - CanUndo method
 - TwwDBRichEdit, 191
 - Canvas property
 - TwwDataInspector, 55
 - Caption property
 - TwwCheckBox, 45
 - TwwDBLookupComboDlg, 162
 - TwwFilterDialog, 208
 - TwwInspectorItem, 73
 - TwwLocateDialog, 232
 - TwwLookupDialog, 236
 - TwwMemoDialog, 243
 - TwwNavButton, 174
 - TwwRadioButton, 255
 - TwwRecordViewDialog, 262
 - TwwSearchDialog, 283
 - CaptionColor property
 - TwwDataInspector, 54
 - CaptionFont property
 - TwwDataInspector, 55
 - CaptionIndent property
 - TwwDataInspector, 55
 - CaptionWidth property
 - TwwDataInspector, 55
 - carriage return to tabs, 147
 - Case insensitive filters
 - TwwTable, 296
 - CaseSensitive property
 - TwwLocateDialog, 232
 - cbStyleAuto, TwwIntl, 225
 - cbStyleCheckmark, TwwIntl, 225
 - cbStyleXmark, TwwIntl, 225
 - CellHeight property
 - TwwInspectorItem, 73
 - Changing a column's width, 130
 - CharCase property
 - TwwLookupDialog, 236
 - TwwSearchDialog, 283
 - Checkbox in grid, 30
 - CheckBoxInGridStyle property
 - TwwIntl, 225
 - Checked property
 - TwwCheckBox, 45
 - TwwInspectorItem, 73
 - TwwRadioButton, 255
 - CheckNewFields property
 - TwwDBGrid IniAttributes, 115
 - Clear method
 - TwwIncrementalSearch, 224
 - TwwNavButtons, 177
 - ClearFilter method
 - TwwFilterDialog, 219
 - ClearHistory method
 - TwwDBComboBox, 86
 - ClearParams method
 - TwwQBE, 251
 - Click method
 - TwwNavButton, 177
 - client datasets, 48
 - CollapseGlyph,
 - TwwDataInspector, 54
 - color alternating rows in grid, 147
 - Color property
 - TwwDBDateTimePicker.CalendarAttributes, 94
 - Coloring of a grid cell during editing, 36
 - Coloring of a TwwDBEdit during editing, 36
 - Coloring the cells in a grid, 127
 - Coloring the titles in a grid, 127
 - column headings. *See* TitleAlignment and TitleColor properties
 - Column1 Width property
 - TwwDBComboBox, 81
 - ColumnByName method
 - TwwDBGrid, 136
 - ColWidthsPixels property
 - TwwDBGrid, 108
 - Combo box
 - TwwDBComboBox, 80
 - Combo box dialog, 90
 - compatibility issues with InfoPower 4, 12
 - Component Hierarchy, 15, 16, 17
 - component overview, 15
 - Connected property
 - TwwIntl, 225
 - ControlInfoInDataset
 - TwwDBGrid, 109
 - ControlInfoInDataset property
 - TwwRecordViewDialog, 262
 - TwwRecordViewDialog, 262
 - ControlInfoInDataSet property
 - TwwRecordViewPanel, 275
 - controller, 22, 50

- ControlOptions property
 - TwwRecordViewDialog, 262
 - TwwRecordViewPanel, 275
 - ControlType property
 - TwwClientDataSet, 48
 - TwwDBGrid, 109
 - TwwQBE, 249
 - TwwQuery, 253
 - TwwRecordViewDialog, 263
 - TwwRecordViewPanel, 276
 - TwwStoredProc, 289
 - TwwTable, 291
 - CopyRichEditFromBlob method
 - TwwDBRichEdit, 192
 - CopyRichEditTo method
 - TwwDBRichEdit, 192
 - CopyRichEditToBlob method
 - TwwDBRichEdit, 192
 - Count property
 - TwwNavButtons, 173
 - crashes, 299
 - custom control
 - within a grid, 144
 - Custom control in
 - grid or record-view, 30
 - custom framing
 - key properties and events, 22
 - properties, 22
 - recordviewdialog, 24
 - recordviewpanel, 24
 - supporting components, 22
 - custom framing and
 - transparency effects, 21
 - CustomControl property
 - TwwInspectorItem, 73
 - CustomControlAlwaysPaints property
 - TwwInspectorItem, 74
 - CustomControlHighlight property
 - TwwInspectorItem, 74
- D**
- data inspector, 51
 - collection, 71
 - item, 73
 - data viewing order. *See* TwwKeyCombo
 - DataField property
 - TwwDBDateTimePicker, 95
 - TwwDBLookupCombo, 156
 - TwwDBLookupComboDlg, 162
 - TwwDBMonthCalendar, 169
 - TwwDBRichEdit, 181, 182
 - TwwInspectorItem, 74
 - TwwMemoDialog, 243
 - DataSetFilterType property, TwwFilterDialog, 211
 - DataSource cannot be a child table, 301
 - DataSource property
 - TwwDataInspector, 55
 - TwwDBDateTimePicker, 95
 - TwwDBGrid, 109
 - TwwDBLookupCombo, 156
 - TwwDBLookupComboDlg, 162
 - TwwDBMonthCalendar, 169
 - TwwDBNavigator, 175
 - TwwExpandButton, 202
 - TwwFilterDialog, 208
 - TwwIncrementalSearch, 222
 - TwwInspectorItem, 74
 - TwwKeyCombo, 229
 - TwwLocateDialog, 232
 - TwwMemoDialog, 244
 - TwwRecordViewDialog, 263
 - TwwRecordViewPanel, 276
 - Date property
 - TwwDBDateTimePicker, 95
 - TwwDBMonthCalendar, 169
 - Date time picker component, 93
 - DateFormat property
 - TwwDBDateTimePicker, 95
 - DBEdit. *See* TwwDBEdit
 - DBNavigator, 173
 - DefaultButton property
 - TwwLocateDialog, 232
 - DefaultEpochYear property
 - TwwIntl, 226
 - DefaultField property
 - TwwExpandButton, 202
 - TwwFilterDialog, 208
 - DefaultFilterBy property
 - TwwFilterDialog, 208
 - DefaultMatchType property
 - TwwFilterDialog, 209
 - DefaultRowHeight property
 - TwwDataInspector, 55
 - defining grid title attributes, 147
 - DeleteItem method
 - TwwDBComboBox, 86
 - deleting InfoPower. *See* uninstalling InfoPower
 - Delimiter property
 - TwwDBGrid
 - ExportOptions, 111
 - TwwDBGrid IniAttributes, 115
 - Design Picture Mask dialog, 39
 - Detecting cell movement, grid, 149
 - Detecting row change, grid, 149
 - dgAllowDelete, TwwDBGrid, 118
 - dgAllowInsert, TwwDBGrid, 118
 - dgColLinesDisableFixed, TwwDBGrid, 121
 - dgDbClickColSizing, TwwDBGrid, 121
 - dgEnterToTab, TwwDBGrid, 118
 - dgFixedEditable, TwwDBGrid, 120
 - dgFixedProportionalResize, TwwDBGrid, 121
 - dgFixedResizable, TwwDBGrid, 120
 - dgFooter3DCells, TwwDBGrid, 120
 - dgHideBottomDataLine, TwwDBGrid, 121
 - dgMultiSelect, TwwDBGrid, 119
 - dgNoLimitColSize, TwwDBGrid, 120
 - dgPerfectRowFit, TwwDBGrid, 119
 - dgProportionalColResize, TwwDBGrid, 120
 - dgRowLinesDisableFixed, TwwDBGrid, 121
 - dgRowResize, TwwDBGrid, 120
 - dgShowCellHint, TwwDBGrid, 120
 - dgShowFooter, TwwDBGrid, 120
 - dgTabExitsOnLastCol, TwwDBGrid, 120
 - dgTrailingEllipsis, TwwDBGrid, 120
 - dgWordWrap, TwwDBGrid, 119

