

1stClass Studio™

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.

1stClass is a trademark of Woll2Woll Software. Delphi is a trademark of Embarcadero Corporation. Other brand and product names are trademarks or registered trademarks of their respective holders.

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

1stClass License Agreement

By using the software product (“1stClass”) 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 1stClass 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 1stClass source code...

You **may** use 1stClass 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 1stClass 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 1stClass component.

Woll2Woll Software reserves the right to modify or remove any function, procedure or property, that is not documented in this 1stClass Developer’s Guide, in future releases of the 1stClass 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 1stClass 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 1stClass instead of a general Delphi question. 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 1stClass 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 <http://www.woll2woll.com>.

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 support@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.

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 1stClass	1
Before You Begin.....	1
What's Included in the Developer's Guide?	2
What is 1stClass?	3
Benefits of Using 1stClass.....	4
Installing 1stClass	7
1stClass Requirements	7
Installation Steps.....	8
Uninstalling 1stClass.....	13
Distributing applications which use the 1stClass components.....	13
Building packages that use the 1stClass components.....	13
1stClass Component Overview.....	14
1stClass Sample Projects.....	14
Complete 1stClass Component Hierarchy	14
Getting Help.....	17
Using the Optional 1stClass Source Code	18
1stClass Component Reference.....	19
Description of Reference	19
TfcBitmap (Class).....	20
TfcButtonEffects (Class)	21
TfcButtonGroup	22
TfcCalcEdit	29
TfcColorCombo.....	36
TfcColorList.....	44
TfcDBImager.....	54
TfcDBTreeNode (Class).....	57
TfcDBTreeView.....	61
TfcEditFrame (Class).....	79
TfcFontCombo	82
TfcGroupBox	88
TfcImageBtn.....	90
TfcImageForm.....	99

TfcImager.....	104
TfcLabel.....	111
TfcOutlookBar	114
TfcOutlookList (Class).....	119
TfcPanel.....	126
TfcProgressBar.....	127
TfcShapeBtn	130
TfcShapeBtn	130
TfcStatusBar	135
TfcStatusPanel (Class)	139
TfcText (Class)	143
TfcTrackBar.....	148
TfcTreeCombo	155
TfcTreeNode (Class)	164
TfcTreeNodes (Class).....	176
TfcTreeHeader	182
TfcTreeHeaderSection (Class).....	186
TfcTreeView.....	188
Index.....	212

Introducing 1stClass

With the assistance of this 1stClass *Developer's Guide*, you will learn what 1stClass is, how to install the 1stClass components into your Delphi/C++ Builder development environments, how to access 1stClass demonstration forms, what each of the 1stClass 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.

Second, that you have a basic understanding of Delphi or C++ Builder terminology and the application development techniques covered in your Delphi / C++ Builder manuals. The 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 or C++ source code .
















What's Included in the Developer's Guide?

The 1stClass *Developer's Guide* is comprised of the following four main chapters:

1. **Introduction** Description of 1stClass, its requirements and how you and your end-users benefit when 1stClass components are included in your Delphi or C++ Builder based Windows applications.
2. **Installation** Complete installation instructions. Building and distributing packages which use 1stClass.
3. **Overview** Text and graphic charts showing the architecture of all 1stClass components. Reference to demonstration programs
4. **Reference** Implementation instructions for each 1stClass component, which includes complete descriptions of new properties and events added to each 1stClass component; how-to section; tips section; and Object Pascal source code examples where necessary to help you implement the 1stClass components in your applications.

What is 1stClass?

1stClass brings a wealth of high-class components to make your applications stand out. 1stClass's powerful and attractive component library installs automatically into the component palette in its integrated development environment (IDE). The 1stClass components include the following, listed alphabetically:

	TfcButtonGroup
	TfcCalcEdit
	TfcColorCombo
	TfcColorList
	TfcDBImager
	TfcDBTreeView
	TfcFontCombo
	TfcGroupBox
	TfcImageBtn
	TfcImageForm
	TfcImager
	TfcLabel
	TfcOutlookBar
	TfcPanel
	TfcProgressBar



TfcShapeBtn



TfcStatusBar



TfcTrackBar { XE “TfcTrackBar”}



TfcTreeCombo



TfcTreeHeader



TfcTreeView

Benefits of Using 1stClass

1stClass was designed for, and with assistance from, professional and corporate Windows application developers. Concerns within this group of developers include ease of development and distribution of applications, responsiveness of the application to end-user actions, and consistency across individual forms and entire applications.

Fast, Easy Application Development

Adding 1stClass components to your form is simple. Select the component from the component palette, click the point on your form where you want the component placed, enter values for properties that do not have a default value, modify other properties as necessary, add optional Object Pascal code to any of the supplied events as required, compile and run your application.

Professional, Consistent Design

Woll2Woll Software took great care in the graphic and operational design of the 1stClass components. Your end-users won't get confused or need to be retrained when you add 1stClass components to your application because they were designed to look and behave in the same manner as Delphi's built-in components.

For developers, Delphi-consistency means we've provided complete Windows on-line help for every 1stClass component and you access on-line help for 1stClass in the same manner you access Delphi on-line help—by pressing F1. For example, to view on-line help for the *Items* property of 1stClass's TfcTreeView component, select the component, select the *Items* property and then press F1. It's that simple.

Installing 1stClass

We've automated the installation of 1stClass as much as possible, but a few manual steps are still required to complete the process *before* you can access the 1stClass components and sample applications provided with 1stClass. Complete instructions for both installing and un-installing 1stClass are provided in this chapter.

1stClass Requirements

To install the 1stClass 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 C++ Builder 6.0 development environment, contain about 8MB of free hard disk space.

1stClass 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 1stClass 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 1stClass 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 `\Delphi7` directory (for Delphi 7), `\Delphi6` directory (for Delphi 6), the `\Delphi5` directory (for Delphi 5), the `\Builder6` directory (for Builder 6), the `\Builder5` directory (for Builder 5) 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.

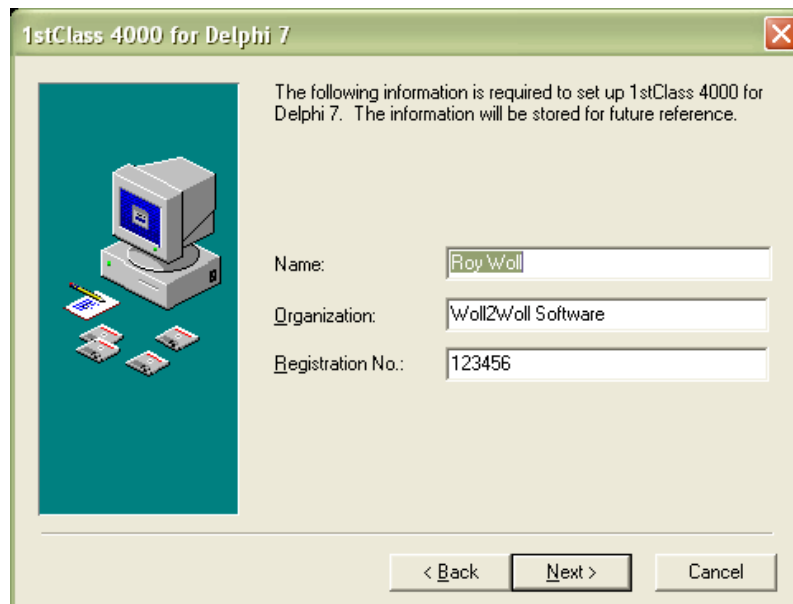


Figure 2.1 - *1stClass's Information dialog box.*

3. Select a directory to place the 1stClass files. The default directory is
“C:\Program Files\Woll2Woll\1st4000vcl7” for Delphi 7,
“C:\Program Files\Woll2Woll\1st4000vcl6” for Delphi 6,
“C:\Program Files\Woll2Woll\1st4000vcl5” for Delphi 5,
“C:\Program Files\Woll2Woll\1st4000vcl6” for C++ Builder 6
“C:\Program Files\Woll2Woll\1st4000vcl5” for C++ Builder 5
If you want to change the installation directory, then type a new name or click the Browse button to select an existing folder.

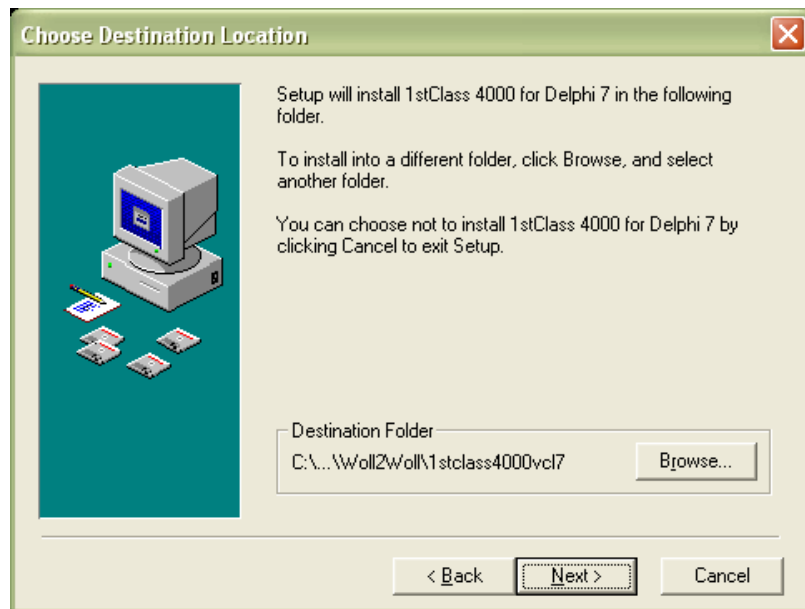


Figure 2.2 - *1stClass main installation dialog box*

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. 1stClass 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 1stClass 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 1stClass DCU library path. For instance if you installed to c:\1stClass, you would add c:\1stClass to the *Library* path edit box. If you wish to debug into the 1stClass source code, then instead add the \1stClass\source directory path to your Library Path.
3. Installing the design time package - The install program will automatically install the fc4000dcl7.bpl (for Delphi 7), fc4000dcl6.bpl (for Delphi 6 and C++ Builder 6), fc4000dcl5.bpl (for Delphi 5 and C++ Builder 5) design time packages for you. If for any reason you fail to see the 1stClass 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 fc4000dcl7.bpl (for Delphi 7), fc4000dcl6.bpl (for Delphi 6 and C++ Builder 6), or fc4000dcl5.bpl (for Delphi 5 and C++ Builder 5) to your list of design time packages for your project. This file can be found in your \1stClass\package subdirectory.
4. Optional - installing the run time package into Delphi/C++ Builder. This step is required if your applications are using the fc4000v7 (for Delphi 7), fc4000v6 (for Delphi 6 and C++ Builder 6), or fc4000v5 (for Delphi 5 and C++ Builder 5) run-time packages.
 - A. Click on (Project | Options | Packages).
 - B. Click on the (Runtime Packages | Add button) to add fc4000v7.dcp (for Delphi 7), fc4000v6.dcp (for Delphi 6), fc4000v6.bpi (for C++ Builder 6) fc4000v5.dcp (for Delphi 5), or fc4000v5.bpi (for C++ 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 1stClass package available to all your projects.

3 - Installing the 1stClass On-line Help Files

The Help files are automatically installed when running the SETUP program.

Installation Tip

If desired, you can move either the *IstClass* 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 as shown in Figure 2.3 below.

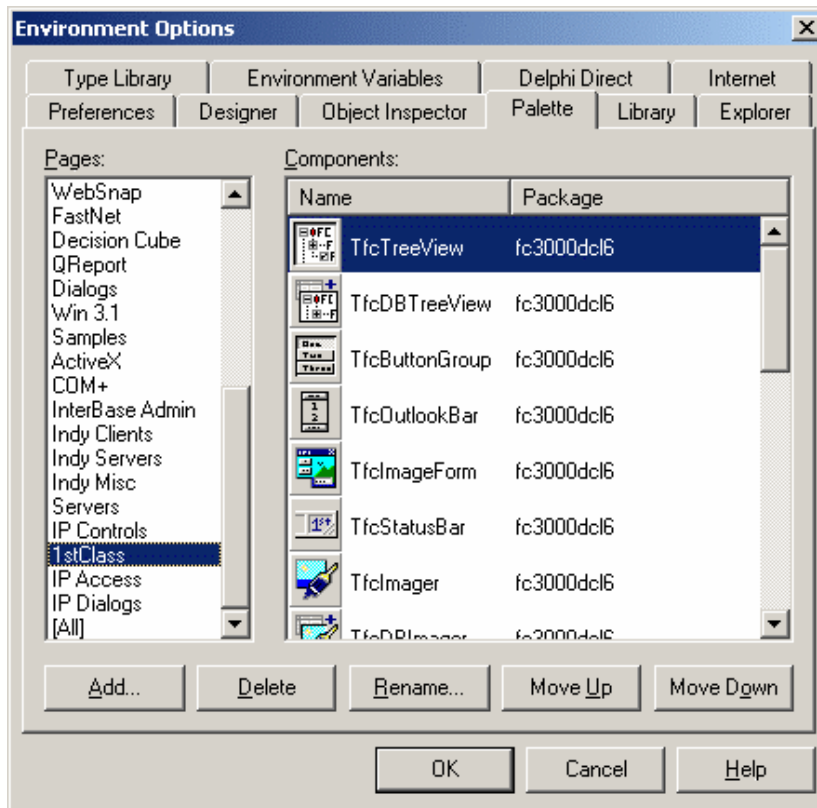


Figure 2.3 - The Delphi Environment Options dialog box with the Palette page selected, which displays the available Pages and Components lists.

3. Click and drag the *IstClass* entry displayed in the Pages list to the desired location within the list.
4. Click the OK button to close the dialog box.

Uninstalling 1stClass

Uninstalling 1stClass 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 1stClass and click on the add/remove button. Only the files that were installed with the Setup program will be removed

Distributing applications which use the 1stClass components.

If you use the 1stClass runtime package fc4000v7 for Delphi 7, fc4000v6 (for Delphi 6 and C++ Builder 6) or fc4000v5 (for Delphi 5 and C++ Builder 5) in your applications, then you will also need to distribute this file to your customer's computer. As explained in the 1stClass license, you may **not** distribute any other 1stClass file except the 1stClass runtime packages. We recommend you place this file in your customer's \windows\system directory.

If you are not using the 1stClass runtime packages when building your applications, but instead only the fc4000dcl7, fc4000dcl6 or fc4000dcl5 design time packages, then you will have no additional distribution requirements beyond what Delphi or C++ Builder already require.

Building packages that use the 1stClass components.

If you wish to build your own custom packages which require the 1stClass component library, then you will need to add the corresponding 1stClass runtime package to the required section of your package fc4000v7 for Delphi 7, fc4000v6 (for Delphi 6 and C++ Builder 6) or fc4000v5 (for Delphi 5 or C++ Builder 5).

1stClass Component Overview

When possible, each 1stClass component was modeled after one of Delphi's built-in components by inheriting either the actual Delphi component itself or one of its ancestors. This ensures that each 1stClass component contains as much of the basic functionality provided by its Delphi ancestor as possible. This section describes the following topics:

- 1stClass Sample Projects
- Complete 1stClass Component Hierarchy
- Getting Help
- Using the Optional 1stClass Source Code

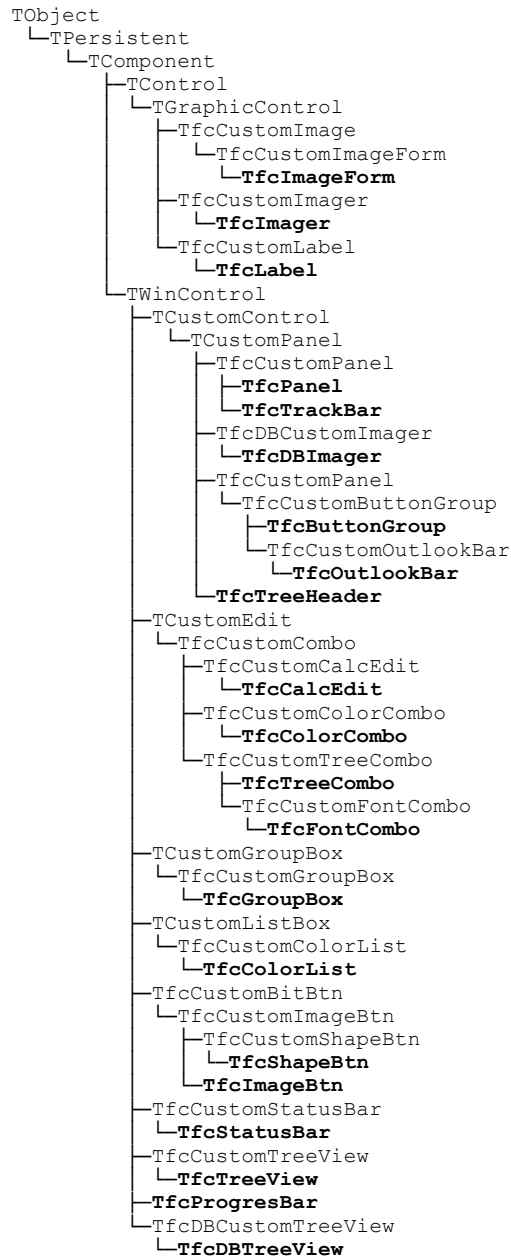
1stClass Sample Projects

Included with 1stClass are several small Delphi sample units that demonstrate the features and functionality of the 1stClass components. During installation, a subdirectory named DEMOS was automatically created within the 1stClass directory. We recommend you build and run the main demonstration program as it includes all of the 1stClass demos in one project. The main demonstration program is located in your 1stClass sub-directory at: `..\1st3000vcl6\demos\Demo1stClass.dpr`.

Complete 1stClass Component Hierarchy

The next page contains a text-based, graphical hierarchy of the complete 1stClass component library with all Delphi ancestors being shown for each 1stClass component. This hierarchy provides 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 a 1stClass component.

Complete 1stClass Component Hierarchy



Getting Help

Windows On-line Help

Accessing on-line help for a 1stClass 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 1stClass 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 1stClass 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

When you run into problems implementing a 1stClass component, please browse our newsgroups and FAQ located at <http://www.woll2woll.com>, **before** calling our technical support department.

The information provided in our newsgroups and FAQ 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 and browse through the Index to get an idea of how it's laid out and how it can help you, before you really need it.

Using the Optional 1stClass Source Code

If you purchased the optional 1stClass component library source code, your use of this code is limited by the terms and conditions specified in the 1stClass **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 1stClass source to determine the cause.

From time to time, you may be tempted to modify one of the existing 1stClass 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 1stClass component source code in any way. In addition, you would not be able to install any 1stClass 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 1stClass components, we suggest that you do one of the following:

1. Inherit the 1stClass 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.

1stClass Component Reference

Description of Reference

This chapter of the *1stClass Developer's Guide* discusses the details of each 1stClass component or class, the properties, the methods, and events along with how-to and tips sections for each component. If the component indicates 'class', then it is not a component you can drop into your form, but instead is a supporting component for another class.

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 components, their properties or events, please read through the *Delphi User's Guide* **before** you begin working with the 1stClass component library.

TfcBitmap (Class)

TfcBitmap is a supporting class for many 1stClass components. These include the TfcImageBtn's *Image* and *ImageDown* properties, and the TfcImager's *WorkBitmap* runtime property. You may wish to access this class to retrieve or set the corresponding 1stClass property from a TBitmap. See also the Delphi *TGraphic* class for a list of its available methods and properties.

Ancestor

TGraphic
TfcBitmap

Added Methods

Clear

Clears the image

```
procedure Clear; virtual;
```

LoadFromBitmap

Loads the image from the TBitmap specified by *Bitmap*

```
procedure LoadFromBitmap(Bitmap: TBitmap); virtual;
```

SaveToBitmap

Saves the image to the TBitmap specified by *Bitmap*

```
procedure SaveToBitmap(Bitmap: TBitmap); virtual;
```


TfcButtonEffects (Class)

TfcButtonEffects is a supporting class for many 1stClass combo controls. The following properties are new in 1stClass 3000 to support the custom button effects in controls that display a button next to the edit control. These include the following controls: TfcCalcEdit, TfcColorCombo, TfcFontCombo, and the TfcTreeCombo.

Ancestor

TPersistent
 TfcButtonEffects

Properties

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.

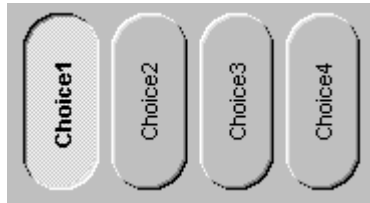
Flat

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.

TfcButtonGroup



The 1stClass *TfcButtonGroup* control allows the creation of a collection of custom-designed radio buttons or checkboxes from a single component. Use a *TfcButtonGroup* to organize buttons into logical groups, easily create radio button groups, checklist style groups, or a group of standard buttons. It has the ability to either contain a collection of shape buttons (*TfcShapeBtn*) or image buttons (*TfcImageBtn*).



The *TfcButtonGroup* control

Use the *ButtonClassName* property to select whether you are using a collection of shape buttons or image buttons. Note: If you change this property you will lose your previous button definitions.

Use the *Columns*, *ControlSpacing*, *Layout* properties to define the way the buttons are organized.

Use the *ClickStyle* property to define whether the buttons click as a radio-group (only 1 selected), as a checkbox (toggle button), or just as a regular button (click button)

1stClass provides the following design-time aids when configuring your shape button.

- If you right-click over a *TfcShapeBtn* or a *TfcImageBtn* within the *TfcButtonGroup*, the popup menu for those controls will appear in addition to the ones for the *TfcButtonGroup*.
- Double-clicking the control will bring up the standard collection editor. See the Delphi / C++ Builder docs for information on this dialog.
- When you right-click the button group at design time, you can access the following actions from the pop-up menu in addition to the menu selections for the right-clicked button.

New Button

Selecting this item will cause a new button to be added to the button group.

Ancestor

TCustomPanel
 TfcCustomTransparentPanel
 TfcCustomButtonGroup
 TfcButtonGroup

Added Properties

BevelInner, BevelOuter, BorderStyle, BorderWidth

These properties have the same meaning as the properties of the same name found in TCustomPanel.

AutoBold

When this property is *True*, the captions of the buttons in the ButtonGroup will become bold when they are depressed.

Data Type: boolean

Buttons

This property returns the *TfcCustomBitBtn* associated with the ButtonGroup item at the specified *Index*. This property is a shortcut, where *Buttons[i]* is equivalent to calling *ButtonItems[i].Button*. See the *ButtonItems* property for more information.

ButtonClassName

This property determines what type of buttons are used in the ButtonGroup. The class *must* be derived from *TfcCustomBitBtn*, and the object inspector only allows the values *TfcShapeBtn* and *TfcImageBtn*. Setting this property will clear the ButtonGroup of its buttons. At design time, a warning will be displayed.

ButtonItems

This property returns the collection *TfcButtonGroupItems* (See below), which contain the individual collection items of the ButtonGroup control. *TfcButtonGroupItems* has a default array property, so each collection can be referenced directly through *Items* using standard array notation. (i.e. *ButtonGroup.ButtonItems[i].Button.Caption*). Clicking on this property from the object inspector brings up 1stClass's collection editor.

Data Type: TfcButtonGroupItems

TfcButtonGroupItems is a class indirectly derived from TCollection. In addition to the properties defined for TCollection, *TfcButtonGroupItems* also has the following properties you can access during program execution.

ButtonGroup Returns the corresponding `TfcButtonGroup` for the `ButtonItems`.

Items *Items* is an array containing `TfcButtonGroupItem` objects. The value of the `index` parameter corresponds to the `index` property of the `TfcButtonGroupItem`.

```
property Items[Index: Integer]: TfcButtonGroupItem
```

Each button in the `ButtonGroup` has its own *TfcButtonGroupItem*. This class is indirectly derived from `TCollectionItem`. In addition to the properties found in that class, the following are added.

Button Contains the reference to the `TfcCustomBitBtn` associated with this item.

ButtonGroup Contains the reference to the `TfcCustomButtonGroup` associated with this item.

ClickStyle

This property determines whether the `ButtonGroup` behaves like a radio group, or like a checklist group.

Data Type: `TfcButtonGroupClickStyle`

Valid Values: `bcsCheckList`, `bcsRadioGroup`, `bcsClick`

bcsCheckList

This is the equivalent of a group of buttons each with their own *GroupIndex* value. The behavior is that each button can be individually pressed and unpressed.

bcsRadioGroup

This is the equivalent of a group of buttons each with same *GroupIndex* value. The behavior is that only *one* button in the group can be pressed at any one time.

bcsClick

This is the equivalent of a group of buttons the *GroupIndex* set to 0 (the Default). The behavior is the same as when buttons are just dropped onto the form without any of their properties set.

ControlSpacing

The amount of space between each button in the `ButtonGroup` is controlled by this property. Negative values will cause the buttons to overlap.

Data Type: `Integer`

Columns

The meaning of this property is related to the value of the *Layout* property. When *Layout* is *loVertical*, this property reflects how many *columns* are in the ButtonGroup. When *Layout* is *loHorizontal*, this property reflects how many *rows* are in the ButtonGroup.

Data Type: Integer

Layout

Determines the orientation of the buttons in the ButtonGroup. If *Layout* is set to *loVertical*, then the buttons are ordered from the top-down, and new columns will appear as a new column. If *Layout* is set to *loHorizontal*, then the buttons are ordered from the left to right, and new "columns" will appear as a new row.

Data Type: TfcLayout

Valid Values: *loVertical*, *loHorizontal*

MaxControlSize

This property will constrain the size of the button so that it is never larger than the specified value. This only affects the length of the side corresponding to the layout property—If *Layout* is *loVertical*, then this affects the *Height* of the button, if it is *loHorizontal*, then this affects the *Width* of the button. Use this property to achieve effects similar to that of the Windows Task Bar.

Data Type: Integer

Selected

Controls which *TfcButtonGroupItem* in the button group is pressed. This property is only valid if the *ClickStyle* property is set to *bcsRadioGroup*.

Data Type: TfcButtonGroupItem

ShowDownAsUp

This property is only valid if the *ClickStyle* property is set to *bcsRadioGroup*. When this property is set to True, the selected button will display as down only while clicking the button. When the button has become selected, then this button will be displayed as an up state button even though the actual state of the button is down. The Default is False.

Data Type: Boolean

Transparent

Controls whether or not the background of the button group will be transparent. The background is defined as any portion of the button group not covered by a button.

Data Type: Boolean

Added Events

OnChange

This event is fired immediately after the selected button changes. The parameters for this event are as follows:

ButtonGroup	ButtonGroup associated with event
OldSelected	TfcButtonGroupItem referring to the previous selection. You can refer to <i>OldSelected.Button</i> to gain access to the actual button control.
Selected:	TfcButtonGroupItem referring to the button that has just become selected. You can refer to <i>Selected.Button</i> to gain access to the actual button control.

OnChanging

This event is fired immediately before the selected button changes. The parameters for this event are as follows:

ButtonGroup	ButtonGroup associated with event
OldSelected	TfcButtonGroupItem referring to the previous selection. You can refer to <i>OldSelected.Button</i> to gain access to the actual button control.
Selected:	TfcButtonGroupItem referring to the button that is about to be selected. You can refer to <i>Selected.Button</i> to gain access to the actual button control.

Added Methods

TfcButtonGroupItems methods

The following methods are accessed through the *ButtonItems* property.

Add

Adds another button to the ButtonGroup. Returns the newly created TfcButtonGroupItem.

```
function Add: TfcButtonGroupItem;
```

FindButton

Returns the *TfcButtonGroupItem* associated with the specified button.

```
function FindButton (AButton: TfcCustomBitBtn) :  
    TfcButtonGroupItem; virtual;
```

Clear

Clears out all the buttons in the ButtonGroup. All items are freed, as well as their associated button.

```
procedure Clear; virtual;
```

Example: The following clears the buttons from the button group.

```
fcButtonGroup1.ButtonItems.Clear;
```

How To

Use the TfcImageBtn with the TfcButtonGroup

Set the ButtonClassName property to 'TfcImageBtn'. Be aware, however, that toggling this property will clear your buttons array.

Constrain the width (or height) of the buttons

If the layout for the button group is *loHorizontal*, then use the *MaxControlSize* property to prevent the buttons from taking up the entire client area of a button group. This is most useful when the button group only contains a few buttons. If the layout is *loVertical*, then *MaxControlSize* affects the buttons' heights.

Select a button in the TfcButtonGroup

There are a number of ways to select the individual buttons within the button group. Double-clicking on the button group brings up the standard collection editor. When selecting an item in this editor, the corresponding buttons get selected into the object inspector. Also, holding down the *Alt* key while clicking over a button will select the button into the object inspector.

Prevent selected buttons from appearing in bold

Turn off the *AutoBold* property.

Conserve System Resources

When using the TfcImageBtn with the button group, the same image is often used for many different buttons. Therefore, when many of the buttons have the same image, set the *ExtImage* (and) *ExtImageDown* properties to point to either a TfcImager or a TfcImageBtn. In this way multiple bitmap handles are not created for each button in the button group. For example, you can add the first button to the button group, set its Image, and then all of the following buttons will take on the properties of the first button, while doing so efficiently.

Iterate through the items in a button group

To iterate through the items within the button group, simply reference the *ButtonItems* property. For example, to have a message box pop-up for every item in the button group that is selected, execute the following code:

```
with fcButtonGroup1, ButtonItems do
begin
  for i := 0 to Count - 1 do
    if ButtonItems[i].Selected then
      ShowMessage('Item ' + ButtonItems[i].Button.Caption +
        ' Selected');
  end;
```

Make the button group behave like the windows task bar

There are a number of properties that should be manipulated to mimic the behavior of the application task bar in Windows. Set the *MaxControlSize* to a value greater than zero that corresponds to the maximum width of a button. Set *Layout* to *loHorizontal*.

Change the color of the selected button

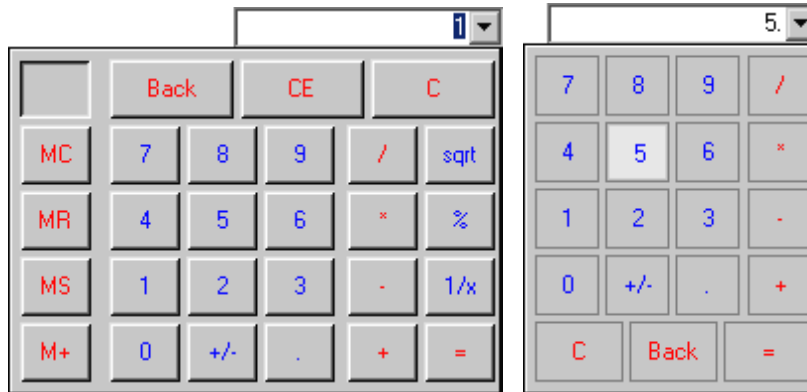
To change the color of the selected button in a button group, just use the *OnChange* event. For example:

```
procedure TForm1.fcOutlookBar1Change(
  ButtonGroup: TfcCustomButtonGroup;
  OldSelected, Selected: TfcButtonGroupItem);
begin
  if OldSelected <> nil then
    OldSelected.Button.Color := clBtnFace;
  Selected.Button.Color := clRed;
end;
```


TfcCalcEdit



Use the 1stClass *TfcCalcEdit* to provide the end-user with the ability to quickly compute new numeric values for fields that require frequent computations.



Sample TfcCalcEdit Control

To configure the display properties of the drop-down calculator control use the *CalcOptions* property. Using the *CalcOptions* property you can load a background, choose it's color, or change the button margin. You can also use the *CalcOptions | Options* to further modify the appearance and behavior of the calculator control for a nearly endless amount of variations.

If you wish to bind the control with a field in your database, then set the *DataSource* and *DataField* properties. If the *DataField* is assigned, it must specify a numeric type field. The *displayformat* property of the *TField* is respected when the control does not have the focus.

InfoPower support: If you are also using Woll2Woll's InfoPower product, you can embed the *TfcCalcEdit* into InfoPower's grid and record-view components. The steps on doing this are the same as with any InfoPower control. See the InfoPower documentation for more information on attaching a custom control to its grid or record-view.

Ancestor

```
TWinControl
  TCustomEdit
    TfcCustomCombo
      TfcCustomCalcEdit
        TfcCalcEdit
```

Added Properties

Anchors, AutoSelect, AutoSize, BorderStyle, and Constraints

These properties are equivalent to the properties of the same name found in *TEdit*. See the Delphi / C++ Builder docs under *TEdit* for more information on these properties.

Alignment

The *Alignment* of the text of the TfcCalcEdit control depends on this property when the control is unbound (Datasource and DataField not set). Otherwise it uses the TField's Alignment.

Data Type: TAlignment

AllowNull

The *AllowNull* 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. The default value is *False*, which means the user is not permitted to clear (set to Null) an existing entry.

ButtonEffects

See TfcButtonEffects for information on this property.

Data Type: TfcButtonEffects

ButtonGlyph

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


Data Type: TBitmap

ButtonStyle

Select the icon to use for this component.

Data Type: TfcComboButtonStyle

Valid Values: cbsEllipsis, cbsDownArrow, cbsCustom

cbsDownArrow The  bitmap is displayed

cbsEllipsis The  bitmap is displayed

cbsCustom: The icon defined by the *ButtonGlyph* property.

ButtonWidth

Determines the width of the Button. Set to zero for the default button width.

Data Type: Integer

CalcOptions

This property is a collection of exposed properties of the dropdown TfcCalculator control making it easier to customize the dropdown calculator control at design time.

Data Type: TfcPopupCalcOptions

Background This property specifies the background of the calculator control. Use this to load your own custom background tile or image.

Data Type: TBitmap

BackgroundStyle This property specifies how the background should be painted in the calculator control. If it is a tile, then set this to cbdTile.

Data Type: TfcCalcBitmapDrawStyle

Valid Values: cbdStretch, cbdTile, cbdTopLeft, or cbdCenter

ButtonMargin Specifies the spacing between buttons in the calculator control.

Data Type: Integer

Options This set of options specifies the behavior and display effects of the dropdown calculator control.

Data Type: Set of TfcCalcOption

Valid Values: cboHotTrackButtons, cboFlatButtons, cboHideBorder, cboHideEditor, cboShowStatus, cboHideMemory, cboSelectOnEquals, cboShowDecimal, cboSimpleCalc, cboFlatDrawStyle, cboRoundedButtons, cboDigitGrouping, cboCloseOnEquals

cboHotTrackButtons If True, then the buttons in the dropdown calculator will track the mouse. This property defaults to False.

cboFlatButtons If True, then the buttons in the dropdown calculator will appear as flat with no borders unless the button is pressed or hot-tracked. This property defaults to False.

cboHideBorder If True, then the 3D Border around the drop-down calculator will not appear and instead a simple flat style black line will appear around the outer edge of the calculator. This property defaults to False.

<i>cboHideEditor</i>	Determines visibility of the calculator display. This property defaults to True.
<i>cboShowStatus</i>	If True, then a status panel displaying the current calculation will be shown at the bottom of the calculator. This property defaults to False.
<i>cboHideMemory</i>	If True, then the dropdown calculator will not display the memory status window and the memory buttons. This property defaults to False.
<i>cboSelectOnEquals</i>	If True, then a when equals is entered the text in the calculator edit portion will be selected after the total is computed. This property defaults to False.
<i>cboShowDecimal</i>	If True, then a decimal point will always be visible. This property defaults to False.
<i>cboSimpleCalc</i>	If True, then the dropdown calculator will only show the basic calculator keys. This property defaults to False.
<i>cboFlatDrawStyle</i>	If True, then calculator buttons will be displayed in a simple flat (Outline) style. This property defaults to False.
<i>cboRoundedButtons</i>	If True, then the calculator buttons will be displayed as rounded buttons. This property defaults to False.
<i>cboDigitGrouping</i>	If True, then the thousands separator will be displayed. Useful for currency data. This property defaults to False.
<i>cboCloseOnEquals</i>	If True, then the calculator will close when the = button is pressed. This property defaults to False.

PanelColor When no bitmap is being used, set PanelColor to the color you wish the calculator to be displayed in. Default is clBtnFace.

DataField

Optional: This property contains the name of the field that you want to bind the TfcCalculator to. You can bind it to any numeric type field. If you do not wish to

bind the TfcCalcEdit to a table field, then leave both the *Datafield* and *Datasource* properties blank. The default value is blank (unbound).

Data Type: String

DataSource

Optional: This property contains the name of a TDataSource component that provides the TfcCalcEdit control with data. The default value is blank (unbound).

Data Type: TDataSource

DisplayFormat

Optional: This property describes the numeric display format for the control when it does not have the focus. If this property is left blank the field's displayformat will be used next if the control is bound to a database field.

Data Type: String

Frame

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

Data Type: TfcEditFrame

ShowButton

When this property is set to *False*, then the TfcCalcEdit's bitmap button is not shown. The default value is *True*.

Text

This property is only respected when the *DataSource* and *DataField* properties are blank. When this property is set to *True*, then the text in the TfcCalcEdit will be displayed.

Added Events

OnBeforeDropDown

Use the OnBeforeDropDown event to decide whether or not you want the drop-down calculator to appear. Call `sysutils.abort` if you wish to prevent the calendar from appearing.

The parameters for this event are as follows:

Sender: TfcCalcEdit TfcCalcEdit control associated with this event.

OnSetCalcButtonAttributes

Use the OnSetCalcButtonAttributes event to further refine the caption, color, or hint for each calculator button in the drop-down calculator.

The parameters for this event are as follows:

Sender: TfcCalcEdit TfcCalcEdit control associated with this event.

Atype: TfcCalcButtonType Atype defines the type of buttons whose attributes can be set. Possible values are:

btNone, bt0, bt1, bt2, bt3, bt4, bt5, bt6, bt7, bt8, bt9, btDecimal, btPlusMinus, btMultiply, btDivide, btAdd, btSubtract, btEquals, btSqrt, btPercent, btInverse, btBackspace, btClear, btClearAll, btMRecall, btMStore, btMClear, btMAdd

ACaption: String Caption of the current button.

AFontColor: TColor Font Color of the Button.

AButtonColor: TColor Color of Button. This is ignored when using cboFlatDrawStyle.

AHint: String Hint of Button

For an example using this event see the How To Topic on “Make calculator look like MS Money Calculator Edit Control”.

Added Methods

CloseUp

Call this method if you wish to force the drop-down list to close. Set *Accept* to *True* if you would like the control to accept the last selected entry.

```
procedure CloseUp(Accept: boolean); override;
```

DropDown

Call this method if you wish to force the combo to drop-down the color list selections

```
Procedure DropDown; override;
```

IsDroppedDown

Call this method when you want to determine if the dropdown control is visible.

```
Function IsDroppedDown: boolean; override;
```

ResetCalculator

Call this method when you wish to reset the calculator.

```
Procedure ResetCalculator; virtual ;
```

How To

Initialize an unbound CalcEdit control.

To initialize an unbound TfcCalcEdit control just set the *Text* property to the corresponding numeric value.

Make calculator look like MS Money Calculator Edit Control

If you wish to make your dropdown calculator look similar to the MS Money Calculator, then just do the following.

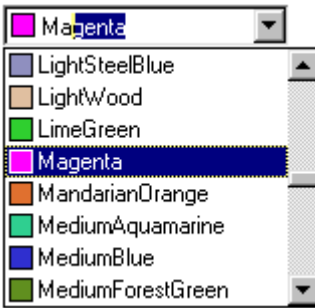
- 1) Set CalcOptions.PanelColor to clBlack and the CalcOptions.Margin = 7.
- 2) Set CalcOptions.Options to: [cboFlatButtons, cboHideBorder, cboHideEditor, cboSimpleCalc].
- 3) Finally add the following code to your OnSetCalcButtonAttributes event to color the buttons different colors.

```
procedure TForm1.fcCalcEdit1SetCalcButtonAttributes(Calc: TfcCalculator;  
  var AType: TfcCalcButtonType; var ACaption: String;  
  var AFontColor: TColor; var AButtonColor:TColor; var AHint: String);  
begin  
  case AType of  
    bt0..bt9, btDecimal: AButtonColor := clWhite;  
    btClear, btBackSpace, btClearAll: AButtonColor := clLime;  
    btDivide, btMultiply, btAdd, btSubtract, btPlusMinus, btEquals:  
      AButtonColor := clBtnFace;  
  end;  
  AFontColor := clBlack;  
end;
```

TfcColorCombo



Use the 1stClass *TfcColorCombo* to provide the end-user with the ability to select a color value or a color string from a drop-down listbox.



TfcColorCombo Control with ShowMatchText

To configure the display properties of the drop-down list use the *ColorListOptions* property. The color choices available in the drop-down list are defined by the *ColorListOptions | Options* property. You can further customize the colors by adding your own colors through the *TfcColorCombo*'s *CustomColors* property. Use the *ColorListOptions | SortBy* property to control the order of the colors in the drop-down list.

If you wish to bind the control with a field in your database, then set the *DataSource* and *DataField* properties. If the *DataField* is assigned, it must specify a string or integer field. If you attach the control to a database integer field, then the drop-down list will display the text names of the colors, but store the color as a number in the database.

InfoPower support: If you are also using Woll2Woll's InfoPower product, you can embed the *TfcColorCombo* into InfoPower's grid and record-view components. The steps on doing this are the same as with any InfoPower control. See the InfoPower documentation for more information on attaching a custom control to its grid or record-view. See the how-to topics at the end of this section for information on displaying the color boxes of a *TfcColorCombo* for all rows in an InfoPower grid.

Use the *ShowMatchText* property to enable incremental searching as the user types. This option also updates the control's display so that the matching text is displayed in the control.

Set the *Style* property to *csDropDownList* to force the entry to come from the list. If *AllowClearKey* is *False*, then the user is not permitted to clear an existing entry.

Use the *OnAddNewColor* event to allow your end users to add new colors to the list, or use the *OnFilterColor* event to filter out colors based on some criteria you define.

Ancestor

TWinControl
 TCustomEdit
 TfcCustomCombo
 TfcCustomColorCombo
 TfcColorCombo

Added Properties

AutoSize, BorderStyle, and CharCase

These properties are equivalent to the properties of the same name found in *TEdit*. See the Delphi / C++ Builder docs under *TEdit* for more information on these properties.

AlignmentVertical

The value of this property determines how the text in the *fcColorCombo* will be vertically aligned. This property defaults to *fcavTop*.

Data Type: TfcAlignVertical

Valid Values: fcavTop, fcavCenter

AllowClearKey

When the style is set to *csDropDownList*, the user is not permitted 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. The default value is *False*, which means the user is not permitted to clear an existing entry.

Data Type: boolean

AutoDropDown

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

Data Type: boolean

ButtonEffects

See *TfcButtonEffects* for information on this property.

Data Type: TfcButtonEffects

ButtonGlyph

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

Data Type: TBitmap

ButtonStyle

Select the icon to use for this component. If the property is set to *cbsEllipsis*, then a ColorDialog will pop-up. Otherwise, if the property is set to *cbsDownArrow* then the color listbox will drop-down instead.

Data Type: TfcComboButtonStyle

Valid Values: cbsDownArrow, cbsEllipsis

ColorAlignment

Determines the alignment of the color relative to the text.

Data Type: TLeftRight

Valid Values: taLeftJustify, taRightJustify

ColorDialog

If you have a color dialog setup with its own custom colors and events, this property allows you to override the color dialog that the TfcColorCombo uses.

Data Type: TColorDialog

ColorDialogOptions

This set of properties allows for the manipulation of the popup color dialog's options. Set *cdoEnabled* to *True* to pop-up a colordialog when double clicking on a colorcombo. For more information on using these options see the Delphi / C++ Builder docs under the *Options* property of the *TColorDialog*.

Data Type: TfcColorDialogOptions

Valid Values: cdoEnabled, cdoPreventFullOpen, cdoFullOpen, cdoSolidColor, cdoAnyColor

ColorListOptions

This property is a collection of exposed properties of the dropdown TfcColorList control making it easier to customize the dropdown color listbox control. For more information on these properties see *TfcColorList*.

Controller

See InfoPower TwwController property

CustomColors

This property is equivalent to the *CustomColors* property described in TfcColorList. (See TfcColorList *CustomColors*).

DataField

Optional: This property contains the name of the field that you want to bind the TfcColorCombo to. You can bind it to a string or integer data type. If you do not wish to bind the colorcombo to a table field, then leave both the *Datafield* and *Datasource* properties blank. The default value is blank (unbound).

Data Type: String

DataSource

Optional: This property contains the name of a TDataSource component that provides the ColorCombo control with data. The default value is blank (unbound).

Data Type: TDataSource

DropDownCount

The *DropDownCount* property determines how many entries will appear in the dropdown color list box.

Data Type: Integer

DropDownWidth

This *DropDownWidth* property determines how wide the drop-down color 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

Frame

See TfcEditFrame for more information on this property.

Data Type: TfcEditFrame

SelectedColor

Use *SelectedColor* to retrieve or set the current color selection. Set the value of *SelectedColor* to the color of the item to be selected. If no item is selected, the value is clNullColor, which is the default value.

Data Type: TColor

ShowButton

When this property is set to *False*, then the ColorCombo's bitmap button is not shown. The default value is *True*.

Data Type: boolean

ShowMatchText

When this property is set to *True*, the ColorCombo will perform ‘Quicken’ style incremental searching. As the user enters text, the control will simultaneously search and display the matching text in the control. If the property is set to *False* and the ColorCombo is dropped down, then the ColorCombo will toggle to the next item that starts with the letter entered. The default value is *True*.

Data Type: boolean

Style

This property determines the *style* of the color combo box. 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. If the property *ShowMatchText* is *True*, the user is permitted to type in a valid choice.

Data Type: TfcComboStyle

Valid Values: fcCombo.csDropDown, fcCombo.csDropDownList

UnboundAlignment

This property is only respected when the *DataSource* and *DataField* properties are blank. When this property is set to *True*, then the text in the ColorCombo’s will be aligned to the left or the right depending on your setting.

Data Type: TLeftRight

Valid Values: taLeftJustify, taRightJustify

Added Events

OnAddNewColor

See TfcColorList’s *OnAddNewColor* event.

OnCloseColorDialog

This will be called whenever the colordialog closes. The parameters for this event are as follows:

<i>Sender:</i> TObject	Color listbox control that is associated with this event.
<i>Dialog:</i> TColorDialog	Color dialog associated with this event.
<i>MResult:</i> TModalResult	Modal Result of the color dialog.
<i>var Accept:</i> boolean	Accepting the entry will put the combo in edit mode and the <i>SelectedColor</i> will be set to the color dialog value.

OnCloseUp

This event is fired immediately after the drop-down list closes. Use this event to perform your own custom action after the drop-down list closes. The parameters for this event are as follows:

<i>Sender</i> :TObject	TfcColorCombo control that is associated with this event.
<i>Select</i> : boolean	This value is <i>True</i> if the user is making a selection. If the user is escaping out of the drop-down list without making a selection, then the value of <i>Select</i> is <i>False</i> .

OnDropDown

This event is fired immediately before the color list is dropped down. The parameters for this event are as follows:

<i>Sender</i> :TObject	TfcColorCombo control that is associated with this event.
------------------------	---

OnFilterColor

This will be called whenever the color listbox is dropped down. For more information on this event, see TfcColorList's *OnFilterColor* event.

OnInitColorDialog

This will be called before the ColorDialog is executed to allow you to further customize the ColorDialog before it is shown.