- dialog boxes
 - TwwDBComboDlg, 90
 - TwwDBLookupComboDlg, 161
 - TwwFilterDialog, 205
 - TwwLocateDialog, 231
 - TwwLookupDialog, 236
 - TwwMemoDialog, 243
 - TwwRecordViewDialog, 261
 - TwwSearchDialog, 282
 - Dialog property
 - TwwNavButton, 174
 - DialogFontStyle property
 - TwwIntl, 226
 - DisableDefaultEditor property
 - TwwInspectorItem, 74
 - DisableDropDownList property
 - TwwDBComboBox, 82
 - DisableThemes, 41
 - DisableThemes property
 - TwwCheckBox, 45
 - TwwDataInspector, 55
 - TwwDBComboBox, 82
 - TwwDBDateTimePicker, 95
 - TwwDBEdit, 99
 - TwwDBGGrid, 109
 - TwwDBLookupCombo, 153
 - DisableThemesInTitle property
 - TwwDBGGrid, 109
 - Display Attributes
 - Select Fields dialog box, 28
 - display order. *See* TwwKeyCombo
 - DisplayAsCheckbox property
 - TwwInspectorItem.PickList, 76
 - Displayed values
 - TwwDBComboBox, 88
 - DisplayFormat property
 - TwwDBDateTimePicker, 95
 - DisplayText property
 - TwwInspectorItem, 75
 - DisplayValueChecked property
 - TwwCheckBox, 45
 - DisplayValueUnchecked property
 - TwwCheckBox, 45
 - distribution requirements, 12
 - Ditto Capability
 - TwwDBGGrid, 103
 - DittoAttributes property
 - TwwDBGGrid, 109
 - DittoDirection property
 - TwwDBGGrid.DittoAttribute, 109
 - DittoField method
 - TwwDBGGrid, 137
 - DlgHeight property
 - TwwMemoDialog, 244
 - DlgLeft property
 - TwwMemoDialog, 244
 - DlgTop property
 - TwwMemoDialog, 244
 - DlgWidth property
 - TwwMemoDialog, 244
 - DottedLineColor property
 - TwwDataInspector, 56
 - Dragging a column in a grid, 130
 - DragVertOffset property
 - TwwDBGGrid, 110
 - DrawCellInfo
 - TwwDBGGrid, 125
 - DropDown method
 - TwwDBLookupCombo, 159
 - Drop-down panels
 - TwwDBGGrid, 102
 - DropDownAlignment property
 - TwwDBLookupCombo, 153
 - DropDownCount property
 - TwwDBComboBox, 82
 - DropDownWidth property
 - TwwDBComboBox, 82
 - DroppedDown method
 - TwwDBComboBox, 86
 - DroppedDown property
 - TwwDBComboBox, 82
 - dynamic captions
 - TwwCheckBox, 47
- E**
- ecoCheckboxSingleClick, TwwDBGGrid, 110
 - ecoDisableCustomControls, TwwDBGGrid, 110
 - ecoDisableEditorIfReadOnly, TwwDBGGrid, 111
 - ecoSearchOwnerForm, TwwDBGGrid, 110
 - ecoUseDateTimePicker, TwwDBGGrid, 111
 - Edit Combo List Dlg Box using, 87
 - Edit Control
 - Select Fields dialog box, 30
 - Edit Control in grid
 - Checkbox, 30
 - CustomEdit, 30
 - ImageIndex, 30, 31
 - RichEdit, 30, 31
 - edit controls
 - custom framing, 21
 - picture masks, 32
 - transparency effects, 21
 - EditCalculated property
 - TwwDBGGrid, 110, 145
 - EditControlOptions property
 - TwwDBGGrid, 110
 - EditFrame properties, 22
 - EditFrame property
 - TwwRecordViewDialog, 263
 - TwwRecordViewPanel, 276
 - editing lookupfields in the grid, 145
 - editing memos in the grid, 149
 - EditorCaption property
 - TwwDBRichEdit, 182
 - EditorEnabled property
 - TwwDBSpinEdit, 197
 - EditorOptions property
 - TwwDBRichEdit, 182
 - EditorPosition property
 - TwwDBRichEdit, 183
 - EditText property
 - TwwInspectorItem, 75
 - EditWidth property
 - TwwDBRichEdit, 184
 - Enabled
 - TwwHistoryList, 82
 - Enabled property
 - EditFrame, 22
 - Frame, 22
 - TwwDBGGrid IniAttributes, 116
 - TwwInspectorItem, 75
 - EndDate property
 - TwwDBMonthCalendar, 169
 - EndUpdate method
 - TwwDataInspector, 67
 - enter to tab, 147
 - Enter to tab property
 - TwwRecordViewDialog, 266
 - Epoch property(Year 2000)
 - TwwDBDateTimePicker, 95
 - Epsilon property, TwwFilterDialog, 213
 - esoAddControls
 - TwwDBGGrid
 - ExportOptions, 112
 - esoBestColFit
 - TwwDBGGrid
 - ExportOptions, 112
 - esoClipboard

- TwwDBGrid
 - ExportOptions, 113
- esoDbQuoteFields
 - TwwDBGrid
 - ExportOptions, 112
- esoDynamicColors
 - TwwDBGrid
 - ExportOptions, 112
- esoEmbedURL
 - TwwDBGrid
 - ExportOptions, 113
- esoSaveSelectedOnly
 - TwwDBGrid
 - ExportOptions, 112
- esoShowAlternating
 - TwwDBGrid
 - ExportOptions, 113
- esoShowFooter
 - TwwDBGrid
 - ExportOptions, 112
- esoShowHeader
 - TwwDBGrid
 - ExportOptions, 112
- esoShowRecordNo
 - TwwDBGrid
 - ExportOptions, 112
- esoShowTitle
 - TwwDBGrid
 - ExportOptions, 112
- esoTransparentGrid
 - TwwDBGrid
 - ExportOptions, 113
- Execute method
 - TwwDBRichEdit, 192
 - TwwFilterDialog, 219
 - TwwLocateDialog, 234
 - TwwLookupDialog, 240
 - TwwMemoDialog, 246
 - TwwRecordViewDialog, 271
 - TwwSearchDialog, 288
- ExecuteDialog method
 - TwwFilterDialog, 220
- ExecuteFindDialog method
 - TwwDBRichEdit, 192
- ExecuteFontDialog method
 - TwwDBRichEdit, 192
- ExecuteParagraphDialog method
 - TwwDBRichEdit, 192
- ExecuteQuery,
 - TwwFilterDialog, 220
- ExecuteReplaceDialog method
 - TwwDBRichEdit, 192
- ExecuteTabDialog method
 - TwwDBRichEdit, 192
- Expanded property
 - TwwExpandButton, 202
 - TwwInspectorItem, 75
- ExpandedOnly parameter
 - TwwInspectorItem, 77
- ExpandGlyph,
 - TwwDataInspector, 54
- Export method
 - TwwDBRichEdit, 193
- Export To Clipboard, 113
- Exporting to Excel (SLK), 113
- Exporting to HTML File, 113
- ExportOptions property
 - TwwDBGrid, 111
- ExportType property
 - TwwDBGrid
 - ExportOptions, 111

F

- FastRecordScrolling property
 - TwwDataInspector, 60
- fdBitmap parameter,
 - TwwDBGrid, 140
- fdCheckBox parameter,
 - TwwDBGrid, 140
- fdCustom parameter,
 - TwwDBGrid, 140
- fdField parameter,
 - TwwDBGrid, 140
- fdImageIndex parameter,
 - TwwDBGrid, 140
- fdRichEdit parameter,
 - TwwDBGrid, 141
- fdByFilter, TwwFilterDialog, 210
- fdByQueryModify,
 - TwwFilterDialog, 210
- fdCaseSensitive,
 - TwwFilterDialog, 212
- fdClearWhenCloseDataSet,
 - TwwFilterDialog, 212
- fdClearWhenNoCriteria,
 - TwwFilterDialog, 212
- fdDisableDateTimePicker,
 - TwwFilterDialog, 213
- fdFilterByRange,
 - TwwFilterDialog, 208
- fdFilterByValue,
 - TwwFilterDialog, 208
- fdHidePartialAnywhere,
 - TwwFilterDialog, 213
- fdNone, TwwFilterDialog, 210
- fdShowCaseSensitive,
 - TwwFilterDialog, 212
- fdShowFieldOrder,
 - TwwFilterDialog, 212
- fdShowNonMatching,
 - TwwFilterDialog, 213
- fdShowOKCancel,
 - TwwFilterDialog, 212
- fdShowValueRangeTab,
 - TwwFilterDialog, 213
- fdShowViewSummary,
 - TwwFilterDialog, 212
- fdSizeable, TwwFilterDialog, 213
- fdSmartFilter,
 - TwwFilterDialog, 209
- fdUseActiveIndex,
 - TwwFilterDialog, 211
- fdUseAllIndexes,
 - TwwFilterDialog, 210
- Field Order, TwwFilterDialog, 206
- Field property
 - TwwInspectorItem, 75
- FieldName method
 - TwwDBGrid, 137
- FieldOperators property
 - TwwFilterDialog, 209
- fields
 - Display Attributes, 28
 - InfoPower Edit Control, 30
- Fields
 - TwwFilterDialog, 206
- FieldSelection property
 - TwwLocateDialog, 232
- FieldsFetchMethod property
 - TwwFilterDialog, 209
- FieldValue property
 - TwwLocateDialog, 232
- FileName
 - TwwHistoryList, 83
- FileName property
 - TwwDBGrid
 - ExportOptions, 112
 - TwwDBGrid IniAttributes, 116
- FilterActivate method
 - TwwTable, 295
- FilterCount property
 - TwwTable, 291
- FilterDialog. *See* TwwFilterDialog
- FilterDialog property
 - TwwIntl, 226
- FilterMemoSize property
 - TwwIntl, 226
- FilterMethod property
 - TwwFilterDialog, 210
- FilterOptimization property
 - TwwFilterDialog, 210
- FilterPropertyOptions
 - TwwFilterDialog, 211
- FilterPropertyOptions property
 - ADO suggested settings, 212
 - Interbase suggested settings, 212