The parameters for this event are as follows:

<i>Sender</i> :TObject	Color listbox control that is associated with this event.
<i>Dialog</i> :TColorDialog	ColorDialog associated with this event.

Added Methods

CloseUp

Call this method if you wish to force the drop-down list to close. Set *Accept* to *True* if you would like the control to accept the last selected entry.

```
procedure CloseUp(Accept: boolean); override;
```

DrawInGridCell

Call this method if you wish to use the ColorCombo information to accurately draw the selected item in an InfoPower grid.

```
Procedure DrawInGridCell(ACanvas:TCanvas; Rect:TRect;  
State:TGridDrawState); override;
```

DropDown

Call this method if you wish to force the combo to drop-down the color list selections

```
Procedure DropDown; override;
```

ExecuteColorDialog

Displays the Microsoft color dialog to the end-user so they can pick a color from this dialog. The return value is *False* if the user cancelled the dialog without making a selection.

```
Function ExecuteColorDialog: boolean; virtual;
```

GetColorFromRGBString

Converts the color string specified by *RGBString* to a TColor value, referred to by *AColor*. Returns *True* if the operation was successful. *RGBString* is of the format RGB:ddd,ddd,ddd, where ddd represents a number between 0 and 255.

```
Function GetColorFromRGBString (RGBString:String;  
  var AColor: TColor): boolean;
```

IsCustomColor

Returns *True* if the color referred to by parameter *s* is in the combo's custom colors list. See also the *CustomColors* property of the TfcColorCombo.

```
function IsCustomColor (s: string): boolean;
```

IsDroppedDown

Call this method when you want to determine if the dropdown control is visible.

```
Function IsDroppedDown:boolean; override;
```

RefreshList

This method will cause the list of colors in the drop-down color listbox to reload.

```
procedure RefreshList; virtual;
```

How To

Iterate through all of the filtered colors in the color combo.

When using the *OnFilterColor* event, some colors filtered out of the *AllColors* property. To iterate through the list of valid colors in the fcColorCombo listbox control you would use the *items* property as in the following example:

```
procedure TForm1.Button1Click(Sender: TObject);  
  var i:integer;  
  begin  
    with (fcColorCombo1.ListBox.Items) do  
      for i:= 0 to Count-1 do  
        ListBox1.Items.Add(Names[i]);  
  end;
```

Initialize an unbound ColorCombo control.

To initialize an unbound TfcColorCombo control just set the *SelectedColor* property to the corresponding color value.

Display the color boxes of the TfcColorCombo in all rows of an InfoPower Grid.

When you embed the ColorCombo into an InfoPower grid, the color box for the control is displayed in the control, but not necessarily for the other rows of the grid. In InfoPower 3000 you can just check the Control Always Paints checkbox in the Edit Control tab page of the Grid's Selected property dialog.

For InfoPower 2000, the following code allows you to paint the color box in all the rows of the grid for the column containing the ColorCombo. It uses a public procedure named "fcGetControlInGrid", declared in fcombo, to retrieve the control associated with a column in the grid.

```
procedure TForm1.wwDBGrid1DrawDataCell(Sender: TObject;  
    const Rect: TRect; Field: TField; State: TGridDrawState);  
var Control: TfcCustomCombo;  
begin  
    Control := fcGetControlInGrid(self,  
        Sender as TwwDBGrid, Field.FieldName);  
    if Control <> nil then Control.DrawInGridCell((Sender as  
        TwwDBGrid).Canvas, Rect, State);  
end;
```

Tips

- See the TfcColorList's properties for details on customizing the drop-down color control.

TfcColorList



Use the 1stClass TfcColorList to provide the end-user with the ability to select a color value or a color string from a listbox.



TfcColorList control

The colors displayed in the listbox are defined by the *Options* property. You can further customize the colors by adding your own colors through the TfcColorList's *CustomColors* property. Use the *SortBy* property to control the order of the colors.

Use the *OnAddNewColor* event to allow your end-users to add new colors to the list, or to filter out colors based on some criteria you define in the event.

Ancestor

TWinControl
 TCustomListBox
 TfcCustomColorList
 TfcColorList

Added Properties

Alignment

Determines the alignment of the text in the non-color portion of the listbox.

Data Type: TLeftRight

Valid Values: taLeftJustify, taRightJustify

AllColors (Runtime Only)

This property is a TStringList where the Name is the string representation of the color and the Value is the Color in BGR (Blue, Green, Red) hex values. You can use the Names or Values properties of the TStringList to access these values. *AllColors* will contain all the possible color/value pairs in the listbox including the ones that the *OnFilterColor* event filters out. If you are using the *OnFilterColor* event and wish to reference a given color in the color listbox by number, then you should use the *Items* property instead.

The example below illustrates how to copy all of the possible names in the color listbox control to a standard Delphi TListBox named Listbox1.

```
procedure TForm1.fcShapeBtn1Click(Sender: TObject);  
var i:integer;  
begin  
    for i:= 0 to fcColorList1.AllColors.Count -1 do  
        Listbox1.Items.Add(fcColorList1.AllColors.Names[i]);  
end;
```

To get the color value as a TColor based on an index, see the *ColorFromIndex* method.

ColorAlignment

Determines the alignment of the color relative to the text.

Data Type: TLeftRight

Valid Values: taLeftJustify, taRightJustify

ColorMargin

Then Default ColorMargin is 2. You can change this value to reduce or enlarge the padding around the color rectangle.

Data Type: Integer

ColorWidth

When the *ColorWidth* property is set to 0, the width of the color rectangle will be calculated based on the *ItemHeight* property. Otherwise the width of the color rectangle will be as wide as you set it.

Data Type: Integer

CustomColors

This property determines what custom colors you can add to the color listbox. Each custom color is represented as a string of the form ColorName = HexValue. For example the following string sets the first custom color. The hex value is in the Delphi BGR format. In order for the listbox to display these colors, *Options | ccoShowCustomColors* must be set.

```
SkyBlue = CC9932
```

Data Type: TStringList

GreyScaleIncrement

This property determines how many shades of gray that will show up in the control when *Options | ccoShowGreyScale* is *True*. The ColorList uses this value to increment the shades of grey from 0 to 255.

Data Type: Integer

Valid Values: A Positive Number

ItemIndex (Runtime Only)

Generally you should use *SelectedColor* instead, however you can use *ItemIndex* to select an item at runtime. Set the value of *ItemIndex* to the index of the item to be selected. The *ItemIndex* of the first item in the list box is 0. If no item is selected, the value is -1, which is the default value.

Data Type: Integer

Valid Values: -1, 0, or a positive integer.

Items (Runtime Only)

This property is a TStringList using the Name=Value form, where Name is the string representation of the color and the Value is the Color in BGR (Blue, Green, Red) hex values. You can use the Names or Values properties of the TStringList to access these values. While *AllColors* will contain all the possible color/value pairs in the listbox, the actual listbox can contain much less colors if the *OnFilterColor* event filters them out. If you are using the *OnFilterColor* event and wish to reference a given color in the color listbox by number, then you should use the *Items* property instead. For example to get the name of the first item in the list you could reference it using code like:

```
ColorName := fcColorList1.Items.Names[0];
```

To get the color value as a TColor based on an index, see the *ColorFromIndex* method.

Data Type: TStringList

NoneString

When *Options | ccoShowColorNone* is *True*, the default string for *clNone* is “None”. Use this property to override this string with your own descriptive name.

Data Type: String

Options

This property is a set of boolean flags that control the display of the colors in the ColorList control.

Data Type: TfcColorListBoxOptions

Valid Values: ccoShowSystemColors, ccoShowColorNone, ccoShowCustomColors, ccoShowStandardColors, ccoShowColorNames, ccoShowGreyScale, ccoGroupSystemColors

cocoShowSystemColors

When set, the ColorList will display system colors in the listbox control. (i.e. *clBtnFace*, *clWindow*, etc.)

cocoShowColorNone

When set, the ColorList will display the *clNone* color in the listbox control. Use the *NoneString* property to override the display of the text.

cocoShowCustomColors

When set, colors defined in the *CustomColors* stringlist property will be displayed in the color listbox control.

cocoShowStandardColors

When set, the ColorList will display the standard windows colors in the listbox control. Default is *True*.

cocoShowColorNames

When set, the ColorList will display the names of the colors in the listbox control along with the color rectangles. When set to *False*, only the color rectangle will be visible. Default is *True*.

cocoShowGreyScale

When set, the ColorList will add a series of GreyScale colors to the listbox control incrementing the *GreyScaleIncrement* property from 0 to 255, where 0 is Black and 255 is White.

cocoGroupSystemColors

When set, the SystemColors will be grouped separately from the other colors, when the *SortBy* property is set to *csoByName* or *csoNone*.

SelectedColor

Use *SelectedColor* to select a color at runtime or design time. Set the value of *SelectedColor* to the color of the item to be selected. If no item is selected, the value is *clNullColor*, which is the default value.

Data Type: TColor

SortBy

This property determines how the ColorList is Sorted.

Data Type: TfcSortByOption

Valid Values: (csoNone, csoByName, csoByRGB, csoByIntensity)

csoNone

When *SortBy* is set to *csoNone*, the list is not sorted.

csoByName

When *SortBy* is set to *csoByName*, the list is sorted by the color name. Note: If *Options | ccoGroupSystemColors* is *True*, then the *SystemColors* are sorted as a separate group from the all of the other colors.

csoByRGB

When *SortBy* is set to *csoByRGB*, the list is sorted by the RGB values. Using this property will group similar colors together based on the color value.

csoByIntensity

When *SortBy* is set to *csoByIntensity*, the list is sorted based on the average of the RGB values, so the most vivid colors will be first and the more faded, lighter ones will be near the end.

Added Events

OnAddNewColor

Occurs when the end-user adds a new color to the color listbox control by assigning to the *SelectedColor* property a value that is not in the list of colors. You can use this event to not allow new colors to be added to your color listbox, or to change the name of new colors that are being added to the listbox. The default name for a new color is of the form: RGB: 128, 128, 128.

The parameters for this event are as follows:

<i>Sender</i> : TObject	Color listbox control that is associated with this event.
<i>AColor</i> : TColor	Color that was not found in the list.
<i>var AColorName</i> : String	Rename the new color to a more meaningful color name.
<i>var Accept</i> : boolean	Set this variable to <i>True</i> to accept the entry, and <i>False</i> otherwise.

Example: The following code will prompt the user for confirmation to add the new RGB Color to the list. Here you could prompt the end user to enter a color name for the new color that is being added.

```
procedure TForm1.fcColorList1AddNewColor(Sender: TObject;  
    AColor: TColor; var AColorName: String; var Accept: Boolean);  
begin  
    if Pos('RGB:',AColorName) =1 then begin  
        if MessageDlg('Add New Color?', mtConfirmation,  
            mbYesNoCancel, 0)=mrYes then  
            Accept := True
```

```

        else Accept := False;
      end;
    end;

```

OnFilterColor

Occurs whenever the list is reinitialized, refreshed or resorted.

The parameters for this event are as follows:

<i>Sender</i> :TObject	Color listbox control that is associated with this event.
<i>AColor</i> :TColor	Color value.
<i>AColorName</i> :String	Color name.
<i>var Accept</i> :boolean	Set this variable to <i>True</i> to accept the entry, and <i>False</i> otherwise.

Example: The following code will allow filter all colors that have red as part of the color name.

```

procedure TForm1.fcColorList1FilterColor(Sender: TObject;
  AColor: TColor; AColorName: String; var Accept: Boolean);
begin
  if Pos('Red',AColorName) = 0 then Accept := False
end;

```

Added Methods

ColorFromIndex

Call this function to retrieve the specified color in the *Items* List, based on the passed in *Index* of the listbox control.

Function ColorFromIndex(Index: Integer):TColor; **virtual**;

InitColorList

Call this method to reload and reinitialize the ColorList.

procedure InitColorList; **virtual**;

SortList

Call this method to sort the color list based on the *SortBy* property.

procedure SortList; **virtual**;

How To

Allow only a limited set of colors

There are a couple of ways to make available only a limited set of colors for the end user. One way is to use the *OnFilterColor* event. However, the easiest way to restrict your color choices is to use the *Options | ccoShowCustomColors* property. Here are the steps.

1. Set the *Options | ccoShowStandardColors* , *Options | ccoShowSystemColors*, and *Options | ccoShowGreyScale* property to *False*.
2. Set the *Options | ccoShowCustomColors* property to *True*.
3. Click on the *CustomColors* StringList editor and add the colors that you wish to the *CustomColors* list. (For a complete list of the color values and color dialog color values see the colors.txt file in the demos directory)

For Example:

```
Red=0000FF
Blue=FF0000
Green=00FF00
Yellow=00FFFF
```

Align the color box on the right side

To align the Color Box on the right side of the text, just set the *ColorAlignment* property to *taRightJustify*.

Retrieve all the user's color selections when MultiSelect is True

To create a multiselectable list of colors, just set the *Multiselect* property to *True* and the *ExtendedSelect* property to *True* if you wish to have Shift Select capability. Then in order to iterate through the selected colors, you just iterate through all of the items of the color list box and check the selected property. The following example will display each of the multiselect colors of the color list box control.

```
procedure TForm1.Button1Click(Sender: TObject);
var i:integer;
begin
    for i:= 0 to fcColorList1.Items.Count-1 do
        if (fcColorList1.Selected[i]) then
            ShowMessage(ColorToString(fcColorList1.ColorFromIndex(i)));
end;
```

Add color dialog support to add new colors to the color list.

You may wish to allow your end-users to add their own custom colors to this list at runtime. The easiest way to do this is to execute a color dialog when the end-user double clicks on the listbox and chooses a color that is not in the list of colors.

1. Set the *Options | ccoShowStandardColors* to *True* and *Options | ccoShowCustomColors* to *True*.

2. Click on the *CustomColors* StringList editor and add the default color dialog colors to the list. For example, these are the default Color Dialog Colors:

LightCoral=8080FF	DeepSeaBlue=804000
Brick=404080	LighterTeal=808040
DarkBrown=000040	SkyBlue=FF8000
GoldenRod=80FFFF	SlateBlue=C08000
Coral=4080FF	LavenderBlue=FF8080
Orange=0080FF	MidnightBlue=A00000
Sienna=004080	BluishBlack=400000
MintGreen=80FF80	Pink=C080FF
LawnGreen=00FF80	Lavender=C08080
DarkGreen=004000	Redwood=400080
OliveDrab=408080	PurplishBlack=400040
LightGreen=80FF00	NeonPink=FF80FF
LighterGreen=40FF00	LipstickRed=8000FF
ForestGreen=408000	Violet=FF0080
DarkForestGreen=404000	Indigo=800040
PowderBlue=FFFF80	

- Now drop a TColorDialog onto your form, and set the *ColorDialogOptions* | *cdPreventFullOpen* to *False* if you want your end-users to add any additional colors to the list. Otherwise, set it to *True*. Then add the following code to the *OnDblClick* event of the TfcColorList.

```
Procedure TForm1.fcColorList1DblClick(Sender: TObject);
begin
    //Initialize Color Dialog to list's selected color.
    ColorDialog1.Color := (Sender as TfcColorList).SelectedColor;

    if ColorDialog1.Execute then
        (Sender as TfcColorList).SelectedColor
        :=ColorDialog1.Color;
    end;
```

- Now add fcCommon to your form's uses clause, and put the following code in the *OnAddNewColor* event.

```
Procedure TForm1.fcColorList1AddNewColor(Sender: TObject;
    AColor: TColor; var AColorName: String; var Accept:
    boolean);
var ColorIndex:Integer;
begin
    with ColorDialog1,CustomColors do begin
        ColorIndex := fcValueinList(IntToHex(-1,8),CustomColors);

        if ColorIndex <> -1 then //Unused Custom Color found
        begin
            if fcValueinList(
                IntToHex(AColor,6),CustomColors)=-1 then
                Values[Names[ColorIndex]]:= IntToHex(AColor,6);
                Accept := True; //Add to color listbox control
            end
            else Accept := False; //Don't add if CustomColors is full
        end;
    end;
```

Drag a color to change the font color of a label

The following example demonstrates how you can drag colors from the color list to change the font color of any control in your application.

This can be accomplished via the following steps:

- Add a TfcLabel component to your form and set the following properties:
Caption = 'My Label' (Set to whatever text you wish to display)
- Put the following code in the TfcLabel's *OnDragOver* event.

```
Procedure TForm1.Label1DragOver(Sender, Source: TObject;
    X, Y: Integer; State: TDragState; var Accept: boolean);
begin
```



```
    if (Source is TfcColorList) then Accept := True;
end;
```

3. Put the following code in the TfcLabel's *OnDragDrop* event.

```
Procedure TForm1.Label1DragDrop(Sender, Source: TObject;
    X, Y: Integer);
begin
    if (Source is TfcColorList) and (Sender is TControl) then
        TEdit(Sender).Font.Color :=
            (Source as TfcColorList).SelectedColor;
end;
```

4. Finally, set the *DragMode* of the TfcColorList to *dmAutomatic*.

Tips

- See the colors.txt document in the demos directory for a sample list of colors for the *CustomColors* property.

TfcDBImager



The 1stClass TfcDBImager Control allows you to display images stored in your database. This versatile control can be used in a TDBCtrGrid as well as many of InfoPower's Grid and DataInspector controls. Store and display jpgs, bitmaps, metafiles, and icons into your database blob fields.



TfcDBImager control

Use the *BitmapOptions* to add special effects to the image as it is displayed. Set the *DrawStyle* property to specify if the image is stretched, tiled, centered, etc. For more info on similar properties see TfcImager.

Ancestor

TCustomControl
TfcDBCUSTOMImager
TfcDBImager

Added Properties

This component has all of the properties of the TfcImager, plus the following additional properties:

BitmapOptions

Use Bitmap Options to set one of many different effects that can be applied to the Image. These changes are not stored in the database. However, you can reference the WorkBitmap property to access the changed bitmap. When used in a grid or a

control that handles the csPaintCopy control state. Only the active image will have the BitmapOptions settings applied to it.

DataField

This property contains the name of the field that you want to bind the TfcDBImager to. The default value is blank (unbound).

Data Type: String

DataSource

This property contains the name of a TDataSource component that provides the TfcDBImager control with data.

Data Type: TDataSource

PictureType

Defines the type of image that is attached to this particular set of different effects that can be applied to the Image. If you wish to store and display jpgs in the database

Data Type: TfcImagerPictureType

Valid Values: fcptBitmap, fcptJpg, fcptMetafile, fcptIcon

fcptBitmap	Blob field is a Bitmap
fcptJpg	Blob field is a Jpg
fcptMetafile	Blob field is a Metafile
fcptIcon	Blob field is a Icon

Picture (Runtime Only)

This property contains the actual graphic that is being displayed.

Data Type: TPicture

Added Events

OnCalcPictureType

Occurs when a picture is loaded into the TfcDBImager and when it is being painted. Use this event only if you are storing different image types in the same field of the database. This event requires you to know based on some other field what type of image is stored for the field attached to the TfcDBImager.

The parameters for this event are as follows:

ImageControl:TfcDBImager Image control associated with this event.

var PictureType:TfcImagerPictureType Set this variable to the associated PictureType for this blob field's stored image.

Example: In the following code `wwTable1` points to a table with two fields. The “PictureType” field describes the type of image that was loaded into the blobfield associated with this `TfcDBImager` control.

```
procedure TForm1.fcDBImager1CalcPictureType(ImageControl:
  TfcDBImager; var PictureType: TfcImagerPictureType);
begin
  case wwTable1.FieldName('PictureType').AsInteger of
    0:PictureType := fcptBitmap;
    1:PictureType := fcptjpg;
    2:PictureType := fcptMetafile;
    3:PictureType := fcptIcon;
  end;
end;
```

How To

Integrate the DBImager into a TDBCtrGrid

The Simply drop this in a `TDBCtrGrid`. For additional indication of which image has the focus, set the `BitmapOptions` properties accordingly.

Integrating a TPicture dialog with a TfcDBImager

To allow the end-user to add/modify the current image, you can either add a popupmenu or respond to a dbl-click event.

```
procedure TForm1.fcDBImager1DblClick(Sender: TObject);
var
  blobstream:TBlobStream;
  photostream:TFileStream;
begin
  with (Sender as TfcDBImager),DataSource.DataSet do
  try
    if openpicturedialog1.execute then begin
      Edit;
      photostream:=tfilestream.create(openpicturedialog1.filename,
        fmopenread or fmsharenedenywrite);
      blobstream :=TBlobstream.create(FieldName(DataField) as
        TBlobField,bmwrite);
      try
        blobstream.copyfrom(photostream,photostream.size);
      except
        photostream.free;
        blobstream.free;
        Dataset.Cancel;
      end;
    end;
    if (State in [dsInsert,dsEdit]) then Post; // Optional
  except
    ShowMessage('Invalid Picture!');
  end;
end;
```

TfcDBTreeNode (Class)

TfcDBTreeNode describes a painted node in a *TfcDBTreeView* control. Each node in a tree view control consists of a number of attributes, and can itself contain 0 or more nodes. *TfcDBTreeNode* is not a design time component you see in your IDE palette, but is created and manipulated internally by the *TfcDBTreeView*. Every time the tree is repainted, the memory associated with the tree nodes is disposed and new ones are re-allocated. For this reason, you should not save a *TfcDBTreeNode*'s pointer value into your own data structures. The pointer would become invalid the next time the tree is repainted. Instead you should reference a *TfcDBTreeNode* only in the context of the *TfcDBTreeView* events.

Ancestor

TObject
TfcDBTreeNode

Added Properties

DataLink

DataLink indicates the *TDataLink* associated with the node.

Data Type: *TDataLink*

DataSet

DataSet indicates the *TDataSet* associated with the node.

Data Type: *TDataSet*

Expanded

Expanded indicates if the node has been expanded so that its children are displayed.

Data Type: boolean

HasChildren

HasChildren returns *True* for all nodes except nodes tied to the last datasource. The last datasource is either defined by the *DataSourceLast* property, or the last datasource specified in the *DataSources* property.

Set this property if you wish to define which nodes have children.

Example: See the *TfcDBTreeView* *how-to* topic *on disabling the expand icon (+) for specific nodes*.

Data Type: boolean

Hot

Hot returns *True* if a node is currently being hot-tracked. The text of a hot-tracked node is displayed with a blue underlined font. You can set this to *False* to selectively disable a node from being displayed as a hot-track node. See also the *TfcDBTreeView Options | dtvoHotTracking* property.

Example: See the *TfcDBTreeView* how-to topic “How to control which specific nodes are displayed as a hot-track”.

Data Type: boolean

ImageIndex

ImageIndex specifies which image is displayed in the tree. *ImageIndex* is the index from the image list specified by the tree’s *Images* property. If the tree’s *Images* property is not assigned then this property does nothing. Use the *TfcDBTreeView OnCalcNodeAttributes* event to customize the image based on run-time criteria.

Example: The following example displays the first image from the tree’s *Images* when the ‘Country’ field is 'US', and the 2nd image otherwise.

```
if Node.DataSet = CustomersTbl then begin
  if Node.DataSet.FieldName('Country').asString = 'US'
then
  Node.ImageIndex:= 0
  else Node.ImageIndex:= 1;
end;
```

Data Type: integer

Level

Level indicates the level of indentation of a node within the *TfcDBTreeView* control.

The value of *Level* is 0 for nodes on the top level. The value of *Level* is 1 for their children, and so on.

Data Type: integer

MultiSelected

MultiSelected returns *True* if the node has been multi-selected. See the *MultiSelectAttributes* for information on enabling multi-select in the tree.

Data Type: boolean

Parent

Parent identifies the parent node of the tree node. A parent node is one level higher than the node and contains the node as a subnode.

Data Type: *TfcDBTreeNode*

Selected

Selected returns *True* if the node is the active node.