- Filters
 - TwwTable, 296
- Filters with wildcards
 - TwwTable, 296
- FindFirst method
 - TwwLocateDialog, 235
- FindNext method
 - TwwLocateDialog, 235
- FindNextMatch method
 - TwwDBRichEdit, 192
- FindReplace method
 - TwwDBRichEdit, 193
- FindReplaceText method
 - TwwDBRichEdit, 193
- FindValue method
 - TwwIncrementalSearch, 224
- FirstDayOfWeek property
 - TwwDBDateTimePicker
 - CalendarAttributes, 94
 - TwwDBMonthCalendar, 170
- fixed columns
 - TwwDBGrid, 147
- FixedColor property
 - TwwDBGrid, 125
- FixedCols property
 - TwwDBGrid, 114
- Flat property, 25
 - TwwDBNavigator, 175
 - TwwNavButton, 174
- FlushChanges method
 - TwwDBGrid, 137
- fmUseSQL, TwwFilterDialog, 210
- fmUseTFields,
 - TwwFilterDialog, 210
- fmUseTTable,
 - TwwFilterDialog, 210
- FocusBorders property
 - EditFrame, 23
 - Frame, 23
- FocusStyle property
 - EditFrame, 23
 - Frame, 23
- Font property
 - TwwDBDateTimePicker
 - CalendarAttributes, 94
 - TwwMemoDialog, 244
 - TwwRecordViewDialog, 263
 - TwwRecordViewPanel, 277
- Font style in dialogs, 226
- FooterCellColor property
 - TwwDBGrid, 114
- FooterColor property
 - TwwDBGrid, 114
- FooterHeight property
 - TwwDBGrid, 114
- footers
 - within a grid, 120, 136, 141
- FormPosition property
 - TwwRecordViewDialog, 264
- frame controller, 50
- frame effects
 - key properties and events, 22
 - recordviewdialog, 24
 - recordviewpanel, 24
 - supporting components, 22
- Frame properties, 22, 153
- Frame property
 - TwwCheckBox, 45
 - TwwController, 50
 - TwwDBComboBox, 82
 - TwwDBComboDlg, 91
 - TwwDBDateTimePicker, 96
 - TwwDBEdit, 99
 - TwwDBLookupCombo, 153
 - TwwDBLookupComboDlg, 162
 - TwwRadioButton, 255
 - TwwRadioGroup, 258
- G**
- GetActiveField method
 - TwwDBGrid, 137
- GetComboDisplay method
 - TwwDBComboBox, 86
- GetComboValue method
 - TwwDBComboBox, 86
- GetFirstChild method
 - TwwDataInspector, 68
 - TwwInspectorItem, 77
- GetItemByCaption method
 - TwwDataInspector, 68
- GetItemByFieldname method
 - TwwDataInspector, 68
- GetItemByRow method
 - TwwDataInspector, 68
- GetItemByTagString method
 - TwwDataInspector, 68
- GetLastChild method
 - TwwInspectorItem, 78
- GetNext method
 - TwwInspectorItem, 78
- GetNextRecordText method
 - TwwDBGrid, 138
- GetNextSibling method
 - TwwInspectorItem, 78
- GetPrior method
 - TwwInspectorItem, 78
- GetPriorRecordText method
 - TwwDBGrid, 137
- GetPriorSibling method
 - TwwInspectorItem, 78
- GetRowByItem method
 - TwwDataInspector, 68
- GetRTFText method
 - TwwDBRichEdit, 193
- GlyphImages property
 - TwwRadioGroup, 258
- GPFs, 299
- graphics in grids, 148
- grid column headings. *See* TitleAlignment and TitleColor properties
- Grid property
 - TwwDBLookupCombo, 153
 - TwwExpandButton, 202
- Grid. *See* TwwDBGrid, 101
- GridColor property
 - TwwDBLookupComboDlg, 162
 - TwwLookupDialog, 236
 - TwwSearchDialog, 283
- GridIndents property
 - TwwExpandButton, 202
- GridOptions property
 - TwwDBLookupComboDlg, 163
 - TwwLookupDialog, 237
 - TwwSearchDialog, 284
- GridTitleAlignment property
 - TwwDBLookupComboDlg, 163
 - TwwLookupDialog, 237
 - TwwSearchDialog, 284
- GroupFieldName property
 - TwwDBGrid, 114
- Grouping
 - TwwDBGrid, 101
- GutterWidth property
 - TwwDBRichEdit, 184
- H**
- HaveVisibleItem method
 - TwwDataInspector, 68
- Height property
 - EditorPosition in
 - TwwDBRichEdit, 183
 - TwwRecordViewDialog, 264
- Help, 20
 - How-To & Tips Sections, 20
 - Implementation & Coding Examples, 20
 - On-line help, 20
 - Troubleshooting Section, 20
- HideAllLines property

- TwwDBGrid, 114
- HighlightColor property
 - TwwDBRichEdit, 184
- HistoryList property
 - TwwDBComboBox, 82
- HorizontalView property
 - TwwRecordViewDialog, 263
 - TwwRecordViewPanel, 276
- hot-tracking
 - TwwDBComboBox, 47
- HTMLBorderWidth property
 - TwwDBGrid
 - ExportOptions, 112

I

- iioAutoDateTimePicker,
 - TwwInspectorItem, 75
- iioAutoLookupCombo,
 - TwwInspectorItem, 75
- ImageIndex property
 - TwwNavButton, 174
- ImageList property
 - TwwDBGrid, 115
 - TwwDBLookupCombo, 153
 - TwwDBNavigator, 175
- Images in a grid, 128
- images in grids, 148
- Images property
 - TwwCheckBox, 45
 - TwwExpandButton, 202
 - TwwRadioButton, 255
 - TwwRadioGroup, 258
- Import method
 - TwwDBRichEdit, 193
- Increment property
 - TwwDBSpinEdit, 197
- incremental search. *See* TwwIncrementalSearch
- Indents property
 - TwwCheckBox, 46
 - TwwExpandButton, 202
 - TwwRadioButton, 255
 - TwwRadioGroup, 258
- Index property
 - TwwNavButton, 174
- IndicatorButton property
 - TwwDBGrid, 115
- IndicatorColor property
 - TwwDBGrid, 115
- IndicatorIconColor property
 - TwwDBGrid, 115
- IndicatorRow property
 - TwwDataInspector, 56
- InfoPower
 - building packages in Delphi 5, 14

- component hierarchy, 15
- component overview, 15
- components. *See* components
- custom framing, 21
- database architecture, 21
- demonstration/sample project, 15
- description, 5
- distribution requirements, 12
- installing, 7
- on-line help. *See* on-line help
- picture masks. *See* picture masks
- Picture Masks, 32
- programming, 19
- reference description, 43
- source code, 20
- transparency, 21
- uninstalling, 12
- whats new, 21
- InfoPower 4000
 - compatibility with InfoPower 2000, 12
 - introduction, 3
 - license agreement, 2
 - technical support, 3
- IniAttributes property
 - TwwDBGrid, 115
- IniFileName property
 - TwwIntl, 226
- InitCombo method
 - TwwKeyCombo, 230
- InitialDelay property
 - RepeatInterval of TwwDBNavigator, 176
- InplaceEditor property
 - TwwDataInspector, 56
 - TwwDBGrid, 117
- Insert method
 - TwwInspectorCollection, 72
- Insert Object Dialog
 - for a TwwDBRichEdit, 187
 - TwwDBRichEdit, 183
- installation, 7
 - requirements, 7
 - step-by-step, 8
- Installation
 - On-line Help
 - Delphi, 10
 - Tip, 10
 - international language support. *See* TwwIntl
 - Internet addresses in a rich edit, 181, 191

- Interval property
 - Interval of TwwDBNavigator, 176
- Intl. *See* TwwIntl
- InvalidateCurrentRow method
 - TwwDBGrid, 138
- InvalidateRow method
 - TwwDataInspector, 68
- IsSelected method
 - TwwDBGrid, 138
- IsVisible method
 - TwwNavButton, 177
- ItemHeight property
 - TwwDBComboBox, 83
- ItemIndex property
 - TwwRadioGroup, 258
- Items property
 - TwwDataInspector, 56
 - TwwDBComboBox, 83, 88
 - TwwInspectorItem, 75
 - TwwInspectorItem.PickList, 76
 - TwwNavButtons, 174
 - TwwRadioGroup, 259

K

- KeyOptions property
 - TwwDBGrid, 118

L

- LabelFont property
 - TwwRecordViewDialog, 264
 - TwwRecordViewPanel, 277
- LabelIndent property
 - TwwRecordViewDialog, 263
 - TwwRecordViewPanel, 276
- Layout property
 - TwwDBNavigator, 175
- ldoCaseSensitiveBelow,
 - TwwLocateDialog, 233
- ldoCloseOnMatch,
 - TwwLocateDialog, 233
- Left margin
 - TwwDBRichEdit, 188
- Left property
 - EditorPosition in TwwDBRichEdit, 183
 - TwwRecordViewDialog, 264
- LeftOffset property
 - TwwRecordViewDialog, 264
 - TwwRecordViewPanel, 277
- Level property
 - TwwInspectorItem, 75

License Agreement, 2

LikeSupportsUpperKeyword property, TwwFilterDialog, 211

LikeWildcardChar property, TwwFilterDialog, 211

LimitEditRect property
TwwDBComboBox, 84
TwwDBComboDlg, 91

LineBreak property
TwwNavButton, 174

LineColors property
TwwDBGrid, 118

Lines property
TwwDBRichEdit, 184
TwwMemoDialog, 244

LinesPerMemoControl property
TwwRecordViewDialog, 264
TwwRecordViewPanel, 277

LineStyle property
TwwDBGrid, 118

LineStyleCaption property
TwwDataInspector, 57

LineStyleData property
TwwDataInspector, 57

LoadAllRTF property
TwwDBGrid, 118

LoadFromFile method
TwwInspectorCollection, 71

LoadFromIniFile method
TwwDBGrid, 138

LoadFromStream method
TwwInspectorCollection, 71

LocateDialog property
TwwIntl, 226

lookup combo
within a grid, 145

lookup table
TwwDBLookupComboDlg, 161
TwwLookupDialog, 236

LookupCombos in a grid, 300

LookupDisplay property
TwwDBLookupCombo, 156

lookupfield
within a grid, 145

LookupField property
TwwDBLookupCombo, 153
TwwDBLookupComboDlg, 163

LookupFields property
TwwQBE, 249, 250
TwwStoredProc, 289

LookupLinks property
TwwStoredProc, 289

LookupSource property, 156

LookupTable property
TwwDBLookupCombo, 154
TwwDBLookupComboDlg, 163
TwwLookupDialog, 237

LookupValue property
TwwDBLookupCombo, 154
TwwDBLookupComboDlg, 163

M

MapList property
TwwDBComboBox, 84
TwwInspectorItem.PickList, 76

Mapped values
TwwDBComboBox, 88

Mapping a combo-box list
TwwDBComboBox, 84

Margin property
TwwNavButton, 174
TwwRecordViewDialog, 264
TwwRecordViewPanel, 277

masks. *See* picture masks

master/detail grids, 143, 200

MatchType property
TwwLocateDialog, 233

MaxDate property
TwwDBDateTimePicker, 96
TwwDBMonthCalendar, 170

MaxHeight property
TwwDBLookupComboDlg, 163
TwwLookupDialog, 237
TwwSearchDialog, 284

MaxSelectCount property
TwwDBMonthCalendar, 170

MaxSize
TwwHistoryList, 83

MaxValue property
TwwDBSpinEdit, 198

MaxWidth property
TwwDBLookupComboDlg, 163
TwwLookupDialog, 237
TwwSearchDialog, 284

mDisableDialog property
TwwDBGrid, 119
TwwMemoDialog, 245

mdoDayState property
TwwDBMonthCalendar, 170

mdoMultiSelect property
TwwDBMonthCalendar, 170

mdoNoToday property
TwwDBMonthCalendar, 170

mdoNoTodayCircle property
TwwDBMonthCalendar, 170

mdoWeekNumbers property
TwwDBMonthCalendar, 170

MeasurementUnits property
TwwDBRichEdit, 184

memo dialog, 243

memo fields
TwwDBGrid, 118, 147
view/edit in
TwwMemoDialog, 243
view/edit options, 245

MemoAttributes property
TwwDBGrid, 118
TwwMemoDialog, 245

Menu property
TwwRecordViewDialog, 265

mGridShow property
TwwDBGrid, 118
TwwMemoDialog, 245

MinDate property
TwwDBDateTimePicker, 96
TwwDBMonthCalendar, 170

MinValue property
TwwDBSpinEdit, 198

modify labels and hints, 228

MouseCoord method
TwwDBGrid, 138

MouseEnterSameAsFocus property
Frame, 23

MouseToCell method
TwwDataInspector, 68

MouseToItem method
TwwDataInspector, 69

MoveBy property
TwwDBNavigator, 175

Moving to a new cell in a grid, 129

MRUEnabled
TwwHistoryList, 83

MRUMaxSize
TwwHistoryList, 83

mSizable property
TwwDBGrid, 118

- TwWMemoDialog, 245
 - msoAutoUnselect,
 - TwwDBGrid, 119
 - msoShiftSelect, TwwDBGrid, 119
 - MSWordSpellChecker method
 - TwwDBRichEdit, 194
 - multiple row record display, 144
 - multiselect
 - with a
 - TwwDBLookupDialog, 241
 - Multi-Select
 - TwwDBMonthCalendar
 - TwwDBMonthCalendar, 170
 - Multi-selection in a grid
 - enabling, 146
 - iterating through the selected records, 146
 - MultiSelectOptions property
 - TwwDBGrid, 119
 - mViewOnly property
 - TwwDBGrid, 119
 - TwwMemoDialog, 245
 - mWordWrap property
 - TwwDBGrid, 118
 - TwwMemoDialog, 245
- N**
- NarrowSearch property
 - TwwTable, 292
 - NarrowSearchUpperChar property
 - TwwTable, 292
 - NavButtons property
 - TwwNavButton, 175
 - Navigator property
 - TwwDBLookupCombo, 154
 - TwwIntl, 226
 - TwwNavButton, 175
 - TwwNavButtons, 173
 - TwwRecordViewDialog, 265
 - Navigator, database, 173
 - NavigatorButtons property
 - TwwRecordViewDialog, 265
 - NavigatorFlat property
 - TwwRecordViewDialog, 265
 - New Search Button,
 - TwwFilterDialog, 206
 - noConfirmDelete property
 - TwwDBNavigator, 175
 - NonFocusBorders property
 - EditFrame, 23
 - Frame, 23
 - NonFocusColor property
 - EditFrame, 23
 - Frame, 23
 - NonFocusFontColor property
 - EditFrame, 23
 - Frame, 23
 - NonFocusStyle property
 - EditFrame, 23
 - Frame, 23
 - NonFocusTextOffsetX property
 - EditFrame, 23
 - Frame, 23
 - NonFocusTextOffsetY property
 - EditFrame, 23
 - Frame, 23
 - NonFocusTransparentFontCol or property
 - EditFrame, 23
 - Frame, 23
 - noUseInternationalText property
 - TwwDBNavigator, 175
 - NullAndBlankState property
 - TwwCheckBox, 46
 - NullChar property
 - TwwFilterDialog, 209
 - NumberColumns property
 - PopupYearOptions, TwwDBMonthCalendar, 171
 - NumGlyphs property
 - TwwNavButton, 174
- O**
- ofoCancelOnEscape property
 - TwwTable, 292
 - ofoEnabled property
 - TwwTable, 292
 - ofoShowHourGlass property
 - TwwTable, 292
 - OK buttons
 - Setting the caption, 225
 - OKCancelBitmapped property
 - TwwIntl, 227
 - OKCancelOptions property
 - TwwRecordViewDialog, 265
 - OLE
 - TwwDBRichEdit, 183, 184
 - OLE Dialog
 - TwwDBRichEdit, 187
 - OnAcceptFilterRecord event
 - TwwFilterDialog, 216
 - OnAddHistoryItem event
 - TwwDBComboBox, 85
 - OnAfterCollapse event
 - TwwExpandButton, 203
 - OnAfterCreateControl event
 - TwwRecordViewDialog, 269
 - TwwRecordViewPanel, 279
 - OnAfterCreateDialog event
 - TwwNavButton, 176
 - OnAfterDrawCell event
 - TwwDBGrid, 125
 - OnAfterExpand event
 - TwwExpandButton, 203
 - OnAfterSearch event
 - TwwIncrementalSearch, 224
 - OnAfterSelectCell event
 - TwwDataInspector, 61
 - OnBeforeCollapse event
 - TwwExpandButton, 203
 - OnBeforeCreateControl event
 - TwwRecordViewDialog, 269
 - TwwRecordViewPanel, 280
 - OnBeforeDrawCell event
 - TwwDBGrid, 125
 - OnBeforeExpand event
 - TwwExpandButton, 203
 - OnBeforePaint event
 - background image, 61
 - Example, 61
 - grid background image, 126
 - Grid Example, 126
 - TwwDataInspector, 61
 - TwwDBGrid, 126
 - OnBeforeSelectCell event
 - TwwDataInspector, 62
 - OnCalcBoldDay event
 - TwwDBDateTimePicker, 97
 - TwwDBMonthCalendar, 171
 - OnCalcCellColors event
 - TwwDBGrid, 127
 - OnCalcDataPaintText event
 - Example, 62
 - summary text of child items, 62
 - TwwDataInspector, 62
 - OnCalcTitleAttributes event
 - TwwDBGrid, 127
 - OnCalcTitleImage event
 - TwwDBGrid, 128
 - OnCancelWarning event
 - TwwRecordViewDialog, 270
 - OnCanCollapse event
 - TwwDataInspector, 62
 - OnCanExpand event