Data Type: boolean

StateIndex

StateIndex specifies which state image is displayed in the tree. If the tree is displaying both state images and images, then the image associated with *StateIndex* is displayed before the image specified by *ImageIndex*. *StateIndex* is the index from the image list specified by the tree's *StateImages* property. If the tree's *Images* property is not assigned then this property has no effect. If the tree is displaying a checkbox in the node, then the state image is not displayed. Use the *TfcDBTreeView.OnCalcNodeAttributes* event to customize the image based on run-time criteria.

Example: The following example displays the first image from the tree's *StateImages* when the 'Country' field is 'US', and the 2nd image otherwise.

```
if Node.DataSet = CustomersTbl then begin
  if Node.DataSet.FieldName('Country').asString = 'US' then
    Node.StateIndex:= 0
  else Node.StateIndex:= 1;
end;
```

Data Type: integer

Text

Text refers to the displayed text for the node. Use the *TfcDBTreeView's DisplayFields* property to customize the appearance of the node's text.

Data Type: String

Added Methods

GetFieldValue

Use *GetFieldValue* to retrieve the node's data. You can refer directly to the dataset to retrieve field information for the active node or its parents. However to get the inactive node's field information you need to use this method.

```
function GetFieldValue(FieldName: string): Variant;
```

Example: The following updates a label control when the mouse is moved over a node relating to a *TTable* object named *CustomersTbl*. Note that the mouse may not be necessarily over the active record in the dataset, so using the *TDataSet FieldByName* method would not work in this case.

```
procedure TForm1.fcDBTreeViewMouseMove(  
    TreeView: TfcDBCustomTreeView; Node: TfcDBTreeNode;  
    Shift: TShiftState;  
    X, Y: Integer);  
begin  
    if (Node<>Nil) and (Node.DataSet = CustomersTbl) then  
        Labell.Caption := Node.GetFieldValue('Company');  
end;
```


TfcDBTreeView



1stClass' advanced and sophisticated data-bound treeview can single-handedly navigate all the tables and queries in your master/detail relationships. You just instruct the component which datasources you wish for it to display as a tree using the *DataSourceLast* and *DataSourceFirst* properties, or using the *DataSources* property.

In reality, the TfcDBTreeView is a live window to your datasource's data. As a result, operations on the dataset are reflected in the TreeView. For instance, if you navigate the datasource, you are also navigating the tree. Likewise if you scroll the TreeView, you are also scrolling the dataset. Similarly if you apply a filter or set a range on the dataset, the TreeView instantly shows only the matching subset of records. To change the collating order of the nodes in the tree, you simply change your dataset's *Index* (if using a TTable), or change the SQL's order by clause (if using a TQuery). To search for a node in the tree, call the TDataSet *Locate* method to search for a record in the dataset.

Since the tree is live and buffered, it also allows for fast loading even when going against large tables.

The TfcDBTreeview can expand only one node per level at a time. If you expand or move to a sibling node, the previously expanded node in the same level will automatically collapse. This restriction is necessary since the tree is a live window using TDataSources, and as a result has access to only the detail records for the currently selected node.

Use the *DisplayFields* property to define the text displayed for each node's data.

Use the *MultiSelectAttributes* property to enable multi-selection within the tree.

Use the *OnCalcNodeAttributes* event to customize the painting of each node in the tree.

Use the *Header* property to associate a header control with a self-referencing tree. This header control defines the column information within the tree. See the *how-to* topics for information on setting up a self-referencing tree.

Ancestor

TWinControl

 TfcDBCUSTOMTreeView

 TfcDBTreeView

Required supporting components

TDataSource

Required property assignments

DataSourceLast or DataSources, DisplayFields

Added Properties

ActiveDataSet (Runtime only)

ActiveDataSet is the TDataSet corresponding to the currently selected node. Each node in the tree is associated with a dataset as defined by the *DataSourceLast*, *DataSourceFirst*, and *DataSources* properties. See also the method *MakeActiveDataSet* for a convenient way to change the active tree level based on a specified dataset.

Data Type: TDataSet

ActiveNode (Read only, Runtime only)

ActiveNode is the currently selected node.

Note: Internally this property is updated whenever the treeview is repainted. After this update, the previous *ActiveNode* value is no longer valid. Therefore you should never save this value into one of your own variables and later refer to it, as the memory for the previous *ActiveNode* would have been freed.

Data Type: TfcDBTreeNode

Canvas (Runtime only)

Provides access to the canvas. Use the *Canvas* property to paint to the canvas from the *OnCalcNodeAttributes* and *OnDrawText* event handlers.

Data Type: TCanvas

DataSourceFirst

Specifies the datasource associated with the tree's root nodes. If this property is unassigned, and the *DataSourceLast* property is assigned, then the tree will recursively traverse the master datasources to automatically compute this value. During traversal, the tree examines either the TTable.*MasterSource* property or the TQuery.*DataSource* property. If you are using a 3rd party engine which does not have a *DataSource* or a *MasterSource* property in its TDataSet, then you should instead assign the tree's *DataSources* property, and leave the *DataSourceFirst* and *DataSourceLast* properties unassigned.

Data Type: TDataSource

DataSourceLast

Specifies the datasource associated with the tree's terminal nodes. You must set either this property or the *DataSources* property for the tree to display any nodes.

Data Type: TDataSource

DataSources

Specifies all the datasources associated with the tree. You must set either this property or the *DataSourceLast* property for the tree to display any nodes. This property's format is a semi-colon delimited string. For instance to bind the tree to three datasources (DataSource1, DataSource2, DataSource3), you would assign the following property value...

```
DataSource1;DataSource2;DataSource3
```

The above value uses DataSource1 for the root nodes, DataSource2 for the child nodes of the root nodes, and so on.

If your datasources are on a different form than this component, then use the '.' notation such as

```
Form1.DataSource1;Form1.DataSource2;Form1.DataSource3
```

Data Type: String

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*.

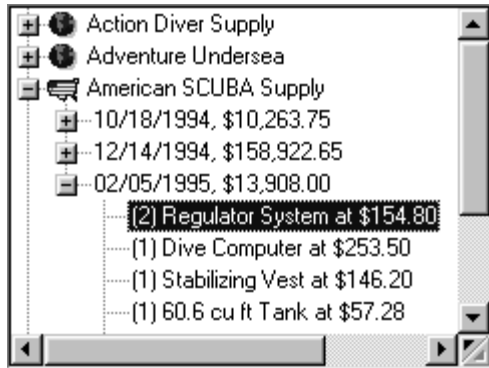
DisplayFields

This property defines the text and display format for each level in the tree. The property is a TStringList, with each string item corresponding to a display format for a given level. You should enclose field names in double quotes. The remaining characters are treated literally.

Example: Consider the following property assignment to *DisplayFields*.

```
"Company"  
"SaleDate", "AmountPaid"  
("Qty") "LookupPartDescription" at "LookupPartPrice"
```

This would result in the following display in the tree.



Note: If the *DisplayFields* is left blank for a given level of the tree, then the first field in the *TDataSet* is displayed.

Note: If the *Header* property is assigned, then this property is ignored as the display information then comes from the header definition.

Data Type: *TStringList*

Header

Assign the *Header* property if you have built a self-referencing data-bound tree, and you wish to associate a header control with the tree. For further information, see the *TfcTreeHeader* component, as well as the *TfcDBTreeView* how-to topics.

Data Type: *TfcTreeHeader*

Imager

Set this property to a *TfcImager* component to paint a background image in the *TfcDBTreeView*. The image can be painted as a tile or it can be stretched in the tree's window. See the *TfcImager DrawStyle* property.

Note: After attaching a *TfcImager* to the tree, you may wish to manipulate its properties in the IDE environment. You will need to click on the top-left corner of the treeview to have the object selected at design-time. Alternatively you can select the *TfcImager* from the object inspector.

Data Type: *TCustomImageList*

Images

Determines which image list is associated with the tree view. Use *Images* to provide a customized list of bitmaps that can be displayed to the left of a node's label. Individual nodes specify the image from this list that should appear by setting their *ImageIndex* property. Use the *OnCalcNodeAttributes* event to customize the *ImageIndex* dynamically based on the node or record information.

Data Type: TCustomImageList

InactiveFocusColor

Specifies the background color of the node that is selected when the treeview does *not* have focus. If the tree's *Options* | *dtvoHideSelection* property is *True* then this property is ignored when painting the selected node. If *MultiSelectAttributes* | *Enabled* is *True*, then this color is also used to paint the multi-selected nodes when the tree does not have focus.

Data Type: boolean

LastVisibleDataSet (Runtime only)

Returns the TDataSet associated with the deepest expanded nodes. Do not confuse this property with *ActiveDataSet*, which correlates to the TDataSet associated with the currently selected node.

Data Type: TDataSet

LevelIndent

Set this property to change the number of pixels used to indent each level in the tree. This property defaults to 21 pixels.

Data Type: Integer

LineColor

Set this property to change the color of the connecting lines in the tree.

Data Type: TColor

MultiSelectAttributes

Specifies the attributes for enabling and controlling multi-selection in the tree. This property contains the following sub-properties.

Data Type: TfcDBMultiSelectAttributes

AutoUnselect

When *True*, the tree will automatically unselect all previously selected nodes when the user clicks on a node without using the Ctrl key. In addition the clicked node is selected.

Data Type: boolean

Enabled

When *True*, the tree will automatically use Ctrl-Click to select/deselect a node. This provides a convenient way to perform multi-selection. To allow the user to multi-select with a checkbox set both the *Enabled* and *MultiSelectCheckbox* properties to *True*.

Data Type: boolean

MultiSelectCheckbox

When *True*, a checkbox is displayed in each node to allow the end-user a convenient way of selecting nodes. The space key will also select the node when this property is *True*.

Data Type: Boolean

MultiSelectLevel

Set this to the level you wish to enable multi-selection for. Defaults to 0, which indicates only the root nodes can be selected. If set to -1, then any node in any level can be selected.

Data Type: integer

MultiSelectList (Runtime only)

Reference this property to access the records that have been multi-selected by the end-user. This property is an array of `TfcMultiSelectItem`

```
TfcMultiSelectItem = class
  Bookmark: TBookmark;
  DataSet: TDataSet;
end;
```

Bookmark is a `TBookmark` associated with the selected record. *DataSet* is the dataset associated with the selected record. The order of this list is the order the selections were made. If you wish for the list sorted based on the dataset's order, then call the *SortMultiSelectList* method before iterating through the list.

Example: See how-to topic on iterating through the list of multi-selected records.

Data Type: Array of `TfcMultiSelectItem`

MultiSelectListCount (Runtime only)

This property contains the number of records selected by the user. See the *MultiSelectList* property to retrieve the actual records.

Data Type: Integer

Options

This property contains a set of boolean values that control the appearance and behavior of the tree

Data Type: Set of `TfcDBTreeViewOption`

Valid Values: `dtvoKeysScrollLevelOnly`, `dtvoAutoExpandOnDSScroll`, `dtvoExpandButtons3D`, `dtvoFlatCheckBoxes`, `dtvoHideSelection`, `dtvoRowSelect`, `dtvoShowNodeHint`, `dtvoShowButtons`, `dtvoShowLines`, `dtvoShowRoot`, `dtvoShowHorzScrollBar`, `dtvoShowVertScrollBar`, `dtvoHotTracking` (described below).

dtvoKeysScrollLevelOnly

This property controls the scrolling behavior when the user enters one of the following keys (Up, Down, PageUp, PageDown). When *True*, the tree will scroll only within the current level. When it reaches the first sibling node or the last sibling node, it will stop scrolling. When *False*, the tree will auto-collapse the parent node when the user scrolls before the first sibling node, or scrolls past the last sibling node. This property defaults to *True*.

dtvoAutoExpandOnDSScroll

Set to *True*, to force the record correlating with the scrolled datasource to be selected in the tree by expanding nodes if necessary. A datasource can be scrolled by another visual control such as a DBGrid or a DBNavigator.

When set to *False*, the tree will still synchronize with the datasource, but the tree will not auto-expand the tree. Thus the scrolled dataset node may not be visible in the tree in this case. Defaults to *True*.

dtvoExpandButtons3D

Set to *True* to display the expand and collapse buttons as three-dimensional buttons. Defaults to *False*.

dtvoFlatCheckBoxes

Set to *True* to display flat checkboxes. Defaults to *False*, which displays checkboxes in the tree-view as three-dimensional buttons.

dtvoHideSelection

This property controls how a treeview displays the selected node when it does not have the focus. Set to *True* to hide the selection when the tree does not have focus. If set to *False*, the tree displays the selected node in the color as defined by the *InactiveFocusColor* property. Defaults to *True*.

dtvoRowSelect

Set to *True* to highlight the entire row to the instead of just highlighting the text. If set to *False*, only the text is highlighted when a node is selected. Defaults to *False*.

dtvoShowNodeHint

Set to *True* to display a hint window when the text for the node will not fit in the tree's width constraints. Defaults to *True*.

dtvoShowButtons

Set to *True* to display the expand and collapse buttons in the tree. Defaults to *True*.

dtvoShowLines

Set to *True* to display the connecting lines in the tree. Defaults to *True*.

dtvoShowRoot

To show lines connecting top-level nodes to a single root, set the tree's *dtvoShowRoot* and *dtvoShowLines* properties to *True*.

dtvoShowHorzScrollBar

Set this property to *False* to disable the horizontal scrollbar from appearing in the tree.

dtvoShowVertScrollBar

Not implemented. This property currently has no effect. It is reserved for possible future use.

dtvoHotTracking

Specifies whether list items are highlighted when the mouse passes over them. Set *dtvoHotTracking* to *True* to provide visual feedback about which item is under the mouse. Set *dtvoHotTracking* to *False*, to provide no visual feedback about which item is under the mouse. To selectively control which nodes are hot-tracked see the *TfcDBTreeView* how-to topics at the end of this component's reference.

StateImages

Determines which image list to use for state images. Use *StateImages* to provide a set of bitmaps that reflect the state of tree view nodes. The state image appears as an additional image to the left of the item's icon.

Data Type: *TCustomImageList*

Added Events**OnCalcNodeAttributes**

This event allows you to change the node and painting canvas attributes before the *TreeView* paints the node. Use this event to change the font, background color, the node's text, and other node attributes.

TreeView: *TfcDBCustomeTreeView* *TreeView* associated with the node to be painted. If you wish to access the painting canvas to change the painting attributes of the node, refer to the *TreeView's Canvas* property.

Node: *TfcDBTreeNode* *Node* that is about to be painted

Example: The following code causes nodes with field *Country*='US' to paint with a blue font. The code also displays the first image index (from the imagelist pointed to by *StateImages*) for 'US', and the 2nd image index otherwise.

```

procedure TForm1.fcDBTreeView1CalcNodeAttributes (
    TreeView: TfcDBCustomTreeView;
    Node: TfcDBTreeNode);
begin
    if (Node.DataSet = CustomerTable) and
        (CustomerTable.FieldName('Country').asString = 'US'
    then
        begin
            TreeView.Canvas.Font.Color:= clBlue;
            Node.StateIndex:= 0;
        end
        else Node.StateIndex:= 1
    end;

```

OnCalcSectionAttributes

This event allows you to change a column's painting canvas attributes before the TreeView paints the node's column. This event is only applicable if you have assigned the *Header* property. Use this event to change the column's font, background color, and text display.

TreeView: TfcDBCustomTreeView *TreeView* associated with the column to be painted. If you wish to access the painting canvas to change the painting attributes of the column, refer to the *TreeView's Canvas* property.

Node: TfcDBTreeNode *Node* that is about to be painted

Section: TfcTreeHeaderSection Header section associated with the column

var DisplayText: string Text to be painted in the column

OnChange

This event allows you to perform some custom action after the active node is changed in the tree. For instance, you may wish to update a label in your form that displays all the text of the active node's parents. The active node can change by the user scrolling the tree, or by navigating the datasources with some other control.

TreeView: TfcDBCustomTreeView *TreeView* associated with the event

Node: TfcDBTreeNode *Node* that is selected

Example: The following code updates a label (TreeStateLabel), when the active node changes.

```

procedure TDMTreeViewForm.fcDBTreeView1Change (
    TreeView: TfcDBCustomTreeView;

```

```

    Node: TfcDBTreeNode);
var s: string;
    tempNode: TfcDBTreeNode;
begin
    { Compute label to indicate tree state }
    s:= '';
    tempNode:= node;
    repeat
        s:= tempNode.Text + #13 + s;
        tempNode:= tempNode.parent;
    until tempNode=nil;
    TreeStateLabel.caption:= s;
end;

```

OnDbClick

See *OnMouseDown* event.

OnDrawSection

This event allows you to change the default text painting of the node. You will rarely need to use this event, as the *OnCalcNodeAttributes* is the preferred event to use to change a node's font, text, background, and other attributes. This event is only applicable if you have assigned the *Header* property. Use this event if you wish to override the actual painting of one or all of the columns of the node being drawn.

The parameters for this event are as follows:

<i>TreeView</i> : TfcDBCUSTOMTreeView	<i>TreeView</i> associated with the node. If you wish to access the painting canvas, refer to the <i>TreeView's Canvas</i> property.
<i>Node</i> : TfcDBTreeNode	<i>Node</i> that is about to be painted.
<i>Section</i> : TfcTreeHeaderSection	Header section associated with the column to be painted.
<i>ARect</i> : TRect	Default rectangle where the text is to be painted.
<i>S</i> : String	Text to be painted in the column.
<i>DefaultDrawing</i> : boolean	Specifies whether the control should paint the item.

OnDrawText

This event allows you to change the default text painting of the node. You will rarely need to use this event, as the *OnCalcNodeAttributes* is the preferred event to use to change a node's font, text, background, and other attributes. You will only need this event if you wish to override the actual painting of the node's text.

The parameters for this event are as follows:

<i>TreeView</i> : TfcDBCustomeTreeView	<i>TreeView</i> associated with the node. If you wish to access the painting canvas, refer to the <i>TreeView's Canvas</i> property.
<i>Node</i> : TfcDBTreeNode	<i>Node</i> that is about to be painted
<i>ARect</i> : TRect	Default rectangle where the text is to be painted.
<i>DefaultDrawing</i> : boolean	Specifies whether the control should paint the item.

Example: The following example underlines the character following ampersands in the node's text by using the *TCanvas DrawText* method. Note that the code below does NOT make the node accessible through an accelerator key. If you desire this behavior you would need to write the code to trap the keys using events such as the *OnKeyDown* event.

```

Procedure TForm1.fcDBTreeView1DrawText (
  TreeView: TfcDBCustomeTreeView; Node: TfcDBTreeNode;
  ARect: TRect; var DefaultDrawing: boolean);
begin
  { Underlines characters following ampersand }
  TreeView.Canvas.DrawText(Node.Text, ARect, 0);
  if Node.selected then begin { Draw focus rect }
    InflateRect(ARect, 1, 1);
    ARect.Left:= ARect.Left - 1;
    TreeView.Canvas.DrawFocusRect(ARect);
  end;
  DefaultDrawing := False;
end;

```

OnMouseDown, OnMouseUp, OnDbClick

Use this event to perform some custom action when the mouse is pressed, released, or double-clicked over the *TreeView*. The parameters for this event are as follows:

<i>TreeView</i> : TfcDBCustomeTreeView	<i>TreeView</i> associated with the event
<i>Node</i> : TfcDBTreeNode	<i>Node</i> that the mouse is over at the time of the event.
<i>Button</i> : TMouseButton	Distinguishes which mouse button generated the mouse event. Can be <i>mbLeft</i> , <i>mbRight</i> , or <i>mbMiddle</i> .
<i>Shift</i> : TShiftState	Use the <i>Shift</i> parameter to respond to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations.

X, Y: Integer X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the *TreeView*.

OnMouseMove

Use this event to perform some custom action when the mouse moves over a node.

The parameters for this event are as follows:

TreeView: *TfcDBCustTreeView* *TreeView* associated with the event

Node: *TfcDBTreeNode* *Node* that the mouse is over

Shift: *TShiftState* Use the *Shift* parameter to respond to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations.

X, Y: Integer X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the *TreeView*.

OnUserCollapse

Use this event to perform some custom action after the user has collapsed a node by pressing on the collapse button, using the left-arrow key, or by pressing the icon under the vertical scrollbar to collapse a node.

The parameters for this event are as follows:

TreeView: *TfcDBCustTreeView* *TreeView* associated with the event

Node: *TfcDBTreeNode* *Node* that has collapsed

OnUserExpand

Use this event to perform some custom action after the user has expanded a node by pressing on the expand button, using the right-arrow key, or by pressing the icon under the vertical scrollbar to expand a node.

The parameters for this event are as follows:

TreeView: *TfcDBCustTreeView* *TreeView* associated with the event

Node: *TfcDBTreeNode* *Node* that has been expanded

Added Methods

Collapse

Call this method to collapse a node so that its children are hidden.

Procedure Collapse(*Node*: *TfcDBTreeNode*); **virtual**;

Expand

Call this method to expand a node so that its children are displayed.

```
Procedure Expand(Node: TfcDBTreeNode); virtual;
```

GetHitTestInfoAt

GetHitTestInfoAt returns information about the location of a point relative to the client area of the tree view control.

```
function GetHitTestInfoAt(X, Y: Integer): TfcTreeHitTests;
```

Call *GetHitTestInfoAt* to determine what portion of the tree view, if any, sits under the point specified by the *X* and *Y* parameters. For example, use *GetHitTestInfoAt* to provide feedback about how to expand or collapse nodes when the mouse is over the relevant portions of the tree view.

GetHitTestInfoAt returns a TfcTreeHitTests type. The possible return values are:

<u>Value</u>	<u>Location of (X,Y)</u>
fchtOnButton	On the button (expand/collapse) associated with a node
fchtdOnActiveNode	On the active tree node
fchtdOnImageIcon	On the image icon associated with a node
fchtdOnText	On the label (text) associated with a node
fchtdOnStateIcon	On the state icon for a node

GetNodeAt

GetNodeAt returns the node that is found at the specified position.

```
Function GetNodeAt(X,Y: integer): TfcDBTreeNode;
```

Call *GetNodeAt* to access the node at the position specified by the *X* and *Y* parameters. *X* and *Y* specify the position in pixels relative to the top left corner of the tree view. If there is no node at the location, *GetNodeAt* returns nil.

InvalidateClient

Call this method to invalidate the tree's client-area. The buttons on the bottom-right are not repainted. Call the inherited *Invalidate* method to have the entire tree repaint.

```
Procedure InvalidateClient; virtual;
```

InvalidateNode

Call this method to invalidate the current node in the tree.

```
Procedure InvalidateNode(Node: TfcDBTreeNode);
```

InvalidateRow

Call this method to invalidate a row in the tree. *Row* is the offset from the top of the tree, where the top node has *row=0*.

```
Procedure InvalidateRow (Row: integer);
```

IsSelectedRecord

When *MultiSelectAttributes | Enabled* is *True*, use this method to test if the active node has been multi-selected.

```
Function IsSelectedRecord: boolean;
```

MakeActiveDataSet

Call this method to have a node associated with a specific *TDataSet* be made the active node. Set *Collapse* to *True* to force the children to be hidden after changing the active node. If *Collapse* is *False*, the children are still displayed if the new active node is the parent of the displayed child nodes.

```
Procedure MakeActiveDataSet (DataSet: TDataSet;  
Collapse: boolean);
```

MoveTo

Use this method within the context of one of the tree events to cause the node to become the active node.

```
Procedure MoveTo (Node: TfcDBTreeNode);
```

SelectRecord

When *MultiSelectAttributes | Enabled* is *True*, use this method to select the active node in a tree. If both *MultiSelectAttributes | Enabled* and *MultiSelectAttributes | MultiSelectCheckbox* are enabled, then the checkbox becomes checked.

```
Procedure SelectRecord; virtual;
```

SortMultiSelectList

Call this method to sort the multi-selected nodes. The nodes are sorted based on the return value of the dataset's *CompareBookmark* function. The result of calling this method will leave the root nodes at the top of the list, followed by all the selected nodes at level 1, and so on.

UnselectAll

Call this method to unselect all previously multi-selected records. See also the *MultiSelectAttributes* property.

```
procedure UnselectAll; virtual;
```

UnselectRecord

When *MultiSelectAttributes | Enabled* is *True*, use this method to unselect the active node in a tree

```
procedure UnselectRecord; virtual;
```

How To

How to use the TfcDBTreeView to display a self-referencing tree from a single table.

Since the TfcDBTreeView provides a generic way to display one or more datasources, you can use it to display a self-referencing tree. The steps to accomplish this are described below and an implementation of this can be found in the demo form `\demos\dbtreeview\DBSelfTree.pas`.

1. **Retrieving the root nodes:** Use a dataset component (such as a TQuery), to retrieve all the root nodes. In this example the data comes from the table `FirstClass:fcmsg.db`. Then assign its SQL property as follows.

```
Select msgid,rootid,parentid,  
       fcmsg."date",subject,fcmsg."name" from fcmsg  
where fcmsg."parentid" is null
```

Now set the dataset's *Active* property to True, and its name property to *RootNodesQuery*. After this drop in a TDataSource component and give it the name *RootNodesDataSource*, and then set its DataSet property to *RootNodesQuery*.

2. **Retrieving the children of a node:** Repeat the steps above with another TQuery component, and this time set its SQL property to the following:

```
Select * from fcmsg  
where fcmsg."parentid"=:msgid
```

Now set the dataset's *Active* property to True, and its name property to *ChildNodesQuery*. After this drop in a TDataSource component, and name it *ChildNodesDataSource*, and then set its DataSet property to *ChildNodesQuery*.

3. **Viewing and editing the active node:** Now drop a 3rd dataset component (such as a TTable or TQuery), and assign its name property to *ViewTable*. This dataset component will be used to display and edit the actual node in the tree. If using a TTable component, you can set its *tablename* property to *fcmsg.db*. After this drop another TDataSource component and set its dataset property to *ViewTable*. Use this datasource to view and edit any data for the currently selected node in the TfcDBTreeView. To keep the ViewTable in sync with the currently selected node of the TfcDBTreeView, you will need to put the following code in the tree's OnChange event.

```
ViewTable.locate('MsgID',  
                node.dataset.fieldbyname('MsgID').asstring, []);
```

4. **Hooking up the Tree to the datasources:** Set the DataSources property of the TfcDBTreeView to *RootNodesDataSource;ChildNodesDataSource*

To allow the dbtreeview to recursively expand, you will need to create additional datasources after the user expands a node. This can be handled with the following

specific code in the OnUserExpand event. LastDS is a global Tdatasource that should be defined in your interface section.

```

procedure TSelfDBForm.fcDBTreeView1UserExpand(
    TreeView: TfcDBCUSTOMTreeView;
    Node: TfcDBTreeNode);
var childquery: TQuery;
    childdatasource: TDataSource;
begin
    if (node.level+1<fcdbtreeview1.displayfields.count) then exit;

    { Dynamically create new detail parameterized query }
    childdatasource:= TDataSource.create(self);
    childdatasource.name:= 'ChildDataSource' + inttostr(node.level+1);
    childquery:= TQuery.create(self);
    childdatasource.dataset:= childquery;
    with childquery do begin
        childquery.sql.assign(ChildNodesQuery.sql);
        childquery.databasesname:= ChildNodesQuery.databasesname;
        if lastds=nil then childquery.datasource:= ChildNodesDataSource
        else childquery.datasource:= lastDS;
        childquery.active:=true;
    end;
    with (TreeView as TfcDBTreeView) do begin
        DataSources:= DataSources + ';' + ChildDataSource.name;
        displayfields.add(displayfields[displayfields.count-1]);
        lastDS:= Childdatasource;
    end
end;

```

5. **Setting up the columns:** To display the field information as columns, drop a TfcTreeHeader control and click on its Sections property to configure the columns. Then set the Header property of the TfcDBTreeView control to this header control.
6. **Hiding an expand button when it has no children:** To disable the expand button if there are no child nodes for a given node, you will need to drop another dataset component that you can lookup to determine if there are any children. For instance, drop a TTable component and set its tablename property to fmsg.db and its IndexName property to ParentID. Then attach the following code in the OnCalcNodeAttributes of the TfcDBTreeView.

```

node.haschildren:= IndexLookupTable.Locate('ParentID',
    node.dataset.fieldbyname('msgid').asstring, []);

```

How to control which specific nodes are displayed as a hot-track

You can enable automatic hot-tracking of tree nodes by setting the *property Options | dtvoHotTracking* to *True*. However this will hot-track each and every node, which may not be appropriate for your tree. To selectively hot-track nodes, use the *OnCalcNodeAttributes* event, and set the *Node.Hot* property to *False* for nodes that should not be hot-tracked. The following code attached to the *OnCalcNodeAttributes* event will hot-track the root nodes in the tree, but not other nodes.

```

if Node.Hot and (Node.Level>0) then Node.Hot:= False;

```


How to disable the expand icon (+) for specific nodes with no children

The tree automatically displays the expand icon for all nodes except the last datasource (specified by *DataSourceLast* or the last datasource specified in the *DataSources* property). However in some cases you may wish to prevent the display of the expand button for other nodes as well. This may be the case when those other nodes have no children. This can be accomplished through use of Delphi/Builder's lookupfields mechanism in combination with using the *OnCalcNodeAttributes* event.

For example, suppose your tree is displaying records from a customer table as the root nodes, and the orders for a customer as the child nodes. You wish to only display the expand icon for the customer nodes that have orders. This can be accomplished by the following steps.

1. Drop a new TTable/TQuery component into your form and associate it with the Orders table. Do not use the existing TTable/Query that the tree is displaying as lookup fields require their own TDataSet.
2. Add a new lookup field (name it 'LookupOrders' for step 3 below) to your customers table, and have it lookup the dataset in step 1. See your Delphi/Builder documentation for the steps on creating a lookup field.
3. Use the *OnCalcNodeAttributes* event to set the *Node.HasChildren* property if the lookupfield is not null. A null lookupfield indicates that the node has no children.

```
{Use lookupfield to decide if expand icon(+) should be visible}
if Node.DataSet = CustomerTable then
    Node.HasChildren:=
        Not Node.dataset.fieldbyname('LookupOrders').isNull;
```

How to change the color of a node based on its field information

See example documented under the *OnCalcNodeAttributes* event.

How to iterate through the list of selected records

The following code iterates through the list of selected records, and displays the 'Company' field from the related dataset.

```
var i: integer;
begin
    fcDBTreeView1.SortMultiSelectList;
    for i:= 0 to fcDBTreeView1.MultiSelectListCount-1 do begin
        with fcDBTreeView1.MultiSelectList[i] do begin
            DataSet.GotoBookmark(Bookmark);
            ShowMessage(DataSet.FieldName('Company').asString);
        end
    end
end
```

Note: You may also wish to call the *DisableControls* method of the dataset before iterating through the *MultiSelectList*. This would prevent controls tied to this dataset

from updating during the iteration. If you call *DisableControls*, make sure you call *EnableControls* to re-enable the dataset.

How to use a TPopupMenu to associate one or more actions action on the right-clicked node.

You can set the *PopupMenu* property of the tree to associate a popup-menu when the tree is right-clicked. To perform on action on the selected node from within your *PopupMenu* item's *OnClick* code, you can refer to the tree's *ActiveNode* property. For instance you could use the following code in your *TMenuItem OnClick* event to insert a record before the right-clicked node.

```
fcDBTreeView1.ActiveNode.Dataset.Insert;
```

How to prevent the text from being highlighted for multi-selected nodes.

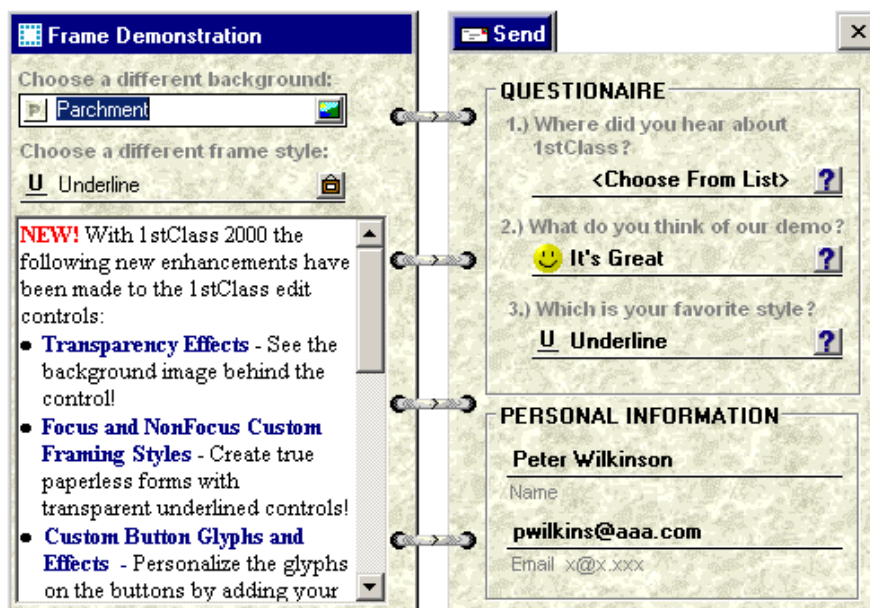
When using *MultiSelectAttributes* | *MultiSelectCheckbox*, the end-user can tell which nodes are selected by observing the checkbox. Thus there is no absolute need to also highlight the text. If you wish to display the text normally (non-highlighted) for multi-selected nodes but not the active node, then use the following code in your *OnCalcNodeAttributes* event.

```
if node.multiselect and (not node.selected) then
begin
  treeview.Canvas.Brush.Color:= clNone;
  treeview.Canvas.Font.Color:= clBlack;
end
```

TfcEditFrame (Class)



TfcEditFrame is a supporting class for many 1stClass components. 1stClass 3000 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 \1stclass3000\demos\framing directory.



Form displayed as a check, using 1stClass's transparent edit controls, custom framing, and custom button effects.

Components that support custom framing and transparency

The following 1stClass components support custom framing, transparency, and button effects (where applicable): TfcTreeCombo, TfcFontCombo, TfcColorCombo, TfcCalcEdit, TfcGroupBox, and TfcPanel.

Key properties and events for custom framing support

The following properties are in 1stClass 3000 to support the custom framing, transparency, and special button effects. Frame is a property available to all the 1stClass edit controls and some other controls. The following details each sub-property of Frame.

Properties

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, 1stClass 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 1stClass 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 1stClass 3000, 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.

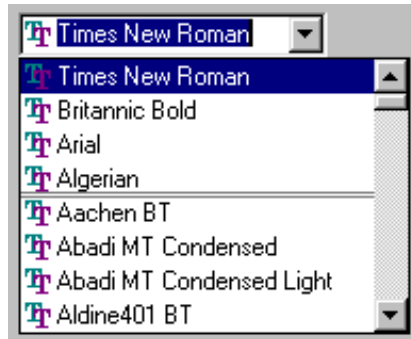
Restrictions: The background must be painted by a non-windows control (not derived from *TWinControl*), such as a Delphi *TImage* or the *TfcImager* (from Woll2Woll's 1stClass product). There may be some painting side effects when using a *TWinControl* to paint the background. You should only set this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

TfcFontCombo



The TfcFontCombo provides a way for the user to select a font from a drop-down list. In addition it saves and displays the most recently selected fonts at the top of the drop-down list.

Set the *MaxMRU* to a value greater than 0 to have the control save and display the most recently selected fonts. These fonts are displayed at the top of the drop-down list.



Screen shot of MRU

Set *ShowHintFont* to *True* to give the user a sample preview of the font that the mouse cursor is over. If you also set the *ImmediateHints* font property to *True*, then the hint window is immediately displayed (as opposed to a delay) as soon as the mouse moves over a font.

Internally the TfcFontCombo uses a *TfcTreeView* to display its drop-down list items. If you wish to change the display options of the drop-down tree, the set the *TreeOptions* property. However since the drop-down list for fonts is non-hierarchical not all options will apply.

Ancestor

```
TCustomEdit
  TfcCustomCombo
    TfcCustomTreeCombo
      TfcCustomFontCombo
        TfcFontCombo
```

Added Properties

AutoSelect, AutoSize, BorderStyle, CharCase

These properties are equivalent to the properties of the same name found in *TEdit*. See the Delphi / C++ Builder docs under *TEdit* for more information on these properties.

AllowClearKey

When the 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. The default value is *False*.

Data Type: boolean

Controller

See InfoPower TwwController property

DropDownCount

The *DropDownCount* property determines how many entries will appear in the dropdown control.

Data Type: Integer

DropDownWidth

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

Data Type: Integer

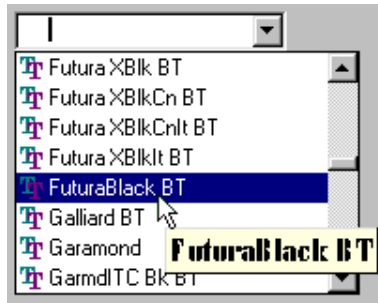
Frame

See TfcEditFrame for more information on this property.

Data Type: TfcEditFrame

ImmediateHints

Use this property when the *ShowFontHint* property is *True*. If *ImmediateHints* is *True*, the font preview is displayed immediately when the mouse moves over a new font. If *ImmediateHints* is *False*, then the font hint is not displayed until after the standard hint delay time.



FontCombo with FontHint Displayed

Data Type: boolean

MaxMRU

This property specifies the maximum number of items that will be added to the most recently used section of the drop down tree view. If this property is -1 then MRU functionality is disabled.

Data Type: integer

PreLoad

When this property is set, the fonts are loaded into the combo upon creation. However, when this property is false, the fonts are loaded when you drop down the list, or when you type in a character when *ShowMatchText* is true.

Data Type: boolean

RecentFonts

A list of fonts that appear at the top of the drop-down list. This property is automatically managed if the *MaxMRU* property contains a value other than -1.

Data Type: TStringList

SelectedFont (Runtime and ReadOnly)

Reference this property to return the name of the currently selected font

Data Type: String

ShowFontHint

Set to *True* to give the user a sample preview of the font that the mouse cursor is over. After the drop-down list is displayed, the user can move the mouse over any font in the drop-down list and the control will display a hint window containing a sample of the font. See also the *ImmediateHints* property.

Data Type: boolean

ShowMatchText

When this property is set to *True*, the FontCombo will perform ‘Quicken’ style incremental searching. As the user enters text, the control will simultaneously search and display the matching font in the control. The default value is *True*.

Data Type: boolean

Sorted

Setting this property to *True* will sort the list alphabetically. Once the drop-down items are sorted, the original order is lost. That is, setting the *Sorted* property back to *False* will not restore the original order of items. The default value is *True*.

Data Type: boolean

Style

This property determines the style of the FontCombo. 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.

Data Type: TfcComboStyle

Valid Values: fcCombo.csDropDown, fcCombo.csDropDownList

TreeOptions

If you wish to change the display options of the drop-down tree, the set the TreeOptions property. However since the drop-down tree for fonts is non-hierarchical not all display options will apply. See the *Options* property of the TfcTreeView control.

Data Type: TfcTreeViewOptions

Valid Values: tvoExpandOnDbIClk, tvoExpandButtons3D, tvoFlatCheckBoxes, tvoHideSelection, tvoRowSelect, tvoShowButtons, tvoShowLines, tvoShowRoot, tvoHotTrack, tvoAutoURL, tvoToolTips, tvoEditText, or tvo3StateCheckbox

TreeView (Runtime only)

Use this runtime only property to access the dropdown treeview control.

Data Type: TfcTreeView

Added Events

OnAddFont

Occurs immediately before adding a font to the font combo. `Accept` is initially true setting it to false will prevent the font from being added to the font combo. The parameters for this event are as follows:

FontCombo: TfcFontCombo	The <i>TfcFontCombo</i> associated with the event
FontName: string	The name of the font that the <i>TfcFontCombo</i> is adding.
FontType: TfcComboFontType	The type of font being added. Can be either <i>ftFontPrinter</i> , <i>ftFontTrueType</i> , or <i>ftFontOther</i> .
EnumLogFont: TEnumLogFont	Consult your Windows API documentation for information on the structure of <i>EnumLogFont</i>
NewTextMetric: TNewTextMetric	Consult your Windows API documentation for the structure of <i>NewTextMetric</i>
var Accept: boolean	Set to <i>False</i> to not add the font into the list.

OnGenerateFontHint

Occurs immediately before displaying a hint for a particular font. Customization on the hint text and font can occur here. Only occurs when the *ShowFontHint* property is true.

<i>FontCombo</i> : TfcFontCombo	The <i>TfcFontCombo</i> associated with the event
<i>FontName</i> : string	The name of the font that a hint is about to be displayed for.
<i>Font</i> : TFont	TFont correlating with the font associated with the hint
var <i>Hint</i> : string	The text to be displayed in the hint window. Set this to customize the text of the hint.

OnSelectionChange

See the TfcTreeCombo *OnSelectionChange* event

Added Methods

CloseUp, DropDown

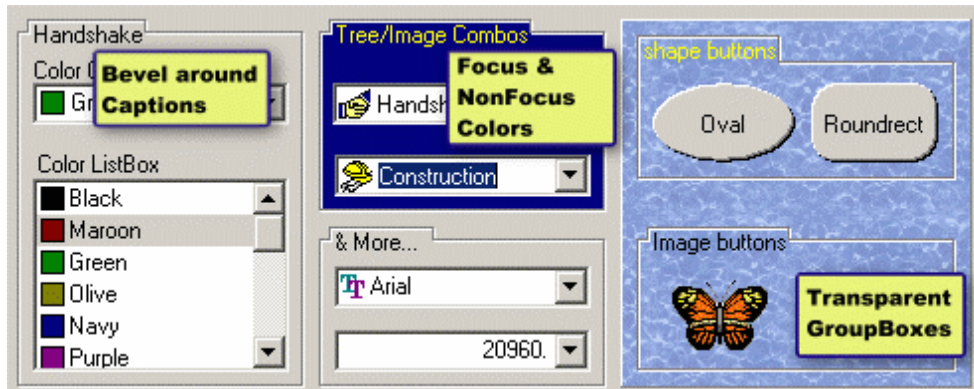
See the corresponding methods declared in the TfcTreeCombo component

Reload

Reload the list of fonts from the system

TfcGroupBox

The TfcGroupBox component is similar to the native TGroupBox component, with additional options for run-time transparency and custom framing.



TfcGroupBox controls

Ancestor

TCustomGroupBox
TfcCustomGroupBox
TfcGroupBox

Added Properties

BorderAroundLabel

When this property is True, then the framing will go around the groupbox text creating sort of a tab look. When this is false, then a normal groupbox look where the text overlaps the upper border applies.

Data Type: Boolean

CaptionIndent

This property determines the indent of the caption relative to the side of the groupbox control. The minimum value is 3.

Data Type: Integer

FullBorder

When `BorderAroundLabel` is `True`, then this property will make the caption of the groupbox taller otherwise the caption will be displayed half in the groupbox and half out of it like a standard groupbox control. When `BorderAroundLabel` is `False`, then `FullBorder` will cause the caption to be displayed above the groupbox transparently.

Data Type: Boolean

Frame

Currently only the `NonFocusFontColor` and `NonFocusColor` properties are supported when `Frame.Enabled` is set to `True`. The other Frame properties are left in for possible future enhancements.

Set `Transparent` to `False` if you wish to have the groupboxes dynamically change colors based on the controls focus and use the `Color`, `Frame.NonFocusColor`, `Font.Color` and `Frame.NonFocusFontColor` property settings accordingly.

See `TfcEditFrame` for more information on this property.

Data Type: `TfcEditFrame`

Transparent

Controls whether or not to display the `TfcPanel` transparently.

Data Type: Boolean

Tips

- If you wish all of the controls in the `TfcGroupBox` to have their font change when the group box's font changes, then you may wish to use the `OnEnter` and `OnExit` events to set the Groupbox Font's Color property and set all of the child's parentfont settings to `True`. Otherwise if you wish the child controls to always have the same font color property, then you can set them individually with their parentfont property set to `False`. Then you can just set the `Frame.NonFocusFontColor` and `Color` properties of the `GroupBox` to handle the coloring of the `GroupBox`.

NOTE: Focus Colors and NonFocusColors don't apply unless the `Transparent` property is set to `False`.

TfcImageBtn



The TfcImageBtn, in addition to providing the functionality found in TButton, TBitBtn, and TSpeedButton, enhances the native buttons in many important ways. Most significantly, the button takes its appearance *and* shape from an image. Therefore, the number of different shapes it can have are virtually limitless.



TfcImageBtn controls

Some of its capabilities include:

- A separate Up and Down image is possible with different shapes for each. Among its other uses, this allows the button to function as a switch control.
- The *Highlight* shade style alters your image to make even a plain flat image appear 3D both in its pressed and non-pressed state.
- The *DitherStyle* property allows the dithering that occurs when it is *down* and part of a group of buttons to display in a number of ways—from blending the image with a specified color, to the standard windows dithering method.
- Regardless of the image you have used to define the button, its color can still be manipulated using the color property.

IstClass provides the following design-time aids when configuring your image button.

- Double-Clicking on the control at design-time will cause either one of two things to occur. If the *Image* property is *empty*, then the Picture Selection Dialog Box will appear. Otherwise, if an image is already defined, the *OnClick* handler will be, if necessary, created, and then shown.
- When you right-click the button at design time, you can access the following actions from the pop-up menu.

Set Shade Colors

Select this item to modify the *ShadeColors* property to be appropriate for the *pixel* you right-clicked on to bring up the pop-up menu. For example, if your image contains an area of yellow pixels, right-click on one of those pixels and then choose this item.

Size To Default

Sets the *size* of the button to match the size of the bitmap stored in the *Image* property.

Ancestor

TWinControl
TfcCustomBitBtn
TfcCustomImageBtn
TfcImageBtn

Added Properties

Action, AllowAllUp, Anchors, Cancel, Constraints, Default, Down, Glyph, GroupIndex, Kind, Layout, Margin, ModalResult, Style, and Spacing

These properties are equivalent to the properties of the same name found in *TSpeedButton*. See the Delphi / C++ Builder docs under *TSpeedButton* for more information on these properties.

Color

Setting this property to a value other than *clNone* will result in the general color of the image to reflect the new value of *Color*. This feature works best with grayscale images.

Data Type: TColor

DitherColor

The value of this property determines what color the button blends with when it is down and part of a group. (i.e., the *GroupIndex* property is set to a value greater than 0) This property defaults to *clWhite*.

Data Type: TColor

DitherStyle

This property determines how dithering is applied when the button is part of a group. (i.e., the *GroupIndex* property is set to a value greater than 0)

Data Type: TfcDitherStyle

Valid Values: dsDither, dsBlendDither, dsFill

dsDither

This provides for the standard dithering seen with other controls like the *TSpeedButton*. The button's color will dither with the value of the *DitherColor*

property. Left at its defaults, the dithered look will be an alternating array of *clSilver* and *clWhite* pixels.

dsBlendDither

This will cause the button's *Image* to be dithered with the color specified by the *DitherColor* property. Every other pixel of the image will be the *DitherColor*.

dsFill

This will simply fill the region of the button with the *DitherColor*.

ExtImage

Provides for the delegation of the actual image that the button uses to a separate component. Use this in place of the *Image* property. This can be useful for a number of reasons including: lowering DFM size, lowering resource consumption, ease of use, and much more. *ExtImage* corresponds to the *Image* property and will supercede it when set.

Data Type: TComponent

Valid Values: TfcImager, TfcImageBtn

ExtImageDown

This property is functionally equivalent to the *ExtImage* in all respects except that it applies to the *ImageDown* property instead of the *Image* property.