- TwwwDataInspector, 63
- OnCellChanged event
 - TwwwDBGrid, 129
- OnCheckValue event, 36
 - TwwwDBEdit, 100
 - TwwwDBGrid, 129
- OnCloseDialog
 - TwwwDBLookupComboDlg, 165
 - TwwwMemoDialog, 246
- OnCloseDialog event
 - TwwwDBRichEdit, 189
 - TwwwLookupDialog, 240
 - TwwwRecordViewDialog, 270
 - TwwwSearchDialog, 287
- OnCloseUp event
 - TwwwDBComboBox, 85
 - TwwwDBLookupCombo, 157
- OnCollapsed event
 - TwwwDataInspector, 63
- OnColumnMoved event
 - TwwwDBGrid, 130
- OnColWidthChanged event
 - TwwwDBGrid, 130
- OnCreateDateTimePicker event
 - TwwwDataInspector, 63
 - TwwwDBGrid, 130
- OnCreateDefaultCombo event
 - TwwwDataInspector, 63
- OnCreateDialog event
 - TwwwDBRichEdit, 189
- OnCreateHintWindow event
 - TwwwDataInspector, 64
 - TwwwDBGrid, 130
- OnCreateRadioButton event
 - TwwwRadioGroup, 260
- OnCustomDlg event
 - TwwwDBComboDlg, 92
- OnDialogSummary event
 - TwwwFilterDialog, 216
- OnDitto event
 - TwwwDBGrid, 131
- OnDrawCaptionCell event
 - Example, 65
 - TwwwDataInspector, 64
- OnDrawDataCell event
 - Example, 65, 66
 - TwwwDataInspector, 65, 69
 - TwwwDBGrid, 131
- OnDrawFooterCell event
 - TwwwDBGrid, 132
- OnDrawGroupHeaderCell event
 - TwwwDBGrid, 132
- OnDrawIndicatorCell event
 - Example, 66
- TwwwDataInspector, 66
- OnDrawItem event
 - TwwwDBComboBox, 85
- OnDrawTitleCell event
 - TwwwDBGrid, 132
- OnDropDown event
 - TwwwDBComboBox, 85
 - TwwwDBLookupCombo, 157
- OnEditButtonClick event
 - TwwwInspectorItem, 77
- OnEncodeDateTime event
 - TwwwFilterDialog, 217
- OnEncodeValue event
 - TwwwFilterDialog, 217
- OnExecuteSQL event
 - TwwwFilterDialog, 217
- OnExpanded event
 - TwwwDataInspector, 66
- OnExportField event
 - TwwwDBGrid, 133
- OnFieldChanged event
 - TwwwDBGrid, 133
- OnFilter event
 - TwwwQBE, 250
 - TwwwQuery, 254
 - TwwwStoredProc, 290
 - TwwwTable, 294
- OnFilter events
 - TwwwClientDataset, 49
- OnFilterEscape event
 - TwwwQBE, 251
 - TwwwQuery, 254
 - TwwwStoredProc, 290
 - TwwwTable, 295
- OnFilterOptions event
 - TwwwQBE, 250
- OnFilterOptions property
 - TwwwQuery, 253
 - TwwwStoredProc, 289
 - TwwwTable, 292
- OnFilterPropertyOptions property
 - TwwwFilterDialog, 212
- OnInitDialog event
 - TwwwDBLookupComboDlg, 165
 - TwwwDBRichEdit, 189
 - TwwwFilterDialog, 217
 - TwwwLocateDialog, 234
 - TwwwLookupDialog, 239
 - TwwwMemoDialog, 246
 - TwwwRecordViewDialog, 270
 - TwwwSearchDialog, 286
- OnInitTempDataSet event
 - TwwwFilterDialog, 217
- OnInvalidValue event, 37
 - TwwwClientDataSet, 49
- TwwwQBE, 251
- TwwwQuery, 254
- TwwwStoredProc, 290
- TwwwTable, 295
- OnItemChanged event
 - TwwwDataInspector, 66
 - TwwwInspectorItem, 77
- OnLeftColChanged event
 - TwwwDBGrid, 133
- On-line help, 20
- OnMemoClose event
 - TwwwDBGrid, 133
- OnMemoOpen event
 - TwwwDBGrid, 133
- OnMenuLoadClick event
 - TwwwDBRichEdit, 189
- OnMenuPrintClick event
 - TwwwDBRichEdit, 190
- OnMenuSaveAndExitClick event
 - TwwwDBRichEdit, 190
- OnMenuSaveAsClick event
 - TwwwDBRichEdit, 190
- OnMouseDown event
 - TwwwDBGrid, 134
 - TwwwDBMonthCalendar, 172
- OnMouseEnter event
 - TwwwCheckBox, 47
 - TwwwDBEdit, 100
- OnMouseLeave event
 - TwwwCheckBox, 47
 - TwwwDBEdit, 100
- OnMouseMove event
 - TwwwDBGrid, 135
 - TwwwDBMonthCalendar, 172
- OnMouseUp event
 - TwwwDBGrid, 135
 - TwwwDBMonthCalendar, 172
- OnMultiSelectRecord event
 - TwwwDBGrid, 135
- OnNotInList event
 - TwwwDBLookupCombo, 157
- OnPerformCustomSearch event
 - TwwwDBLookupCombo, 158
 - TwwwDBLookupComboDlg, 165
 - TwwwIncrementalSearch, 227
 - TwwwLookupDialog, 240
 - TwwwSearchDialog, 287
- , 190
- OnPrintHeader event

- TwwDBRichEdit, 190
 - OnResize Event
 - TwwDBNavigator, 176
 - OnResizeDialog event
 - TwwRecordViewDialog, 270
 - OnRowChanged event
 - TwwDBGrid, 135
 - TwwNavButton, 176
 - OnSelectField event
 - TwwFilterDialog, 218
 - OnSetControlEffects event, 24
 - TwwRecordViewDialog, 271
 - TwwRecordViewPanel, 280
 - OnSortChange event
 - TwwLookupDialog, 240
 - TwwSearchDialog, 287
 - OnSyncDataSets event
 - TwwSearchDialog, 287
 - OnTitleButtonClick event
 - TwwDBGrid, 135
 - OnTopLeftChanged event
 - TwwDataInspector, 67
 - OnTopRowChanged event
 - TwwDBGrid, 135
 - OnUpdateFooter event
 - TwwDBGrid, 136
 - OnUpdateState event
 - TwwNavButton, 176
 - OnURLOpen event
 - TwwDBGrid, 136
 - TwwDBRichEdit, 181
 - OnURLOpen,
 - TwwDBRichEdit, 191
 - OnUserButton1Click
 - TwwDBLookupComboDlg, 166
 - TwwLookupDialog, 240
 - TwwMemoDialog, 246
 - TwwSearchDialog, 287
 - OnUserButton2Click
 - TwwDBLookupComboDlg, 166
 - TwwLookupDialog, 240
 - TwwMemoDialog, 246
 - TwwSearchDialog, 288
 - OnValidationErrorUsingMask event
 - custom validation error message, 67
 - Example, 67
 - TwwDataInspector, 67
 - TwwIntl, 227
 - opFixFirstColumn property
 - TwwDBLookupComboDlg, 164
 - TwwLookupDialog, 238
 - TwwSearchDialog, 238
 - opGroupControls property
 - TwwDBLookupComboDlg, 163
 - TwwLookupDialog, 238
 - TwwSearchDialog, 238
 - opShowOKCancel property
 - TwwDBLookupComboDlg, 163
 - TwwLookupDialog, 237
 - TwwSearchDialog, 237
 - opShowSearchBy property
 - TwwDBLookupComboDlg, 163
 - TwwLookupDialog, 237
 - TwwSearchDialog, 237
 - opShowStatusBar property
 - TwwDBLookupComboDlg, 164
 - TwwLookupDialog, 238
 - TwwSearchDialog, 238
 - Optimization in
 - TwwFilterDialog, 210
 - Options property
 - TwwDataInspector, 57
 - TwwDBDateTimePicker CalendarAttributes, 94
 - TwwDBGrid, 119
 - TwwDBGrid ExportOptions, 112
 - TwwDBGrid.DittoAttribute s, 110
 - TwwDBLookupComboDlg, 163
 - TwwDBMonthCalendar, 170
 - TwwDBNavigator, 175
 - TwwFilterDialog, 212, 213
 - TwwInspectorItem, 75
 - TwwLocateDialog, 233
 - TwwLookupDialog, 237
 - TwwRecordViewDialog, 266
 - TwwRecordViewPanel, 277
 - TwwSearchDialog, 237, 284
 - OrChar property
 - TwwFilterDialog, 209
 - Order of fields
 - in a grid or record-view, 28
 - OrderByDisplay property
 - TwwDBLookupCombo, 154
 - TwwDBLookupComboDlg, 164
 - OutputWidthInTwips
 - TwwDBGrid ExportOptions, 113
 - ovActiveRecord3DLines,
 - TwwDataInspector, 58
 - ovAllowInsert,
 - TwwDataInspector, 58
 - ovColumnResize,
 - TwwDataInspector, 57
 - ovEnterToTab,
 - TwwDataInspector, 57
 - ovFillNonCellArea,
 - TwwDataInspector, 58
 - ovHideCaptionColumn,
 - TwwDataInspector, 58
 - ovHideVertDataLines,
 - TwwDataInspector, 57
 - ovHideVertFixedLines,
 - TwwDataInspector, 58
 - ovHighlightActiveRow,
 - TwwDataInspector, 57
 - ovRowResize,
 - TwwDataInspector, 57
 - ovShowCaptionHints,
 - TwwDataInspector, 58
 - ovShowCellHints,
 - TwwDataInspector, 58
 - ovShowTreeLines,
 - TwwDataInspector, 58
 - ovTabExits,
 - TwwDataInspector, 57
 - ovTabToVisibleOnly,
 - TwwDataInspector, 58
- P**
- Pack method
 - TwwTable, 295
 - PadColumnStyle property
 - TwwDBGrid, 121
 - PaintOptions property
 - TwwDataInspector, 58
 - TwwDBGrid, 121
 - paperless forms, 21
 - ParentItem property
 - TwwInspectorItem, 75
 - PerformSearch method
 - TwwDBLookupCombo, 159
 - PickList property
 - TwwInspectorItem, 76
 - picture masks, 32
 - creating picture masks, 33
 - defining a property string, 33
 - dialogs
 - Design Picture Mask dialog, 39
 - Select Picture Mask dialog, 38
 - editing, 35
 - events, 35
 - examples, 34
 - properties, 35