Data Type: TComponent

Valid Values: TfcImager, TfcImageBtn

Image

This property controls both the shape of the button and the background image of the button. By default, the upper-left hand corner pixel of the image is taken to be the transparent color. Any pixel of this color will not be part of the button and will therefore be wholly transparent.

Data Type: TfcBitmap

ImageDown

Set this property if you want the down state of the image to have a different appearance than the up state. The shape of the down image does not necessarily have to be the same as that of *Image*.

Data Type: TfcBitmap

NumGlyphs

This property is similar to the property of the same name for TSpeedButton. It determines how many images the *Glyph* property can store up to a maximum of four. Images within the *Glyph* property are arranged horizontally. (i.e., A glyph with four

images has a dimension of Height x 4 * Width) The left-most image is the standard glyph that is displayed when the mouse is in an idle state. The next glyph to the right is displayed when the button is disabled. The third glyph is displayed when the button is down/clicked. The last glyph is displayed when the mouse is over the control (hot-tracking).

Data Type: TNumGlyphs

Valid Values: 1..4

Offsets

This set of properties allows you to manipulate how various parts of the button are painted.

GlyphX, GlyphY

Set these properties to displace the glyph from its default position. Positive and negative values are allowed. Positive *GlyphX* displaces the glyph towards the left, a negative value for *GlyphX* displaces the glyph towards the right. Positive and negative values for *GlyphY* work the same, but *Up* and *Down* respectively.

Data Type: Integer

TextX, TextY

Similar in functionality to *GlyphX* and *GlyphY*. However, these properties affect the displacement of the caption of the button.

Data Type: Integer

TextDownX, TextDownY

Similar in functionality to *TextX* and *TextY*, these properties affect the displacement of the caption when the button is pressed.

Data Type: Integer

Options

This property is a set of boolean flags that control various aspects of the button.

Data Type: TfcButtonOptions

Valid Values: boFocusable, boToggleOnUp, boFocusRect, boAutoBold

boFocusable

When set, the button is a valid control for receiving focus. However, regardless of the value of this property, arrow keys will always be able to select the button.

boToggleOnUp

When set, the button will not toggle its Up/Down state until the user has released the mouse button. This property is useful when the button is part of a group. (i.e., the *GroupIndex* property is set to a value greater than 0) It is particularly useful

when you have a button with different up and down images and you want to more closely replicate the behavior of a "switch" style button.

boFocusRect

When set, the button will have a standard focus rectangle (A dotted line rectangle) around the button caption. This property is only used when the *boFocusable* property is *True*.

boAutoBold

When set, the button caption's font will go to bold when pressed down. This property is only used when the button is part of a group. (i.e., the *GroupIndex* property is set to a value greater than 0)

ParentClipping

When *True* (the default), the button's container (form, panel, etc.) will clip the region where the button is located. This can occasionally cause the form to appear poorly until the buttons are finished loading. Therefore, set this property to *False* to obtain a better appearance when painting the form/container.

Data Type: boolean

RespectPalette

When the underlying *Image* and/or *ImageDown* property contains a bitmap with a palette (256 Colors or less) then setting this property to *True* can significantly improve appearance on systems that display 256 colors or less. However, note that this can potentially degrade performance.

Data Type: boolean

ShadeColors

This property contains a set of sub-properties that control the colors used to paint the highlight and shade colors of the button.

BtnHighlight

The color of what is typically the farthest displayed pixel to the upper-left of a button. Defaults to *clBtnHighlight* (Which, in turn, defaults to *clWhite* on most systems)

Btn3DLight

The color one pixel farther in from the pixels controlled by the *BtnHighlight* property.

BtnBlack

The color of what is typically the color the farthest to the lower right of a button. Defaults to *clBlack*.

BtnShadow

The color one pixel farther in from the pixels controlled by the *BtnBlack* property.

BtnFocus

The color of the outline that appears around a button if it obtains focus and has the *Options* | *boFocusable* property set to *True*.

Shadow

The fill color that is used when the button's *ShadeStyle* is *fbsNormal* or *fbsRaised*. When *ShadeStyle* is set to *fbsNormal*, the fill color is the color that appears on the upper-right when the button is pressed. When *ShadeStyle* is set to *fbsRaised*, the fill color is the color that appears on the lower right when the button is not pressed.

ShadeStyle

This property determines how the button shades the image. Note that this property is only respected if the *ImageDown* property is **empty**.

fbsNormal

When *ShadeStyle* is set to *fbsNormal*, the button appears exactly as the image in the *Image* property dictates when the button is not pressed. When the button is pressed, however, the image is offset by two pixels, and the resultant gap is filled in with the color defined in the *ShadeColors.Shadow* property.

fbsRaised

When *ShadeStyle* is set to *fbsRaised*, the button appears elevated from the form and its shadow is drawn in the color defined in the *ShadeColors.Shadow* property. When the button is depressed, the button appears exactly as the image in the *Image* property.

fbsHighlight

Setting *ShadeStyle* to this value causes 3D lines to be applied to the edges of the *Image*. The colors of the 3D lines are determined by the *ShadeColors* property.

fbsFlat

This setting is similar to *fbsHighlight*, but the 3D lines are only applied when the mouse is over the button.

TextOptions

Please see the documentation for *TfcText*.

TransparentColor

This property determines what color in *Image* will be taken to be transparent. If it is set to *clNone* (the Default), the upper-left hand pixel will be taken to be the transparent color. If it is set to *clNullColor*, the button will not have any transparent elements and will therefore be a standard rectangle. Any other value for this property will cause that specific color to be transparent, and therefore not be contained within the Button's region.

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.

OnSelChange

Occurs when the *Down* property changes. Although similar to the *OnClick* event, this event differs in that it is also fired when the user clicks on a different button assigned to the same *GroupIndex*.

Added Methods

SizeToDefault

Call this method when you want to make the size of the button match the size of the bitmap stored in the *Image* property.

```
procedure SizeToDefault; override;
```

UpdateShadeColors

This method will modify the *ShadeColors* property to be appropriate for *Color*.

```
procedure UpdateShadeColors(Color: TColor); override;
```

How To

Respond to when the mouse enters and leaves the button

Attach handlers to the *OnMouseEnter* and *OnMouseLeave* events. In those events, you can set the *Color* properties of the *Font*, *Button*, *TextOptions*, etc. To easily update *ShadeColors* after setting the *Color* property, call the *UpdateShadeColors* method.

Have multi-line captions

When editing the *Caption* property click on the ellipsis button to bring up a standard stringlist editor.

Make the buttons more resource efficient

When using the *Image* and *ImageDown* properties of the button, each button requires its own bitmap handle, which is a limited Windows resource. Therefore, if any of your buttons use the same image as another button, all subsequent buttons can simply use that button's *Image* and *ImageDown* by setting their *ExtImage* and *ExtImageDown* properties to that original button. In addition, you can have the *ExtImage* and *ExtImageDown* properties point to a *Tfclmager* and take advantage of all the image manipulation that that component allows.

Change the positions of the caption and glyph

You can use the *Offsets* property to change the position of the *Caption* and *Glyph* for both when the button is up and when the button is down.

Make the parent of the button paint at the button's position

To cause the parent to paint in the button's rectangle before painting the button, set the button's *ParentClipping* property to *False*. This can improve how the buttons look when they are painted, as there will not ever be a period of time where the user can "see through the form". (The purpose of clipping)

Prevent the button from having a region

It can often be desirable to utilize the *TfclImageBtn* to display a rectangular button with a background. In order to accomplish this, you need to tell the button that there is no transparent color. *clNone* tells the button to use the pixel on the upper-left hand corner of the button's image as the transparent color. *clNullColor*, on the other hand, tells the button to not even use a transparent color, and will result in a rectangular button.

How to simulate certain button types

The *TfclImageBtn* is very versatile and can be used in the following traditional ways:

Use it as a *TButton* or *TBitBtn*, which receives focus and is tabbable by setting *Options | boFocusable* and *Options | boFocusRect* to *True*. For some shaped buttons the focusrect is not that attractive and you may wish to instead change the color when the control receives the focus. You can do this by using the *OnEnter* and *OnExit* events to change the *Color* property of the control.

Use it as a *TSpeedButton*, which does not receive focus and thus does not steal focus away from a different control.

Tips

- Use *Options | boToggleOnUp* when the non-transparent area of the image specified by *Image* is different than that of the image specified by *ImageDown*. This improves display behavior in the case where the mouse is held down, and user moves cursor to the non-transparent part of the image.
- When creating a different up and down image, you may want to make the actual image dimensions the same but make the position in the image change. See figure to the left. Notice how the picture on the right has padding on the left. In essence, offset the images within the bitmap much like you would if you were creating an animation.



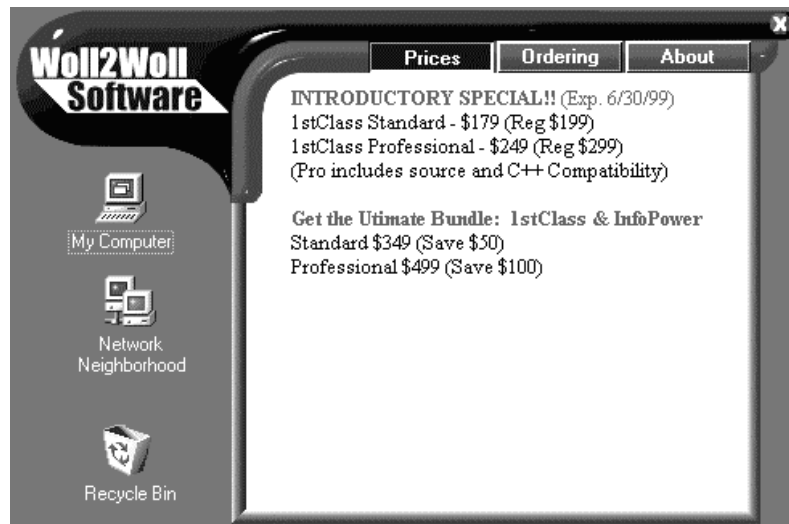
TfcImageForm



The *TfcImageForm* allows you to create a non-rectangular shaped form whose border is defined by the transparent areas of a loaded image, and whose background is defined by the loaded image. When you drop a *TfcImageForm* onto a form, the *BorderStyle* is changed to *bsNone* since the image will define the borders of the form. The *bsNone BorderStyle* has no visible borderline, is not resizable, and does not have a caption bar.

By default clicking on the *TfcImageForm* control at runtime will allow an end user to move/drag the form to a new position. As this is not always desirable, 1stClass has added the ability to bind any *TControl* to be used as a *DragControl* instead. Any control placed on top of this drag control that is not using it's *OnClick* or *OnMouseDown* event will pass the mousedown to the drag control.

Double-clicking the control will bring up the standard picture editor. See the Delphi /C++ Builder does for information on this dialog.



TfcImageForm control

Ancestor

TfcCustomImage
 TfcCustomImageForm
 TfcImageForm

Added Properties

AutoSize

When *True*, this property resizes the form based on the width and height of the loaded image in the *Picture* property.

Data Type: boolean

CaptionBarControl

This property is used to set any *TControl* to be used as a drag control for the form. Since the *TfcImageForm* changes the border style to *bsNone*, there is no caption bar, which the user can drag. To enable some mechanism for the user to drag the form, drop a *TGraphicControl* (such as a Delphi *TShape*, *TImage*, or a 1stClass *TfcImager*) into your form, and then set the *CaptionBarControl* property to this *TGraphicControl*.

The end-user only needs to click on this control to drag the form. If they click on anywhere else, the form will not be dragged. Any other control placed on top of the *CaptionBarControl* that does not have an assigned *OnClick* or *OnMouseDown* event will pass the mouse events to the *CaptionBarControl* for dragging.

Data Type: *TControl*

DragTolerance

This property determines how far the end user must move the mouse before dragging occurs.

Data Type: Integer

Options

This property is a set of boolean flags that control the behavior or display of the *TfcImageForm* control.

ifUseWindowsDrag

When *True*, the form will use the users windows settings when dragging the form. If the Display Properties | Plus! | Visual Settings | Show window contents while dragging checkbox is checked, then the contents of all windows will be shown while the user is dragging. For large forms, this may be a little slower to paint, and cause trails to be left behind as you move the window. To see only the Windows outline when you move a window then that checkbox should be set to *False*. If *ifUseWindowsDrag* is set to *False*, then only the outline of the form will be shown while dragging.

Data Type: *TfcImageFormOptions*

Valid Values: *ifUseWindowsDrag*

Picture

Set *Picture* to a *TPicture* object that contains a bitmap graphic to be displayed by the image control. This property determines the background, borders, and shape of the form. Setting *Picture* at design time brings up the Picture Editor, which can be used to specify the file that contains the image.

Data Type: *TPicture*

TransparentColor

By default this is set to *clNone* which means the pixel at the bottom left hand color of the *Picture* property will be used as the transparent color. Otherwise, you can use this property to override which color should be transparent. The transparent areas of the picture will not be part of the form.

Data Type: *TColor*

Visible

Visible is *True* by default. Set this property to *False* if you wish the *Picture* to only define the shape of the form and to not display the image. Normally you will want to use the bitmap defined by your *Picture* property, as otherwise the form's edges will appear as if they have been cut-off.

However, when setting this to *False* the best way to drag your form would be to drop a *TImage* onto your form and set it to *alClient* and assign it to the *TfcImageForm*'s *CaptionBarControl* property.

Data Type: boolean

Added Methods

ApplyBitmapRegion

This method computes the new transparent area of a form, by rescanning the bitmap defined by the *Picture* property. Call this method if you change the *TfcImageForm*'s *Picture* property during program execution.

You may wish to make the size of your program executable smaller, by storing your image form's bitmaps in external .BMP files. To load the image at runtime, you will need to first call the *LoadFromFile* method of the *Picture* property, followed by calling the *ApplyBitmapRegion* method. For example...

```
fcImageform1.Picture.LoadFromFile('mainform.bmp');  
fcImageform1.ApplyBitmapRegion;
```

How To

How to Create a Picture for the TfcImageForm.

Most of the sample form images found in the images/forms directory were created by a tool called PhotoImpact by Ulead Systems, Inc. However, any photo editing tool would be useful for creating these type of images. A few things to be careful of when designing your own image are:

- For large forms, try to minimize the number of colors used to 16 colors or 256 colors since the image is stored in your .DFM file and the larger your image the larger your .DFM file and executable will be.
- Be careful when creating forms not to antialias (blur) the edges into your background transparent color. As then you will have artifacts along the edges of your form.
- Keep in mind that your form will not have any borders and design that into your image. As your image is exactly what the Form will look like. Many photoediting tools will allow you to extrude or buttonize an image which create cool 3D effects (be careful of antialiasing around the outside of the image).
- When you are done designing the form you can select parts of the form and copy them to a new file to create buttons or the *CaptionBarControl*.
- The bottom left pixel is used to determine the color of the form that will not be accessible. (You can override this later by setting the *TransparentColor* property.)

How to create your own draggable caption control.

After you have created your ImageForm picture, you can use a subset of this image to create the *CaptionBarControl* image. Select the portion of the imageform that you want to be draggable and copy it to the clipboard. Then create a new file and paste the selected portion from the clipboard. Now you can extrude, buttonize, or add any other 3D effects that you want and then load this image into a TfcImager or a TImage and assign that control to the *CaptionBarControl* property of the *TfcImageForm*. For more information see the *CaptionBarControl* property.

Note: For precise tweaking of the position of the caption control, it is useful to hold down the control key while you press the arrow keys.

Tips

- Using a bitmap can increase the size of your .dfm files and executables. So for large forms we recommend using only 16 to 256 colors instead of High or True Color.

- After dropping a `TfcImageForm` onto your form at design time, you can get access to the form properties by selecting the `TfcImageForm` and hitting the `<ESC>` key.
- After you drop a `TImage` or `TfcImager` on a form and assign it to the `CaptionBarControl` property, it is not necessary to actually assign a `Picture` to these controls. If you do not need a graphic, then this will reduce the size of your executable and `.DFM` file. You can size the control to the area that you wish to be draggable. Any control dropped on this area (like a panel, button, etc.) that has a `mousedown` or `click` event assigned will not be draggable. Thus you can use a panel to make a certain portion of the `CaptionBarControl` not draggable by assigning some dummy event to its `OnClick` or `OnMouseDown` events.

Tfclmager



The 1stClass Imager Control allows for the easy creation of spectacular images that can be integrated into any application. All of the effects available through this control can be easily manipulated by setting just a few properties.



Tfclmager control

Use the *BitmapOptions* to add special effects to the image. Set the *DrawStyle* property to specify if the image is stretched, tiled, centered, etc.

This control can be attached to a *TfcDBTreeView* or a *TfcOutlookBar* to paint a background bitmap into these controls. See the *Imager* property for these controls

Ancestor

TGraphicControl
 TfcCustomImager
 Tfclmager

Added Properties

AutoSize

Set this property to *True* to make the size of the Tfclmager the same size as the bitmap stored in its *Bitmap* property.

Data Type: boolean

BitmapOptions

A set of different effects that can be applied to the Image.

AlphaBlend

A set of properties that allow the alpha-blending of TfcImager's *Bitmap* and this *Bitmap*.

Amount A number ranging from 0 to 255 that specifies how much of the TfcImager's image is blended with *Bitmap*. A value of 255 will show only the image stored in *AlphaBlend.Bitmap*. A value of 0 will show only the image stored in *TfcImager.Bitmap*.

Data Type: Integer

Bitmap The bitmap to blend the base image with. The amount of blending is dependent on the *Amount* property.

Data Type: TfcBitmap

Transparent Determines whether or not to use the upper-left hand corner pixel of the *AlphaBlend.Bitmap* property as a transparent color. If this is *True*, then any pixel of that color will not be blended with the base bitmap.

Data Type: boolean

Color

Setting this property to a value other than *clNone* will result in the colors of the image being heavily skewed towards a roughly equivalent shade of the specified color. Setting *Grayscale* to *True* in conjunction with this property is highly recommended.

Data Type: TColor

Contrast

Setting this property to a high value causes dark and light pixels to appear darker and lighter relative to each other. This can make the image appear more vivid. Setting this property to a low (negative) value causes dark and light pixels to approach each other in intensity, thereby simulating a "faded" appearance. A value of 0 will disable this effect.

Data Type: Integer

Embossed

When this property is *True*, the image will take on a *bas-relief* look. The image appears embossed.

Data Type: boolean

GaussianBlur

The effects of this property can be noticed at low values. A value of 1 causes noticeable blur. This property does not work well in tandem with the *Transparent* property, as the transparent color will get blended with the rest of the image. The effect of this property is similar to that of a camera going out of focus.

Data Type: Integer

Grayscale

Setting this property to *True* will convert the image to a grayscale image. This property has a similar (although not quite identical) effect as the setting the *Saturation* property to 0.

Data Type: Integer

HorizontallyFlipped

Setting this property to *True* will flip the image along its *Y* axis.

Data Type: Integer

Inverted

Setting this property to *True* will invert every pixel to its opposite. This causes an appearance similar to a camera photo's negative.

Data Type: boolean

Lightness

This property controls the overall lightness/brightness of the image. Positive values lighten the image, whereas negative values darken the image.

Data Type: Integer

Rotation

This property controls the rotation of the image.

Angle

The precise amount of rotation (in degrees) is controlled by this property.

Data Type: Integer

CenterX, CenterY

These properties determine the coordinate about which the rotation occurs. Values of -1 , -1 , cause the rotation to occur at the center of the image.

Data Type: Integer

Saturation

This property determines the color intensity in the image. A low value causes the image to appear more grayscale. A high value causes the image to appear awash in color. This property has a *large* range. A noticeable increase in color saturation is generally not noticeable until the value of this property approaches 500 or more.

Data Type: integer

Sharpen

When this property is set to even small values, the image appears to come into better focus. Even a value of 1 causes a noticeable change.

Data Type: Integer

Sponge

This property causes a somewhat artistic effect to be applied to the image. The result is an image that looks as though it was rendered using sponges or a thick spray.

Data Type: Integer

TintColor

When this property is set to a value other than *clNone*, the base bitmap is painted with a bias towards that color. Grayscale images, in particular, will appear in this color.

Data Type: TColor

VerticallyFlipped

Setting this property to *True* will flip the image along its *X* axis.

Data Type: boolean

Wave

This property allows sophisticated warping effects to be applied to the image.

Note: The properties *Ratio*, *XDiv*, and *YDiv* must contain values greater than zero for this effect to be applied.

Ratio Setting this property to a high value increases the overall warping.

Data Type: Integer

Wrap If this property is *True*, then portions of the image that would otherwise be empty will contain the pixels on the opposite side of the image as though the image is tiled.

Data Type: boolean

XDiv, YDiv These properties control the level at which the image is distorted along the *X* and *Y* axis respectively.

Data Type: Integer

DrawStyle

This property allows for the image to be rendered in many different styles.

dsNormal

The image will be drawn with the upper-left hand corner aligned with the upper-left hand corner of the *TfcImager* control. No stretching occurs.

dsCenter

The image is centered such that the space to the left is equal to the space to the right, and the space above is equal to the space below. No stretching occurs.

dsStretch

The image is stretched to fill the entire client area of the *TfcImager*.

dsTile

The image is not repeated in a tile pattern across the client area of the imager.

dsProportional

The image is stretched to the maximum size possible while still retaining its original aspect ratio. (i.e. The quotient of the image's width divided by its height is equal from the original bitmap to the rendered image.)

dsProportionalCenter

The image is stretched to the maximum size possible while still retaining its original aspect ratio. Additionally the resulting stretched image is centered.

Focusable

This property determines whether or not this control can receive focus. This is a departure from a standard *TGraphicControl*. Other supporting properties and events were added that will only work if this property is set to *True*.

Data Type: Boolean

Picture

This property contains the actual graphic to display. Because it is of type `TPicture`, it is possible to load any picture that the Delphi IDE is equipped to handle.

Data Type: `TPicture`

PreProcess

When this property is *True* (the default), all the effects are applied to an internal bitmap the same size as the image defined in *Picture*. The final stage of the painting process then stretches or tiles this bitmap onto the imager. When this is set to *False*, the internal bitmap size is set to the size of the actual imager control. This is only valid when *DrawStyle* is set to *dsStretch*, *dsProportional*, or *dsTile*. The result is that because the effects are applied after stretching or tiling occurs, the final image can look better. In addition, because the stretching operation only occurs once, greater performance can be achieved in certain situations.

Data Type: `boolean`

RespectPalette

This property is only relevant on systems displaying 256 colors or less. When this is *True*, and there is a palette defined for *Picture* (*Picture* must be a `TBitmap` and the bitmap must have 256 colors or less) then the resulting image that is displayed will look much closer to what is defined for the image than if this property were set to *False*. However, this can occasionally result in slower performance.

Data Type: `boolean`

ShowFocusRect

This property does nothing if `Focusable` is `False`. Otherwise a focusrect will be drawn around the perimeter of the image when this property is set to `True`.

Data Type: `Boolean`

SmoothStretching

When this property is *True*, a more lengthy algorithm is used when stretching the image. The result is a slight performance loss, but a significant improvement in appearance.

Data Type: `boolean`

TabOrder, TabStop

These properties depend on `Focusable` being `true`. See Delphi documentation for a description of these standard `TWinControl` properties.

Transparent

When this property is *True*, then a given color (dependent on the *TransparentColor* property) will not be drawn when the imager is drawn and will therefore be "see-through". The default color is the color on the upper-left hand corner of the image.

Data Type: boolean

TransparentColor

This property is only used when *Transparent* is *True*. It determines what color will be the "see-through" color of the image. If it is set to *clNone*, then the upper-left hand pixel of the image will be used. Otherwise, whatever value is set for this property will be transparent.

Data Type: TColor

WorkBitmap (Runtime only)

Reference this property if you wish to gain access to the working bitmap. The working bitmap refers to the image after it has been manipulated by the *TfcImager*'s property settings. This differs from the *Picture* property, which is the original bitmap. In order to convert a *TfcBitmap* to a *TBitmap*, use the *TfcBitmap*'s *SaveToBitmap* method.

Data Type: TfcBitmap

How To

Integrate the imager into other 1stClass components

The *TfcOutlookBar* and *TfcDBTreeView* have native support for embedding a *TfcImager* within its client area. To accomplish this, simply set the *Imager* property of the *TfcOutlookBar* or the *TfcDBTreeView* to the desired *TfcImager*. The associated *TfcOutlookLists* can be made transparent in order to see the background image of the *TfcOutlookBar* by setting the *Transparent* property of the *TfcOutlookList* to *True*.

Partially Blend a Second Bitmap with the Imager

Set the properties of the *AlphaBlend* sub-property. Set the *AlphaBlend.Bitmap* property to the bitmap that you want to blend with. Set the *AlphaBlend.Amount* property to indicate how much you want to blend the current image with the new image. To specify that you want to not blend in any color that matches the color on the upper-left hand corner of the blended bitmap, set the *AlphaBlend.Transparent* property to *True*.

TfcLabel



The 1stClass TfcLabel Control enables the easy creation of impressive text effects. Using a TfcLabel one can add shadows, extrusions, engraved, embossed, or outline effects.



TfcLabel with some sample effects.

All of the effects available through this control can be easily manipulated by setting just a few properties of the TfcTextOptions. The 1stClass TfcLabel control has also added *OnMouseEnter* and *OnMouseLeave* events so that you can easily add hot-tracking or URL Links to the labels in your applications.

Ancestor

TGraphicControl
 TfcCustomLabel
 TfcLabel

Added Properties

DataField

Optional: This property contains the name of the field that you want to bind the TfcLabel to. If you do not wish to bind the label to a table field, then leave both the *Datafield* and *Datasource* properties blank. The default value is blank (unbound).

Data Type: String

DataSource

Optional: This property contains the name of a TDataSource component that provides the label control with data. The default value is blank (unbound).

Data Type: TDataSource

TextOptions

Please see the documentation for *TfcText*.

Added Events

OnMouseEnter

Occurs when the mouse cursor passes from outside the control to inside the control.
The parameters for this event are as follows:

Sender:TObject 1stClass label that is associated with this event.

OnMouseLeave

Occurs when the mouse cursor passes from inside the control to outside the control.
The parameters for this event are as follows:

Sender:TObject 1stClass label that is associated with this event.

How-to

Make multi-line labels

Double click on the *Caption* property to add multiple lines to your label control. The *TextOptions* property contains wordwrap and line spacing properties to further customize the appearance of multi-line text.

Center caption vertically or horizontally

Use the *TextOptions.Alignment* and *TextOptions.VAlignment* properties.

Add Hot-Tracking and URL Links

The following example demonstrates how you can use the *OnMouseEnter* and *OnMouseLeave* events to easily create URL Links in your applications. In this example the TfcLabel will turn blue when the mouse is over the label and the cursor will change to a hand. Then after clicking on the label your internet browser will open up to the specified URL.

- 1) Add a TfcLabel component to your form and set the following properties.

```
AddCaption = 'Go to the Woll2Woll Home Page '  
Cursor = crHandPoint
```

- 2) Put the following code in the TfcLabel's *OnMouseEnter* event to change the label's font to blue when the mouse enters the control.

```
Procedure TForm1.fcLabel1MouseEnter(Sender: TObject)  
begin  
    (Sender as TfcLabel).Font.Color := clBlue;  
end;
```

- 3) Put the following code in the TfcLabel's *OnMouseLeave* event to change the label's font to black when the mouse exits the control

```
Procedure TForm1.fcLabel1MouseLeave(Sender: TObject)
begin
    (Sender as TfcLabel).Font.Color := clBlack;
end;
```

- 4) Finally, put the following code in the TfcLabel's *OnClick* event to open the registered program for the specified selection.

```
Procedure TForm1.fcLabel1Click(Sender: TObject)
begin
    ShellExecute(Handle, 'OPEN', PChar('http://www.woll2woll.com'),
        nil, nil, sw_shownormal);
end;
```

Tips

- Set *AutoSize* to *False*, before you set the *WordWrap* property to *True*. Then size the label control to your desired width and height or set the *Align* property.
- Some fonts may not be able to be rotated. *TextOptions.Rotation* works only with TrueType fonts.
- You may often wish to set *TextOptions.DoubleBuffered* to *True* so that all of the text is painted at once. See *TfcText* for more on this property.

TfcOutlookBar



The 1stClass *TfcOutlookBar* allows for the grouping of *TfcOutlookLists* and any other control into logical and easy to manage subsets. This control provides the functionality seen in Microsoft's Outlook Bar, in addition to a number of other enhancements.



TfcOutlookBar control

The *TfcOutlookList* within a *TfcOutlookBar* can present a vertical column or horizontal row of icons for selecting program functions. You can additionally group the icons into logical sets and divide them into separate outlook pages. Clicking on an outlook button will roll the corresponding outlook page and its associated icons into view.

1stClass supports glyphs in the group buttons, and allows you to even embed controls in each group. You can also embed a background image into the outlook list by using the *Imager* property

If instead you wish to embed controls into an outlook page, then delete the *TfcOutLookList* and then add the control.

1stClass provides the following design-time aids when configuring your shape button.

- If you right-click over the selected button in the OutlookBar, the popup menu for those controls will appear in addition to the ones for the *TfcOutlookBar*.
- Double-clicking the control will bring up the standard collection editor. See the Delphi / C++ Builder docs for information on this dialog.
- When you right-click the button group at design time, you can access the following actions from the pop-up menu.

New Button

Selecting this item will cause a new button to be added to the ButtonGroup.

TfcOutlookPanel – Create OutlookList

Selecting this item will create an outlook list in the current outlook page.

TfcOutlookPanel – Paste

Selecting this item will paste the controls in the clipboard to the current outlook page.

Ancestor

TCustomPanel
 TfcCustomTransparentPanel
 TfcCustomButtonGroup
 TfcCustomOutlookBar
 TfcOutlookBar

Added Properties

The TfcOutlookBar provides all the properties, events, and methods of the TfcButtonGroup. It additionally provides for the properties, events, and methods defined in the following pages.

ActivePage

This property controls the currently selected button and page in the OutlookBar.

Animation

Manipulate the properties of this property to control the animation that occurs when changing pages.

Data Type: TfcAnimation

The following are properties of TfcAnimation.

Enabled

Set to *False* to disable animation. You may wish to turn off animation if *ButtonClassName* is set to *TfcImageBtn*, as animation can paint slowly in this case.

Data Type: boolean

Interval

Specifies the amount of time that occurs between each frame of the animation.

Data Type: Integer

Steps

Specifies how many frames the animation will go through to get from one page to the other.

Data Type: Integer

ButtonClassName

See the *TfcButtonGroup.ButtonClassName* property.

ButtonSize

Determines how large the buttons in the OutlookBar are. If *Layout* is set to *loVertical*, then this affects the *Height* of the buttons. If *Layout* is set to *loHorizontal*, this affects the *Width* of the buttons.

Data Type: Integer

Imager

Assign this property to an existing TfcImager control to give the TfcOutlookBar a tiled background image. Each individual panel of an outlook bar is already transparent, but to make the individual outlook lists transparent, set their corresponding *Transparent* property to *True*.

Data Type: TfcCustomImager

OutlookItems

Same as the inherited *TfcButtonGroup.ButtonItems* property, but returns the derived class *TfcOutlookPages* instead. *TfcOutlookPages* contains an array of *TfcOutlookPage*, in which you can access an individual page by specifying an index. For instance, the following returns the *TfcOutlookPage* associated with *index*.

```
OutlookItems[index]
```

TfcOutlookPage has the following properties which you can access via code.

OutlookBar

Returns the *TfcOutlookBar* associated with this page.

OutlookList

Returns the *TfcOutlookList* associated with this page.

Panel

Returns the *TfcOutlookPanel* associated with this page. *TfcOutlookPanel* is the same as *TPanel*. It is introduced for component-editor reasons. There is no enhanced functionality. However, this component is the parent of the associated *TfcOutlookList* and *any* other control on the page.

Options

This property is a set of boolean flags that control various aspects of the OutlookBar.

Data Type: TfcOutlookBarOptions

Valid Values: cboAutoCreateOutlookList, cboTransparentPanels

cboAutoCreateOutlookList

If set (the default), each page will automatically have a `TfcOutlookList` created for it. Otherwise, in order to create a `TfcOutlookList` for a page, the popup-menu for the component, `Create Outlook List`, must be selected.

cboTransparentPanels

If set, the underlying panel of each page will be transparent. This property is best used if the *Transparent* property of the entire control is also set to *True*.

PanelAlignment

This property determines where the buttons in the `OutlookBar` are grouped.

Data Type: `TfcPanelAlignment`

Valid Values: `paDynamic`, `paTop`, `paBottom`

paDynamic

When this is set (the default), the current page appears underneath the selected button.

paTop

This causes all the buttons to be grouped at the top, and the current page to be at the bottom.

paBottom

This is the opposite of *paTop*. All of the buttons are grouped at the bottom, and the current page appears at the top.

ShowButtons

If this property is set to *False*, then the buttons in the button group will not be shown. This is useful if you want to mimic the functionality of a *PageControl* which has the *TabVisible* property of its *TTabSheets* set to *False*.

ShowDownAsUp

When this property is set to *True*, the selected outlook bar button will display as down only while clicking the button. When the button has become selected, then this button will be displayed as an up state button even though the actual state of the button is down. The Default is *False*.

Data Type: `Boolean`

Added Events**OnChange**

See `TfcButtonGroup` *OnChange* event

OnChanging

See `TfcButtonGroup` *OnChange* event

Added Methods

TfcOutlookPages methods (Access via *OutlookItems* property)

Add

Adds a new *TfcOutlookPage* to the OutlookBar and returns the newly created *TfcOutlookPage*.

```
function Add: TfcOutlookPage;
```

TfcOutlookPage methods (Access via *OutlookItems[index]* property)

CreateOutlookList

Call this method if a *TfcOutlookList* for this item has been destroyed (or was never created) and it needs to be recreated.

How To

Change the Speed and Style of the Animation

Adjust the *Steps* and *Interval* sub-properties of the *Animation* property.

Tips

- When embedding a lot of controls in the Outlookbar's individual panels, you may wish to turn *Animation.Enabled* to *False* for enhanced performance. As an alternative, you can reduce the number of animation steps to improve performance, while retaining some animation. Animation can also be slow if you have set your *ButtonClassName* to *TfcImageBtn*.
- Delete the default *TfOutlookList* if you wish to embed your own controls in a section. In addition, you can set the *Options | cbo.AutoCreateOutlookList* to *False* if you do not want the OutlookLists to be automatically created. In order to re-create the outlook lists, simply right-click on the outlook panel and choose "Create OutlookList" from the pop-up menu.

TfcOutlookList (Class)

The TfcOutlookList allows for the grouping of a number of related items into an organized and easy to use set. The items in the OutlookList can be clicked on or selected. The layout can be arranged in either a vertical or horizontal fashion. **Note:** This component is only available *through* the TfcOutlookBar control.



TfcOutlookList control

Double-clicking the control will bring up the standard collection editor. See the Delphi / C++ Builder docs for information on this dialog.

Ancestor

TCustomControl
TfcCustomOutlookList
TfcOutlookList

Added Properties

ClickStyle

This property determines the behavior of the TfcOutlookListItems when they are clicked on.

csClick

This value causes the items to behave like buttons. Clicking on them will fire the *OnItemClick* event.

csSelect

Clicking on items when *ClickStyle* is set to *csSelect* will cause the items to be selected in a similar manner to Buttons that get selected when the buttons are part of the same group. (i.e. The *GroupIndex* property of the buttons are set to the same value that is greater than 0) Only one item at a time can be selected, and that item will remain pressed until another item is selected.

Data Type: TfcCustomOutlookListClickStyle

Valid Values: csClick, csSelect

HotTrackStyle

The appearance of the items as the mouse tracks over it is controlled by this property.

Data Type: TfcOutlookHotTrackStyle

Valid Values: hslconHilite, hsItemHilite

hsIconHilite

When this is the *HotTrackStyle*, only the *Icon* is hot-tracked.

hsItemHilite

When this is the *HotTrackStyle*, the entire *Item* (Icon and Text) is hot-tracked.

Images

Controls the images that are displayed as the icon of each item.

Data Type: TCustomImageList

Items

This property contains TfcOutlookListItems collection, which in turn stores the individual TfcOutlookListItem's of the OutlookList. It has a default array property that can be accessed through the Items property itself. (i.e. Items[i])

For instance, Items[index] returns the *TfcOutlookListItem* associated with *index*.

The following are the properties of TfcOutlookListItem

Action

Assign this property if you wish to associate this outlook item with an action in a TActionList control. See Delphi documentation on *TActionList*

Enabled

Set this property to False to disable the item. The item text is painted according to the *ItemDisabledTextColor* property. Note: When setting this property to False, you may wish to modify your default outlooklist *Color* so that the *ItemDisabledTextColor* has more contrast and is more readable.

GlyphOffset

Assign this property when using the *TextAlignment* property. This property defines the spacing between the glyph and the edge of the outlook list. This value must be a non-zero value for the property to be enabled.

Hint

Assign this property to display a hint for the item. The hint is displayed at the top left of the item rectangle.

ImageIndex

Controls which image in the associated TfcOutlookList's *ImageList* is displayed with this item.

ItemDisabledTextColor

Set this property to change the color of the text that is used when painting disabled items.

ItemRect

Returns the bounding rectangle of the item.

MouseOnItem

Returns true if the mouse is currently over the item.

Selected

This property controls whether or not the item is selected. This property is only valid if the TfcOutlookList's *ClickStyle* property is set to *csSelect*. Setting this property will clear the *Selected* property of the other items in the TfcOutlookList.

Separation

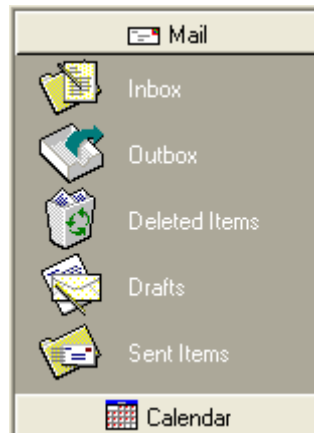
Indicates the amount of space between the item's *Icon* and its *Text*.

Text

The text that appears next to the *Icon* of each item.

TextAlignement

Use this property when the OutlookItem's *ItemLayout* property is set to *blGlyphLeft* or *blGlyphRight*, and you wish to align the text to the right or left (instead of the default centering). Note: *You will also need to set the item's GlyphOffset to a non-zero value for this property to be enabled.*



Text aligned vertically by using the `TextAlignment` property

Visible

Set this property to false to hide this item

ItemHighlightColor

Controls what color the upper-left sides of the item (when hot-tracked, lower-right sides when pressed) will be painted in.

Data Type: TColor

ItemHotTrackColor

When the mouse is over an item, this property determines what color the item will be painted in.

Data Type: TColor

ItemLayout

Controls the position of the glyph relative to the text.

Data Type: TButtonLayout

ItemShadowColor

Controls what color the lower-right sides of the item (when hot-tracked, upper-right sides when pressed) will be painted in.

Data Type: TColor

ItemSpacing

This property controls how far apart each item in the TfcOutlookList is.

Data Type: Integer

ItemsWidth

Controls the width of each item in the TfcOutlookList. This property is only used if the *Layout* property is set to *loHorizontal*.

Data Type: Integer

Layout

Determines whether the items are arranged from the top-down, or from the left to the right. Affects the position and orientation of the arrows of the scroll buttons.

Data Type: TfcLayout

Valid Values: loVertical, loHorizontal

PaintCanvas

For use with the OnDrawText event. Use this property when painting to the outlook list from within the event.

Data Type : TCanvas

ScrollButtonsVisible

Set this property to *False* to hide the scroll buttons.

Data Type: boolean

ScrollInterval

This property controls how quickly the items in the OutlookList scroll when the user holds down the mouse button over a scroll button. The value is the time in milliseconds between each scrolling operation.

Data Type: Integer

Selected (Runtime only)

This read-only property returns the currently selected item in the OutlookList. This property *can* be **nil**.

Data Type: TfcOutlookListItem

Transparent

Set this property to *True* in order to see the underlying *Imager* contained in the outlook bar. The *Imager* property of the corresponding outlook bar must be set.

Data Type: boolean

Added Events

OnDrawItem

Use this event to customize the appearance of the items within the outlook list. Alter the GlyphPos and TextPos parameters to affect the position of the glyph and text portions of the item respectively. Reference the outlook list's *PaintCanvas* in conjunction with the *ItemRect* property to draw onto the item. The parameters for this event as follows:

- OutlookList: The TfcOutlookList being drawn on.
- Item: The current TfcOutlookListItem being drawn.
- GlyphPos: This is the upper-left hand point of the rectangle that the glyph will be painted to. Adjust this position to alter where the glyph will be painted.

TextPos: This is the upper-left hand point of the rectangle that the text will be painted to. Adjust this position to alter where the text will be painted.

DefaultDrawing Set this parameter to False to prevent the item from being painted by the outlook list.

The following example uses the `OnDrawItem` event to custom paint the outlook items. The glyph is painted 5 pixels from the left border. The text is painted 10 pixels to the right of the glyph. The selected item is painted with a blue background.

```
procedure TForm1.fcOutlookBar1OutlookList1DrawItem(  
    OutlookList: TfcCustomOutlookList; Item: TfcOutlookListItem;  
    var GlyphPos, TextPos: TPoint; var DefaultDrawing: Boolean);  
begin  
    GlyphPos.x := 5;  
    TextPos.x := OutlookList.Images.Width + 15;  
  
    if Item.Selected then  
    begin  
        OutlookList.PaintCanvas.Brush.Color := clBlue;  
        OutlookList.PaintCanvas.FillRect(Item.ItemRect);  
    end;  
end;
```

OnItemClick

This event is only valid when the `ClickStyle` property is set to `csClick`. The event fires when the `TfcOutlookListItem` is clicked on. The parameters for this event are as follows:

OutlookList: The `TfcOutlookList` related to the item that was clicked on.

Item: The `TfcOutlookListItem` that was clicked on.

OnItemChange

This event is only valid when the `ClickStyle` property is set to `csSelect`. The event fires when the active selection in the `TfcOutlookList` changes. The parameters for this event are as follows:

OutlookList: The `TfcOutlookList` related to the item that was clicked on.

Item: The `TfcOutlookListItem` that has just become selected.

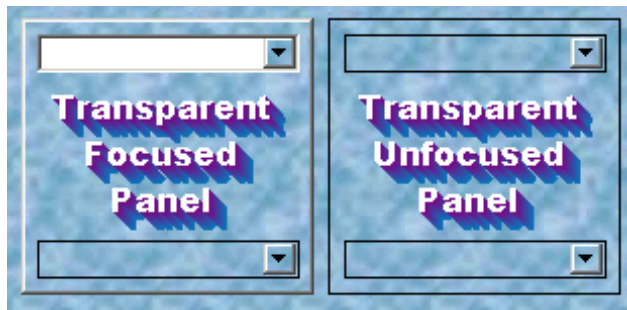
How To

Make the OutlookListItems appear like Outlook Express' OutlookBar

Change the `HotTrackStyle` property to `hsItemHilite`. You may want to change the `ItemHottrackColor`, `ItemHighlightColor`, and `ItemShadowColor` properties to `clBtnFace`, `clBtnHighlight`, and `clBtnShadow` respectively.

TfcPanel

The TfcPanel component is similar to the native TPanel component, with additional options for run-time transparency and custom framing. These panels can even change colors to indicate focus.



TfcPanel control with Transparency and Framing

Ancestor

TCustomPanel
 TfcCustomPanel
 TfcPanel

Added Properties

Frame

See TfcEditFrame for more information on this property. However, *MouseEnterSameAsFocus* and *AutoSizeHeightAdjust* do not apply to the TfcPanel.

If you wish the panels to change to a different color when a control on the panel gets the focus, then just set the *Color* property of the TfcPanel to the desired focused color and the *Frame.NonFocusColor* property to the color when the panel loses focus. The *transparent* property must be *False* in order to create this effect.

Data Type: TfcEditFrame

Transparent

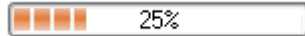
Controls whether or not to display the TfcPanel transparently.

Data Type: Boolean

TfcProgressBar



Use the TfcProgressBar to add a progress bar to a form. Progress bars provide users with visual feedback about the progress of a procedure within an application. As the procedure progresses, the rectangular progress bar gradually fills from left to right with the system highlight color.



Added Properties

DataField

Optional: This property contains the name of the field that you want to bind the control to. If you do not wish to bind the control to a table field, then leave both the *Datafield* and *Datasource* properties blank. The default value is blank (unbound).

Data Type: String

DataSource

Optional: This property contains the name of a TDataSource component that provides the control with data. The default value is blank (unbound).

Data Type: TDataSource

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

Assign this property to format the text displaying the progress. See the Delphi TNumericField.DisplayFormat property for more details on the format string. When this property is unassigned the default text displays a trailing '%' after the number.

Max

Specifies the upper limit of the range of possible positions. See the *Min* property.

Min

Specifies the lower limit of the range of possible positions. Use Max along with the Min property to establish the range of possible positions in a progress bar. When the process tracked by the progress bar begins, the value of Position should equal Min.

BlockColor

The filled area of the progress bar is composed of a series of blocks. Each block's color is determined by the *BlockColor* property. If your project has enabled XP themes then this property is ignored as the theme paints the progress and not the control.

BlockSize

The filled area of the progress bar is composed of a series of blocks. Each block's size is determined by the *BlockSize* property. Set Smooth to True to display the progress as one contiguous block

Orientation

This property determines if your progress bar is displayed vertically or horizontally.

Progress

Assign this property to change the current progress of the progress bar.

Smooth

Assign this property to display the progress as one contiguous block. See also the *BlockSize* property.

Step

This property is referenced by the *StepIt* method. *StepIt* increments the current progress by this amount.

ShowProgressText

Set this property to false to hide the text that shows the current progress.

Added Events

OnChange

Use this event to write your own custom handler when the progress changes.

Added Methods

StepIt

Calling this method increments the current progress by the *Step* property.

```
procedure StepIt
```

StepBy

Calling this method increments the current progress by the amount defined by the *Delta* parameter.

```
procedure StepBy(Delta: integer)
```

TfcShapeBtn



The 1stClass TfcShapeBtn provides the combined functionality of the TSpeedButton and the TBitBtn with the ability to choose any shape or color. Some of its enhancements include the following:

- Use the *Shape* property to customize the shape of the button to one of the pre-defined shapes: Arrow, Ellipse, RoundRect, Rectangle, Star, Triangle, Trapezoid. In addition, by defining a simple list of points (see the *PointList* property), any shape imaginable is possible.
- The button can appear in any color and the highlight colors that are responsible for the 3D effects are completely changeable.
- Any of the shapes (even the custom shapes) can have one of four orientations: Up, Down, Right, and Left.
- The shape is transparent outside its boundaries.



TfcShapeBtn controls

1stClass allows the button to be data-bindable. In addition, you can embed the button in an InfoPower grid.

1stClass provides the following design-time aids when configuring your shape button. When you right-click the button at design time, you can access the following actions from the pop-up menu.

Set Shade Colors

Calls the *UpdateShadeColors* method (See *TfcImageBtn.UpdateShadeColors*) which approximates appropriate highlight/shadow colors for the button based on the value of the *Color* property.

Size To Default

Sets the size of the button to a square.

Ancestor

TWinControl
 TfcCustomBitBtn
 TfcCustomImageBtn
 TfcCustomShapeBtn
 TfcShapeBtn

Added Properties

The TfcShapeBtn provides all the properties, events, and methods of the TfcImageBtn with the exception of *DitherStyle*, *ExtImage*, *ExtImageDown*, *Image*, *ImageDown*, and *TransparentColor*. It additionally provides for the properties, events, and methods defined in the following pages.