- removing, 40
 - special characters, 33
 - supporting components, 22, 32
 - usage, 32
 - Picture property
 - AllowInvalidExit, 35
 - AutoFill, 35
 - PictureMask, 35
 - TwwDBEdit, 99
 - TwwDBLookupCombo, 154
 - TwwInspectorItem, 76
 - PictureMask property
 - TwwIncrementalSearch, 223
 - PictureMaskAutoFill property
 - TwwIncrementalSearch, 223
 - PictureMaskFromDataset property
 - TwwLookupDialog, 238
 - PictureMaskFromDataSet property
 - TwwDataInspector, 60
 - TwwDBGrid, 122
 - TwwRecordViewDialog, 267
 - TwwRecordViewPanel, 278
 - TwwSearchDialog, 284
 - PictureMaskFromField property
 - TwwIncrementalSearch, 223
 - TwwLookupDialog, 238
 - TwwSearchDialog, 284
 - PictureMasks property, 35
 - TwwClientDataSet, 48
 - TwwDBGrid, 122
 - TwwLookupDialog, 238
 - TwwQBE, 250
 - TwwQuery, 253
 - TwwRecordViewDialog, 267
 - TwwRecordViewPanel, 278
 - TwwSearchDialog, 285
 - TwwStoredProc, 289
 - TwwTable, 293
 - PopupMenu property
 - TwwDBRichEdit, 186
 - PopupOptions property
 - TwwDBRichEdit, 186
 - PopupYearOptions property
 - TwwDBDateTimePickerCalendarAttributes, 94
 - TwwDBMonthCalendar, 171
 - PrimaryKeyName property
 - TwwKeyCombo, 230
 - Print method
 - TwwDBRichEdit, 194
 - PrintFooter property
 - TwwDBRichEdit, 185
 - PrintHeader property
 - TwwDBRichEdit, 185
 - PrintJobName,
 - TwwDBRichEdit, 188
 - PrintMargins property
 - TwwDBRichEdit, 188
 - PrintPageSize property
 - TwwDBRichEdit, 185
 - programming InfoPower, 19
- Q**
- QBE. *See* TwwQBE
 - QBE property
 - TwwQBE, 250
 - queries
 - QBE, 248
 - SQL, 253
 - Query By Example (QBE). *See* TwwQBE
 - Query property
 - TwwTable, 293
 - Quicken style search
 - TwwDBLookupCombo, 156
- R**
- ReadOnly property
 - TwwCheckBox, 46
 - TwwInspectorItem, 76
 - ReadOnlyColor property
 - TwwRecordViewDialog, 267
 - TwwRecordViewPanel, 278
 - Record view dialog, 261
 - Record view panel, 275
 - RefreshLinks method
 - TwwTable, 295
 - RegistrationNo property
 - TwwIntl, 227
 - removing picture masks, 40
 - reoAdjustPopupMenu property
 - TwwDBRichEdit, 184
 - reoCloseOnEscape property
 - TwwDBRichEdit, 183
 - reoDisableOLE property
 - TwwDBRichEdit, 185
 - reoFlatButtons property
 - TwwDBRichEdit, 183
 - reoNoConfirmation property
 - TwwDBRichEdit, 183
 - reoShowFormatBar property
 - TwwDBRichEdit, 182
 - reoShowHints property
 - TwwDBRichEdit, 183
 - reoShowInsertObject property
 - TwwDBRichEdit, 183
 - reoShowJustifyButton property
 - TwwDBRichEdit, 183
 - reoShowLoad property
 - TwwDBRichEdit, 182
 - reoShowMainMenuIcons property
 - TwwDBRichEdit, 183
 - reoShowPageSetup property
 - TwwDBRichEdit, 182
 - reoShowPrint property
 - TwwDBRichEdit, 182
 - reoShowRuler property
 - TwwDBRichEdit, 183
 - reoShowSaveAs property
 - TwwDBRichEdit, 182
 - reoShowSaveExit property
 - TwwDBRichEdit, 182
 - reoShowSpellCheck property
 - TwwDBRichEdit, 183
 - reoShowStatusBar property
 - TwwDBRichEdit, 183
 - reoShowToolBar property
 - TwwDBRichEdit, 182
 - reoUseBackColor property
 - TwwDBRichEdit, 183
 - RepeatInterval property
 - TwwDBNavigator, 176
 - requirements, 7
 - Resizable property
 - TwwInspectorItem, 76
 - ReturnWhereClause,
 - TwwFilterDialog, 220
 - Rich Edit Control, 179
 - RichEdit property
 - TwwIntl, 227
 - Right margin
 - TwwDBRichEdit, 188
 - RightOffset property
 - TwwRecordViewDialog, 264
 - TwwRecordViewPanel, 277
 - Rounding property
 - TwwFilterDialog, 213
 - RoundingMethod property,
 - TwwFilterDialog, 213
 - RowHeightPercent property
 - TwwDBGrid, 122
 - rpoPopopUnderline property
 - TwwDBRichEdit, 187
 - rpoPopupBold property
 - TwwDBRichEdit, 187
 - rpoPopupBullet property
 - TwwDBRichEdit, 187
 - rpoPopupCopy property
 - TwwDBRichEdit, 187
 - rpoPopupCut property
 - TwwDBRichEdit, 187

TwwDBRichEdit, 187
 rpoPopupEdit property
 TwwDBRichEdit, 187
 rpoPopupFind property
 TwwDBRichEdit, 187
 rpoPopupFont property
 TwwDBRichEdit, 187
 rpoPopupInsertObject property
 TwwDBRichEdit, 187
 rpoPopuPltalic property
 TwwDBRichEdit, 187
 rpoPopupMSWordSpellCheck
 property
 TwwDBRichEdit, 187
 rpoPopupParagraph property
 TwwDBRichEdit, 187
 rpoPopupPaste property
 TwwDBRichEdit, 187
 rpoPopupReplace property
 TwwDBRichEdit, 187
 rpoPopupTabs property
 TwwDBRichEdit, 187
 rvcTransparentButtons
 property
 TwwRecordViewDialog,
 262
 TwwRecordViewPanel, 276
 rvcTransparentLabels property
 TwwRecordViewDialog,
 262
 TwwRecordViewPanel, 276
 rvoClosesCancel property
 TwwRecordViewDialog,
 266
 rvoConfirmCancel property
 TwwRecordViewDialog,
 266
 rvoConsistentEditWidth
 property
 TwwRecordViewDialog,
 266
 TwwRecordViewPanel, 278
 rvoEnterToTab property
 TwwRecordViewDialog,
 266
 rvoHideCalculated property
 TwwRecordViewDialog,
 266
 TwwRecordViewPanel, 278
 rvoHideNavigator property
 TwwRecordViewDialog,
 266
 rvoHideReadOnly property
 TwwRecordViewDialog,
 266
 TwwRecordViewPanel, 277
 rvokAutoCancelRec property
 TwwRecordViewDialog,
 265
 rvokAutoPostRect property
 TwwRecordViewDialog,
 265
 rvokShowOKCancel property
 TwwRecordViewDialog,
 265
 rvoLabelsBeneathControl
 TwwRecordViewDialog,
 267
 rvoLabelsBeneathControl
 property
 TwwRecordViewDialog,
 278
 rvoMaximizeMemoWidth
 property
 TwwRecordViewDialog,
 267
 TwwRecordViewPanel, 278
 rvoModalForm property
 TwwRecordViewDialog,
 266
 rvoShortenEditBox property
 TwwRecordViewDialog,
 266
 TwwRecordViewPanel, 278
 rvoStayOnTopForm property
 TwwRecordViewDialog,
 266
 rvoUseCustomControls
 property
 TwwRecordViewDialog,
 266
 TwwRecordViewPanel, 278
 rvoUseDateTimePicker
 TwwRecordViewDialog,
 267
 rvoUseDateTimePicker
 property
 TwwRecordViewPanel, 278

S

SaveToFile method
 TwwInspectorCollection,
 72
 SaveToIniFile method
 TwwDBGrid, 139
 SaveToRegistry property
 TwwDBGrid IniAttributes,
 116
 SaveToStream method
 TwwInspectorCollection,
 72
 Saving Grid info to ini file, 116
 Saving Grid info to system
 registry, 116
 scroll back to prior records,
 299
 search
 against field, 231
 against index. *See*
 TwwSearchDialog. *See*
 TwwIncrementalSearch
 SearchDelay property
 TwwDBLookupCombo,
 155
 TwwIncrementalSearch,
 223
 SearchDialog property
 TwwIntl, 227
 SearchField property
 TwwDBLookupCombo,
 155
 TwwIncrementalSearch,
 223
 TwwLocateDialog, 233
 SearchTable property
 TwwSearchDialog, 285
 Section
 TwwHistoryList, 83
 SectionName property
 TwwDBGrid IniAttributes,
 116
 Select Fields dialog box, 25
 Adding Fields, 26
 Display Attributes, 28
 Edit Control, 30
 Select Fields Dialog Box
 Removing Fields, 28
 Select Picture Mask dialog, 38
 SelectAll method
 TwwDBGrid, 139
 Selected property
 TwwDBGrid, 122
 TwwDBLookupCombo,
 155
 TwwLookupDialog, 238
 TwwRecordViewDialog,
 267
 TwwRecordViewPanel, 279
 TwwSearchDialog, 285
 SelectedFields property
 TwwFilterDialog, 214
 SelectedList property
 TwwDBGrid, 123
 selecting table index. *See*
 TwwKeyCombo
 SelectRecord method
 TwwDBGrid, 139
 SeqSearchOption property
 TwwDBLookupComboDlg,
 164
 SeqSearchOptions property
 TwwDBLookupCombo,
 155
 Set # fixed columns, 147
 SetActiveField method
 TwwDBGrid, 139

- SetActiveRow method
 - TwwDBGrid, 139
- SetBold method
 - TwwDBRichEdit, 194
- SetBullet method
 - TwwDBRichEdit, 194
- SetControlType
 - fetBitmap, 140
 - fetCheckbox, 140
 - fetCustom, 140
 - fetField, 140
 - fetImageIndex, 140
 - fetRichEdit, 141
- SetControlType method
 - TwwDBGrid, 139
- SetDataSourceFromComponent method
 - TwwDBNavigator, 177
- SetFocusTabStyle property
 - TwwDataInspector, 60
- SetItalic method
 - TwwDBRichEdit, 194
- SetLookupField method
 - TwwQuery, 254
 - TwwTable, 296
- SetParam method
 - TwwQBE, 251
- SetPictureAutoFill method
 - TwwDBGrid, 141
- SetPictureMask method
 - TwwDBGrid, 141
- setting line colors
 - TwwDBGrid, 118
- SetUnderline method
 - TwwDBRichEdit, 194
- ShadowSearchTable property
 - TwwSearchDialog, 285
- ShortCutDittoField property
 - TwwDBGrid.DittoAttributes, 109
- ShortCutDittoRecord property
 - TwwDBGrid.DittoAttributes, 110
- ShowAllIndexes property
 - TwwKeyCombo, 230
- ShowAsButton property
 - TwwExpandButton, 203
- ShowBorder property
 - TwwRadioGroup, 259
- ShowButton property
 - TwwDBComboBox, 84
 - TwwDBComboDlg, 91
 - TwwDBDateTimePicker, 96
 - TwwDBLookupCombo, 156
 - TwwDBLookupComboDlg, 164
- ShowEditYear property
 - PopupYearOptions,
 - TwwDBMonthCalendar, 171
- ShowFocusRect property
 - TwwCheckBox, 46
 - TwwExpandButton, 203
 - TwwRadioButton, 256
 - TwwRadioGroup, 259
- ShowGroupCaption property
 - TwwRadioGroup, 259
- ShowHorzScrollBar property
 - TwwDBGrid, 123
- ShowMatchText property, 156
 - TwwDBComboBox, 84
 - TwwDBLookupComboDlg, 164
 - TwwIncrementalSearch, 223
 - TwwInspectorItem.PickList, 76
- ShowMessages property
 - TwwLocateDialog, 233
- ShowText property
 - TwwCheckBox, 46
 - TwwExpandButton, 203
 - TwwNavButton, 174
 - TwwRadioGroup, 259
- ShowVertScrollBar property
 - TwwDBEdit, 99
 - TwwDBGrid, 123
- SizeLastColumn method
 - TwwDBGrid, 141
- SortBy property
 - TwwFilterDialog, 214
- Sorted property
 - TwwDBComboBox, 84
- SortFields property
 - TwwLocateDialog, 233
- SortSelectedList method
 - TwwDBGrid, 141
- source code. *See* InfoPower source code
- Spacing property
 - TwwNavButton, 174
- SQL Tables, 299
 - Backward scrolling, 299
 - Navigating, 299
 - Performance, 299
 - TwwDBGrid, 300
- SQLPropertyName property
 - TwwFilterDialog, 215
- SQLTables property
 - TwwFilterDialog, 215
- SQLUpperString property
 - TwwFilterDialog, 215
- ssoCaseSensitive property,
 - TwwDBLookupCombo, 156
- ssoEnabled property,
 - TwwDBLookupCombo, 155
- StartDate property
 - TwwDBMonthCalendar, 171
- StartYear property
 - PopupYearOptions,
 - TwwDBMonthCalendar, 171
- State property
 - TwwCheckBox, 46
- StorageType
 - TwwHistoryList, 83
- stored procedures, 289
- Stored values
 - TwwDBComboBox, 88
- Style
 - TwwDBComboBox, 89
- Style property
 - TwwDBComboBox, 84
 - TwwDBComboDlg, 91
 - TwwInspectorItem.PickList, 76
 - TwwNavButton, 174
 - TwwRecordViewDialog, 268
 - TwwRecordViewPanel, 279
- SyncSQLByRange property
 - TwwTable, 293

T

- table lookups
 - TwwDBLookupCombo, 151
 - TwwDBLookupComboDlg, 161
 - TwwLookupDialog, 236
- TabStop property
 - TwwInspectorItem, 76
- Tag property
 - TwwInspectorItem, 76
 - TwwLocateDialog, 233
 - TwwLookupDialog, 238
 - TwwMemoDialog, 245
 - TwwSearchDialog, 285
- TagString property
 - TwwInspectorItem, 76
- Technical Support, 3
- Text property
 - TwwDBLookupCombo, 156
 - TwwIncrementalSearch, 224
- Themes, 41
- Time property

- TwwDBDateTimePicker, 96
- TwwDBMonthCalendar, 171
- TitleAlignment property
 - TwwDBGrid, 124
- TitleButtons property
 - TwwDBGrid, 124
- TitleColor property
 - TwwDBGrid, 124
- TitleImageList property
 - TwwDBGrid, 124
- TitleLines property
 - TwwDBGrid, 124
- TitleName
 - TwwDBGrid
 - ExportOptions, 113
- Top margin
 - TwwDBRichEdit, 188
- Top property
 - EditorPosition in
 - TwwDBRichEdit, 184
 - TwwRecordViewDialog, 264
- TopOffset property
 - TwwRecordViewDialog, 264
 - TwwRecordViewPanel, 277
- transparency
 - supporting components, 22
- Transparent property, 25
 - EditFrame, 24
 - Frame, 24
 - TwwDBNavigator, 176
 - TwwRadioGroup, 259
- TransparentActiveItem property
 - TwwRadioGroup, 259
- TreeLineColor property
 - TwwDataInspector, 61
- TwoColumnDisplay property
 - TwwDBComboBox, 85, 88
- TwwCheckBox, 5, 44
 - added events, 47
 - added properties, 44
 - description, 44
 - how to, 47
 - screenshot, 44
- TwwClientDataset, 48
 - added events, 49
 - added methods, 49
 - added properties, 48
- TwwClientDataSet, 5
- TwwController, 22, 50
 - added methods, 50
 - added properties, 50
- TwwCustomDrawGridCellInfo
 - TwwDBGrid, 125
- TwwDataInspector, 5, 51
- add a background, 69
- added events, 61
- added methods, 67
- added properties, 54
- adding dropdown items, 69
- adding rows at runtime, 71
- alternate row painting
 - sections, 59
- architecture, 52
- background image, 61
- changing expand/collapse glyphs, 69
- custom drawing in caption
 - cell, 65
- custom drawing in data cell, 65, 69
- custom drawing in indicator
 - cells, 66
- Customizing backgrounds
 - and colors, 58
- customizing rows, 53
- customizing validation error message, 67
- design-time
 - adding a new item, 53
 - adding a new sub-item, 53
 - adding fields, 54
 - deleting an item, 53
 - moving an item down, 54
 - moving an item up, 54
- design-time aid, 53
- displaying a checkbox, 70, 76
- embedding 3rd Party
 - controls, 69
- hide caption column, 69
- highlighting data cell
 - example, 65
- highlighting data font
 - example, 66
- how to, 69
- items, 53
- iterating through items, 69, 78
- manipulating items at
 - runtime, 71
- performance issues, 60
- row properties and methods, 73
- screenshot, 52
- selecting embedded
 - controls, 53, 74
- setting active row, 69
- summarizing text of child
 - items, 62
 - , 51
- TwwDataSource, 5, 79
- TwwDBComboBox, 5, 80
 - added events, 85
 - added methods, 86
 - added properties, 81
 - history lists, 82
 - how to, 87
 - persistent items, 82
 - two column display, 88
- TwwDBComboDlg, 5, 90
 - added events, 92
 - added properties, 90
- TwwDBDateTimePicker, 5, 93
 - added events, 97
 - added properties, 94
- TwwDBEdit, 5, 98
 - added events, 100
 - added methods, 100
 - added properties, 99
- TwwDBGrid, 5, 101
 - added events, 125
 - added methods, 136
 - added properties, 108
 - assigning to a new dataset, 300
 - Auto-resizing of column, 149
 - changing title attributes, 127
 - coloring alternating rows, 147
 - coloring an entire row, 127
 - Coloring the data cells, 127
 - column order, 28
 - deciding which fields to
 - export, 133
 - deleting multiple selected
 - records, 123
 - detect moving to different
 - record, 135
 - detecting a column drag, 130
 - detecting a column resize, 130
 - detecting a modified field, 133
 - detecting cell movement, 129
 - detecting horizontal
 - scrolling, 133
 - Detecting row change, 149
 - detecting user multi-select, 135
 - detecting vertical scrolling, 135
 - Displaying images in grids, 148
 - displaying images in the
 - title, 128

- displaying TGraphic fields, 148
- edit lookupfields in the grid, 145
- editing memos, 149
- Enter to Tab, 147
- expandable composite fields, 143
- How To, 141
- indicatorbutton, 145
- integrating the record-view, 146
- keyboard shortcuts
 - disabling, 147
- master/detail grids, 143
- multiple grids on one dataset, 149
- multirow record display, 144
- painting a data cell, 131
- painting a footer cell, 132
- Saving to ini file, 116
- Saving to system registry, 116
- setting options at runtime, 121
- Tips, 149
- TwwDBLookupCombo, 5, 151
 - added events, 157
 - added methods, 159
 - added properties, 152
 - default LookupTable index overriding, 160
 - defining attributes, 160
 - drop-down list
 - keyboard shortcut, 160
 - fill drop-down list from QBE, 160
 - Query, 160
 - how to, 159
 - multiple
 - TwwDBLookupCombos, 160
 - no data in drop-down lists, 300
 - selecting fields, 160
 - unable to scroll backwards, 300
 - update other fields, 159
- TwwDBLookupComboDlg, 5, 161
 - added events, 165
 - added properties, 162
 - wwidlg, 165
- TwwDBMonthCalendar, 5, 168
 - added properties, 169
 - displaying more than one month, 172
- events, 171
 - how to, 172
 - selecting a range of dates, 172
- TwwDBNavigator, 5, 173
 - added events, 176
 - added methods, 177
 - added properties, 173
 - how to, 177
- TwwDBRichEdit, 5, 179
 - added events, 189
 - added methods, 191
 - added properties, 181
 - adding own code to main menu, 195
 - appending text in popup editor, 196
 - binding to a database field, 195
 - displaying pop-up editor with button, 195
 - embedding richedit text in grid, 196
 - forcing wordwrap to adjust to printer, 196
 - how to, 195
 - storing RichText at design time, 195
- TwwDBSpinEdit, 5, 197
 - added properties, 197
- TwwExpandButton, 6, 200
 - added events, 203
 - added properties, 201
- TwwFilterDialog, 6, 205
 - added events, 216, 219
 - added properties, 208
 - end-user resizing, 221
 - how to, 204, 220
 - loading and saving filters, 221
 - unique display labels, 221
 - Using wwSearchDialog with wwFilterDialog, 220
- TwwIncrementalSearch, 6, 222
 - added events, 224
 - added methods, 224
 - added properties, 222
 - can't find value, 301
 - case-insensitive, 224
- TwwInspectorCollection, 71
 - added methods, 71
 - added properties, 71
 - adding rows at runtime, 71
- TwwInspectorItem, 73
 - added events, 77
 - added methods, 77
 - added properties, 73
- TwwIntl, 6, 225
 - added events, 227
 - added properties, 225
 - how to, 228
 - modify labels and hints, 228
- TwwKeyCombo, 6, 229
 - added methods, 230
 - added properties, 229
 - case insensitive, 230
 - display indexes, 301
 - drop-down list
 - keyboard shortcut, 230
 - properties
 - PrimaryKeyName, 230
- TwwLocateDialog, 6, 231
 - added events, 234
 - added methods, 234
 - added properties, 232
 - how to, 235
 - search field, 235
- TwwLookupDialog, 6, 236
 - access lookup tables, 242
 - added events, 239
 - added methods, 240
 - added properties, 236
 - how to, 240
 - lookup and fill, 240
 - multiselect use, 241
 - OK or Cancel, 302
- TwwMemoDialog, 6, 243
 - added events, 246
 - added methods, 246
 - added properties, 243
 - adding buttons, 247
 - adding timestamps to memos, 247
 - background color, 247
 - buttons, 247
 - how to, 247
 - OnInitDialog event, 247
 - position, 247
 - size, 247
 - using buttons, 247
- TwwNavButtons,
 - TwwDBNavigator, 173
- TwwQBE, 6, 248
 - added events, 250
 - added methods, 251
 - added properties, 248
 - heterogeneous QBEs, 252
 - how to, 251
 - multiple alias QBEs, 252
 - parameters, 252
- TwwQuery, 6, 253
 - added events, 254
 - added methods, 254
 - added properties, 253
 - how to, 254
- TwwRadioButton, 255
 - added properties, 255

- TwwRadioGroup, 6, 257
 - added events, 260
 - added properties, 258
 - description, 257
 - how to, 260
 - make radio group transparent, 260
 - screenshot, 257
 - setting items at runtime, 260
 - special features, 257
 - TwwRecordViewDialog, 6, 261
 - added events, 269
 - added methods, 271
 - added properties, 262
 - Attaching your own custom menu, 273
 - Controlling onClose behavior, 273
 - Creating accelerators for the controls, 272
 - Customize spacing between controls, 273
 - Customizing selection and order of fields, 271
 - Defining picture masks for fields, 274
 - Embedding custom controls, 272
 - how to, 271
 - Integrating with the grid, 272
 - Testing layout at design time, 272
 - TwwRecordViewPanel, 275
 - added events, 279
 - added properties, 275
 - Creating accelerators for the controls, 280
 - Customize spacing between controls, 281
 - Customizing selection and order of fields, 280
 - Embedding custom controls, 280
 - field order, 28
 - how to, 280
 - TwwRTFHeaderFooter
 - TwwDBRichEdit, 185
 - TwwSearchDialog, 6, 282
 - added events, 286
 - added methods, 288
 - added properties, 283
 - Adding functionality, 288
 - case-insensitive, 288
 - how to, 288
 - OK or Cancel, 302
 - require datasource, 301
 - search text, 302
 - SQL tables, 288
 - SQLSyncByRange property, 288
 - Using ADOTables, 288
 - Using with non-TTable, 287, 288
 - TwwStoredProc, 6, 289
 - added events, 290
 - added methods, 290
 - added properties, 289
 - how to, 290
 - TwwTable, 6, 291
 - added events, 294
 - added methods, 295
 - added properties, 291
 - field properties, 297
 - GotoCurrent, 303
 - how to, 296
 - pack, 303
 - record display order, 297
 - selecting fields, 297
 - user selection of table index. *See* TwwKeyCombo
- U**
- UnboundDataType property
 - TwwDBComboBox, 99
 - TwwDBDateTimePicker, 96
 - TwwDBEdit, 99
 - TwwDBSpinEdit, 99, 198
 - underline controls, 21
 - uninstalling InfoPower, 12
 - UnselectAll method
 - TwwDBGrid, 141
 - Unselecting all records in a grid, 146
 - UnselectRecord method
 - TwwDBGrid, 141
 - UpdateRecord method
 - TwwDBEdit, 100
 - Updating other fields
 - TwwDBLookupCombo, 147
 - UpperRangePadChar property
 - TwwFilterDialog, 215
 - URL Detection in rich edit, 181, 191
 - UseBracketsAroundFields property, TwwFilterDialog, 211
 - UseLikeOperator property, TwwFilterDialog, 211
 - UseLocateMethod property
 - TwwLocateDialog, 234
 - UseLocateMethodForSearch property
 - TwwIntl, 227
 - UseLocateWhenFindingValue property
 - TwwIntl, 225
 - UsePictureMask property, 36
 - TwwDBEdit, 99
 - User button in dialog
 - closing the dialog, 302
 - UserButton1Caption property
 - TwwDBLookupComboDlg, 164
 - TwwLookupDialog, 239
 - TwwMemoDialog, 245
 - TwwSearchDialog, 286
 - UserButton2Caption property
 - TwwDBLookupComboDlg, 164
 - TwwLookupDialog, 239
 - TwwMemoDialog, 245
 - TwwSearchDialog, 286
 - UserMessages property
 - TwwIntl, 227
 - UserSpeedButton1,
 - TwwDBRichEdit, 188
 - UserSpeedButton2,
 - TwwDBRichEdit, 189
 - UseTFields* property
 - TwwDBGrid, 123, 125, 149
 - TwwDBLookupCombo, 156
 - TwwDBLookupComboDlg, 164
 - TwwLookupDialog, 239
 - TwwSearchDialog, 286
 - using InfoPower picture masks, 32
 - using multiple grids on one dataset, 149
 - using the Design Picture Mask dialog, 39
 - using the Select Picture Mask dialog, 38
- V**
- ValidateWithMask event
 - TwwStoredProc, 290
 - ValidateWithMask property
 - TwwClientDataSet, 48
 - TwwQuery, 254
 - TwwTable, 293
 - Value property
 - TwwDBComboBox, 85
 - TwwDBSpinEdit, 198
 - TwwRadioGroup, 259
 - ValueChecked property
 - TwwCheckBox, 46
 - TwwRadioButton, 256
 - Values property

- TwvRadioGroup, 259
- ValueUnchecked property
 - TwvCheckBox, 47
 - TwvRadioButton, 256
- VersionInfoPower property
 - TwvIntl, 227
- VerticalView property
 - TwvRecordViewDialog, 263
 - TwvRecordViewPanel, 276
- view order. *See*
 - TwvKeyCombo
- View Summary Button
 - TwvFilterDialog, 206
- Visible property
 - TwvInspectorItem, 77
- VisibleItemsOnly parameter
 - TwvInspectorItem, 77

W

- WantReturns property

- TwvDBEdit, 100
- Whats new in InfoPower, 21
- Width property
 - EditorPosition in
 - TwvDBRichEdit, 184
 - TwvRecordViewDialog, 264
- Word-wrap
 - TwvDBGrid, 147
- WordWrap property
 - TwvDBEdit, 100
 - TwvInspectorItem, 77
- word-wrapping in grids, 147
- wwDittoNext
 - TwvDBGrid.DittoAttribute s, 109
- wwDittoPrior
 - TwvDBGrid.DittoAttribute s, 109
- wwDittoPriorOrNext
 - TwvDBGrid.DittoAttribute s, 109

- wwFilter property
 - TwvTable, 293
- wwFilterField
 - TwvClientDataset, 49
- wwFilterField method
 - TwvQBE, 251
 - TwvQuery, 254
 - TwvStoredProc, 290
 - TwvTable, 296
- wwFindKey method
 - TwvTable, 296
- wwidlg, 165

Y

- Year 2000 compliance
 - TwvDBDateTimePicker, 93, 95
- YearsPerColumn property
 - PopupYearOptions, TwvDBMonthCalendar, 171