Color

To change the color of the button, set this property to the desired color.

Data Type: TColor

DataField

Optional: This property contains the name of the field that you want to bind the control to. If you do not wish to bind the control to a table field, then leave both the *Datafield* and *Datasource* properties blank. The default value is blank (unbound).

Data Type: String

DataSource

Optional: This property contains the name of a TDataSource component that provides the control with data. The default value is blank (unbound).

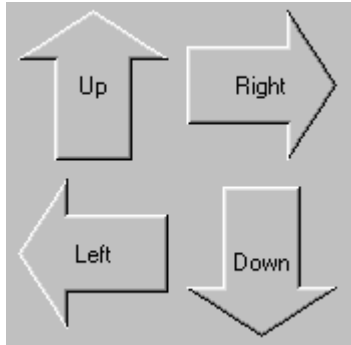
Data Type: TDataSource

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*.

Orientation

Determines the direction that the button faces. For custom shapes, the points in the point list are assumed to have an orientation of *Up*.



Note: Shapes that are both vertically and horizontally symmetric will obviously not be affected by this property.

Data Type: TfcShapeOrientation

Valid Values: soLeft, soRight, soUp, soDown

PointList

When the *Shape* property is set to *bsCustom*, this property is used to define the set of points that make up the shape of the button. The format is standard "x, y" notation. Two special variables are allowed—Width and Height—which are replaced by the width and height of the shape. In addition, the standard math operators "+", "-", "*", and "/" (Add, Subtract, Multiply, and Divide, respectively) are valid. For example, to define the points of a Right-Triangle shape, the PointList property should look like:

```
0, 0
Width, 0
0, Height
```

To define a regular hexagon for your shape, enter the following coordinates into the PointList:

```
2 * Width / 7, 0
Width - 2 * Width / 7, 0
Width, 2 * Height / 7
Width, Height - 2 * Height / 7
Width - 2 * Width / 7, Height
2 * Width / 7, Height
0, Height - 2 * Height / 7
0, 2 * Height / 7
```

Data Type: TStringList

RoundRectBias

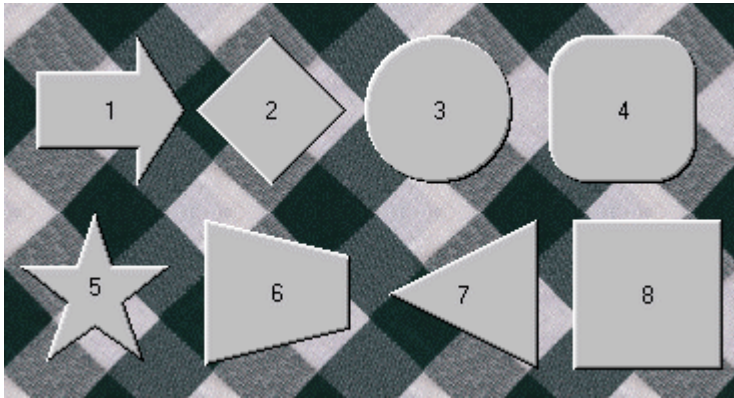
This property is only used when the *Shape* property is set to *bsRoundRect*. When this is the case, *RoundRectBias* determines how much the corners of the rectangle are rounded. This property defaults to a value of 25.

Data Type: Integer

Shape

The shape of the button is determined by this property. If this property is set to *bsCustom*, then the *PointList* property must be defined with a set of valid points. If this is not set properly, the button will be displayed by a red box.

The standard shapes are pictured below:



Data Type: TfcButtonShape

Valid Values: bsRoundRect, bsEllipse, bsTriangle, bsArrow, bsDiamond, bsRect, bsStar, bsTrapezoid, bsCustom

StaticCaption

Set this property to false to ignore the value from the underlying database (specified by datasource/datafield). You may wish to set StaticCaption to true when embedding in an InfoPower grid to show the same caption for each record.

Added Methods

SizeToDefault

Call this method to set the size of the button to a square. The side with the shorter length will be increased to that of the side with the greater length.

procedure SizeToDefault; **override**;

How To

Define a custom shaped button

Fill the *PointList* with all the points in the button in "dot-to-dot" order. For more information on this property, please see the documentation under *PointList*. Once you have all the points set, set the *Style* property to *bsCustom*.

Automatically set the shade colors based on the current color

To automatically set the *ShadeColors*' highlight and shadow color sub-properties based on the current value of the *Color* property, call the *UpdateShadeColors* method. At design time, right-clicking on the control will yield an "Update Shade Colors" popup menu that can be selected to call *UpdateShadeColors*.

Mimic the look and behavior of the TSpeedButton's flat style

When the mouse goes over a TSpeedButton, the borders of the button appear with a thickness of one pixel. To duplicate this behavior, first set the *ShadeStyle* property to *fbsFlat*. At this point, the border will be two pixels thick. To make the borders one pixel thick, just change the *ShadeColors*' *Btn3dLight* and *BtnShadow* properties to the same color as your button, and the *ShadeColors.BtnBlack* property to what the *BtnShadow* property was (usually *clBtnShadow*).

Allow the button to receive focus

Enable the *boFocusable* option in the *Options* property.

Change the focus color

Change the *Options* | *boFocusable* property to *True* and set the *ShadeColors.BtnFocus* to the desired color.

Tips

- Some shapes look best when width and height are the same (like the star).
- When setting the color of a TfcShapeBtn to a color that is different than the parent control, then often there is an optical illusion that makes the caption appear to go in a different direction than the button. As a result you can use the offsets property to make your text/glyphs move in the same direction.

TfcStatusBar



The 1stClass TfcStatusBar is composed of a series of TfcStatusPanels. This control provides the functionality seen in Delphi's TStatusBar, in addition to major enhancements in the panels it displays.

- Some of its features include the following:
- Automatically display hints from controls and menu items.
- Display the current date, time, or datetime information
- Automatically display the current state of your keyboard (Num Lock, Caps Lock, Scroll Lock, Insert/Overwrite).
- Embed your own custom controls in the individual panels of the statusbar. Using the control property you can embed gauges, progress bars, edits, comboboxes, spinedits, etc.
- Display the computer information, richedit line/column information, and so much more.



TfcStatusBar control

Double-clicking the control will bring up the standard collection editor. See the Delphi / C++ Builder docs for information on this dialog.

Ancestor

TWinControl
 TfcCustomStatusBar
 TfcStatusBar

Added Properties

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*.

Images

Images contains a list of images that can appear on the Statusbar's status panels. Each panel's *ImageIndex* property determines the specific image displayed on it. If *Images* is unassigned, no images are displayed on the status panels.

Data Type: TCustomImageList

Panels

This property returns the collection *TfcStatusPanels*, which contain the individual collection items (*TfcStatusPanel*) of the StatusBar control. *TfcStatusPanels* is indirectly derived from TCollection.

TfcStatusPanels has a default array property, so each collection can be referenced directly through *panels* using standard array notation. (i.e. `fcStatusBar1.Panels[i]`). See the documentation on *TfcStatusPanel* for information on the properties and methods of TfcStatusPanel.

Data Type: TfcStatusPanels

Added Events

OnDrawPanel

See TStatusBar's *OnDrawPanel* event.

OnDrawKeyBoardState

Event occurs before drawing the text for a panel whose style property is set to *psOverwrite*, *psCapsLock*, *psScrollLock*, or *psNumLock*. Used to customize the text that is displayed. The parameters for this event are as follows:

StatusBar	The statusbar that is displaying the keyboard information.
StatusPanel	The panel that is displaying the keyboard information.
KeyIsOn	<i>True</i> if the corresponding key is enabled. <i>False</i> , otherwise.
Rect	The rectangle of the current status panel.
AText	Assign <i>AText</i> to your own customized string you wish to display in the panel.

Added Methods

TfcStatusPanels methods (access via Panels property)

Add

Adds another panel to the StatusBar. Returns the newly created TfcStatusPanel.

```
function Add: TfcStatusPanel;
```

PanelByName

Returns the *Panel* that is named AName.

```
function PanelByName (AName: string): TfcStatusPanel;
```

GetPanelFromPt

Returns the TfcStatusPanel located at the specified x and y coordinates. Passing in (-1, -1) for x,y will return the panel beneath the current cursor position, otherwise it returns the panel that has the passed in point in its area.

```
Function GetPanelFromPt (x, y: Integer):TfcStatusPanel; virtual;
```

Invalidate

Call this method when you want to refresh the statusbar.

```
Procedure Invalidate; override;
```

How To

Display long hints of any control in the TfcStatusBar

When using the *psHint Style* of a TfcStatusPanel, TstClass captures all hint messages that are sent to the application and displays it without writing a single line of code. If there is a long hint attached to the message, then the long hint will be displayed. Otherwise, the short hint will be displayed. To assign a long hint to a control, set its *hint* property using the following format. 'Short Hint | This is the long hint part of a hint'. This includes menu hints.

Change the display format of the date or time styles

There are two ways of approaching this. One is to use the *OnDrawText* event to reformat a date, time, or datetime text. The other alternative is to set the Delphi global variables *ShortDateFormat* and *ShortTimeFormat* properties. The example below will make all dates use the *LongDateFormat* variable that defaults to display dates in the following format: Thursday, March 11, 1999.

```
procedure TForm1.FormShow(Sender:TObject);  
begin  
    ShortDateFormat := 'dddddd';  
    ShortTimeFormat := 'h:mm:ss am/pm';  
End
```

For more information on using the format specifiers, see the Delphi documentation for the *FormatDateTime* function.

Change the displayed text of a panel at runtime.

Using the *OnDrawText* event of the TfcStatusPanel, you can override the display of the text of the status panel to whatever format you desire. If you are internationalizing an application, you can easily convert hint strings to different languages using this event. See the example below, if you wish to change the

enabled\disabled look of the Overwrite key to display Insert when the keyboard is in insert mode and Overwrite when the keyboard is in overwrite mode. First double click on a TfcStatusBar and bring up the panels editor and select Panel0 and change the style in the Object Inspector to *psOverWrite*. Then put the following code in it's *OnDrawText* event.

```
procedure TForm1.fcStatusBar1Panels0DrawText(  
    Panel: TfcStatusBarPanel; var Text: String;  
    var Enabled: boolean);  
begin  
    if (Text = 'Overwrite') and not Enabled then begin  
        Enabled := True;  
        Panel.TextOptions.Style := fclsLowered;  
        Panel.TextOptions.ShadeColor := clNone;  
        Panel.Font.Style := [];  
    end  
    else begin  
        Panel.TextOptions.Style := fclsRaised;  
        Panel.TextOptions.ShadeColor := clNone;  
        Panel.Font.Style := [fsBold];  
        Text := 'Insert';  
    end;  
end;
```

Tips

- The *width* property is a string instead of an integer in order to allow you to define individual status panels as being a certain percentage of the StatusBar's width. For example, you may wish to have the main text panel always be half the size of the StatusBar then you would set the first panels width to '50%'. Then when or if the statusbar is resized this panel will be sized accordingly.

TfcStatusPanel (Class)

Each Panel in the StatusBar has its own *TfcStatusPanel*. This class is indirectly derived from *TCollectionItem*. In addition to the properties and method found in that class, the following are added.

Ancestor

TCollectionItem
 TfcCollectionItem
 TfcStatusPanel

Added Properties

Bevel

Determines the type of bevel the panel has. Default is *pbLowered*.

Data Type: TfcStatusPanelBevel

Valid Values: pbNone, pbLowered, pbRaised

Col (Runtime only)

When the *style* is set to *psRichEditStatus* and the *Component* is assigned to a RichEdit control, then this read only runtime property contains the current column number in the richedit control.

Data Type: Integer

Color

Use this property to set the *Color* of this status panel.

Data Type: TColor

Component

When the style property is set to *psControl* then this is the control that gets displayed in the status panel. Setting this property to any control besides a richedit control will set the *Style* property to *psControl*. Otherwise the *Style* will be set to *psRichEditStatus* automatically. See *Style* for more information on this setting.

Data Type: TControl

Enabled

When set to *False*, this property sets the panel's state to disabled which will disable the text, glyph, or control embedded in the status panel.

Data Type: boolean

Font

Use the *Font* property to set the default font properties of the StatusPanel's text.

Data Type: TFont

Hint

The hint that gets displayed when the mouse cursor remains over the panel for a short period of time. If the *Control* property is assigned, then you will need to manually assign the Control's *Hint* and *ShowHint* properties.

Data Type: String

ImageIndex

Determines which images appears on the StatusPanel when the *Style* is set to *psGlyph* and an imagelist is assigned to the StatusBar's *Images* property.

Data Type: Integer

Indent

When the *Style* property is set to *psTextOnly*, this property specifies how much from the border the text is spaced away. When the *Alignment* property is *taLeftJustify*, the *Indent* property pertains to the left border.

Data Type: Integer

Margin

This property specifies how much of a margin to put around an attached control.

Data Type: Integer

Name

Use this property to set the name of this status panel. This property is used with the *PanelByName* method of the TfcStatusPanels.

Data Type: String

PopupMenu

Use this property to assign a specific *PopupMenu* to an individual status panel.

Data Type: TPopupMenu

Style

Determines the type of information the status panel displays. There are a large number of different styles to choose from.

Data Type: TfcStatusPanelStyle

Valid Values: psTextOnly, psControl, psOverWrite, psCapsLock, psNumLock, psDateTime, psDate, psTime, psGlyph, psRichEditStatus, psHint, psUserName, psComputerName, psScrollLock

psCapsLock, psNumLock, psOverWrite, psScrollLock

These keyboard styles will display the current state of the keyboard. For example, if your caps lock key is off, then the text displayed in the panel will be disabled.

psComputerName, psUserName

These styles will display computer or user names in the status panel. The *psComputerName* style retrieves the computer name of the current system. This name is established at system startup, when it is initialized from the registry. The *psUserName* style retrieves the user name of the current thread. This is the name of the user currently logged onto the system.

psDate, psTime, psDateTime

Setting the *Style* property to one of these settings will cause the status panel to display date, time, or datetime information. A built-in timer automatically refreshes this information.

psHint

This powerful *Style* will save you a lot of time trying to display hints in the statusbar. Just assign the *Style* property to psHint and all hint messages from menus or controls will be captured and displayed in this status panel. The panel will display the hint regardless of the *ShowHint* property of the menu item or control. The status panel uses the long hint portion of the *Hint* string if it is defined. For more information on hint strings see the Delphi documentation on the *hint* property of TApplication.

psRichEditStatus

This *Style* is used to automatically display line and column information in a TRichEdit component. Just assign the *Component* property to a TCustomRichEdit descendant and this panel will display this information. Use the *OnDrawText* event to format the text in any way that you wish.

psTextOnly

This is the default style setting. When set to *psTextOnly* the status panel will display the contents of the *Text* property.

psControl

See the *Component* property. This style will embed any component that descends from TControl and not TCustomRichEdit. The embedded component needs to be assigned to the *Component* property of the TfcStatusPanel.

TextOptions

See the documentation on *TfcText* for a description on how to customize the display of the default text.

Width

Determines how wide the status panel is. This value is a string but can be only one of two types of values.

It can be a string containing an integer (i.e. 5, 25, 30, etc.)

It can be a string containing a percentage (i.e. 5%, 50%, 66%, etc.)

Data Type: String

Added Events

OnDrawText

Event occurs immediately prior to drawing text onto the panel. The supplied *Text* variable can change to override the text that gets drawn. The parameters for this event are as follows:

Panel The status panel that is currently being painted to.

Text Set *Text* to change the display text for this status panel.

OnTextChanged

Occurs immediately after the text for the panel changes. Use this event to update other panels or controls when the state has changed in the status panel that caused this panel's text to change. The parameters for this event are as follows:

Panel The *TfcStatusPanel* that is currently being painted to.

Text The new text value for the status panel.

Added Methods

GetRect

Returns the Rectangle of the Panel.

```
function GetRect: TRect;
```

TfcText (Class)

This class is used by many 1stClass components to alter the appearance of the text that is shown within the control. It is a collection of properties that can vary the text from appearing with an outline to extruding from the form.

Ancestor

TPersistent
TfcText

Added Properties

Alignment

Specifies whether the text is drawn against the left margin, the right margin, or centered.

Note: This property is only valid if *Rotation* is 0.

Data Type: TAlignment

Valid Values: taLeftJustify, taRightJustify, taCenter

DisabledColors

Specifies what colors to use for disabled text.

HighlightColor

Specifies what color to use to paint the highlight. The default is *clBtnHighlight*.

Data Type: TColor

ShadeColor

Specifies what color to use to paint the shadow. The default is *clBtnShadow*.

Data Type: TColor

DoubleBuffered

This property determines whether the label is painted to a temp bitmap before it is displayed. Often (with certain TextOptions settings like extrusions, word-wrapped or full-justified text), you may wish to set this to True.

Data Type: Boolean;

ExtrudeEffects

Specifies what kind of extrusion effects are applied to the text.

Depth

Specifies how far, in pixels, the text extrudes out.

Data Type: integer

Enabled

This property must be *True* for any extrusion effects to be applied.

Data Type: boolean

FarColor

Set this to the color you want the extrusion to blend to. This is the portion of the extrusion farthest away from the text.

Data Type: TColor

NearColor

Set this to the color you want the extrusion to blend from. This is the portion of the extrusion nearest to the text.

Data Type: TColor

Orientation

Specifies the orientation of the extrusion. The mnemonic type indicates the direction of the extrusion from the reference point of the text.

Data Type: TfcOrientation

Valid Values: fcTopLeft, fcTopRight, fcBottomLeft, fcBottomRight, fcTop, fcRight, fcLeft, fcBottom;

Striated

If *True*, each layer of the extrusion will have alternating colors slowly blending from *NearColor* to *FarColor*.

Data Type: boolean

HighlightColor

This property is only applicable when *Style* is set to *fclsRaised* or *fclsLowered*. It determines what color is used to provide the highlight. By default, this is *clBtnHighlight*.

Data Type: TColor

LineSpacing

Specifies the distance between lines. This property is only applicable when the text takes up more than one line.

Data Type: Integer

Options

This property contains a set of optional flags that can be set to change the behavior of the text drawing.

toShowAccel

If *True*, all ampersands (&) will be used to underline the following character.

toShowEllipsis

If *True*, and the text will not fit within its boundaries, ellipsis will be appended to the end of the text to indicate that there is more, unseen text.

toFullJustify

If *True*, then the text in the control will be fully justified on the right and left edges of the text with appropriate padding.

Warning: this operation can be expensive so use this property with caution. You may wish to set `DoubleBuffered` to `True` when using this property.

OutlineColor

This property is only relevant when *Style* is set to *fclsOutline*. It controls what color the outline around the text appears in.

Data Type: TColor

Rotation

This property determines at what angle the text will be drawn. The center point for the rotation is at the center of the text. The angle is in degrees, so 0 is normal, and 180 is upside-down. This property is only valid for True-type fonts.

Data Type: Integer

ShadeColor

This property is the opposite of *HighlightColor*. It determines the color the shadow appears in when *Style* is either *fclsRaised* or *fclsLowered*.

Data Type: TColor

Shadow

This is a set of properties that control the appearance of the shadow for the text.

Color

This property determines what color the shadow will be drawn in.

Data Type: TColor

Enabled

This property determines whether or not the shadow will be drawn.

Data Type: boolean

XOffset, YOffset

Determines how much the shadow will be offset from the rest of the text. If these values are both zero, then the shadow will appear directly beneath the regular text.

Data Type: Integer

Style

This property controls what style will be used when painting the text.

fclsDefault

Text is drawn in the normal fashion.

fclsLowered

Text appears engraved—lowered into the form. It is often desirable to set the `ShadeColor` property to `clNone` in this style.

fclsOutline

The text has a one pixel outline surrounding it.

fclsRaised

The text appears to stick out in bas-relief from the form. It is often desirable to set the `ShadeColor` property to `clNone` in this style.

VAlignment

This property determines whether the text will be drawn along the top margin, the bottom margin, or centered between the two.

Data Type: TfcVAlignment

Valid Values: vaTop, vaVCenter, vaBottom

WordWrap

When this property is *True*, lines will be wrapped when they are longer than the width of the space allotted them.

Data Type: boolean

How To

Add a shadow to the text

Set the `Shadow.Enabled` property to *True*. To move the shadow's placement underneath the text, modify the `XOffset` and `YOffset` properties of `Shadow`.

Tips

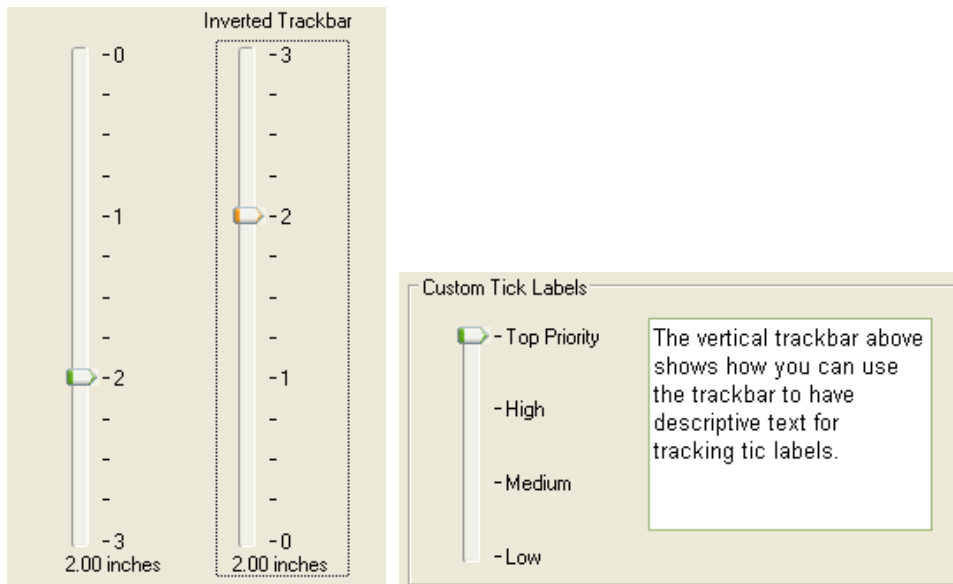
- When using the *fclsRaised* or *fclsLowered* styles with a smaller font size, you may wish to set the *TextOptions.ShadeColor* to *clNone*. Doing so will result in a more optimal look for smaller font sizes.

TfcTrackBar



Use the 1stClass TfcTrackBar to put a track bar on a form. A track bar represents a position along a continuum using a slider and, optionally, tick marks. A track bar can also display a selected range marked by triangular ticks at the starting and ending positions of the selection.

During program execution, the slider can be moved to the desired position by dragging it with the mouse or by clicking the mouse on the bar. To use the keyboard to move the slider, press the arrow keys or the Page Up and Page Down keys.



The following details some of the functionality of the trackbar control

- The 1stClass Trackbar can be bound to a database field (float or integer) or used without a database.
- Use the *Min*, *Max* properties to control the range of the tracking
- Use the *TextAttributes* property to control the way text is painted within the trackbar
- Use the *Inverted* property to flip the trackbar so that the highest value is on the other side of the trackbar.
- Use the *TickMarks*, *OnDrawTickText* event, and *TickStyle*, *TextAttributes.TickDisplayFormat*, and *TextAttributes.TickLabelFrequency*,

Frequency, and the *OnDrawTickText* event to customize the way tick labels are painted.

- Use the *ThumbLength*, *ThumbThickness*, *TrackThumbIcon*, *ThumbColor* to customize the way the thumb is painted.
- Use the *PageSize* and *Increment* properties to control how much the trackbar is moved during PageUp, PageDown, vk_up, and vk_down keyboard entries.

Ancestor

TWinControl
 TCustomControl
 TCustomPanel
 TfcCustomPanel
 TfcTrackBar

Added Properties

DataField

Optional: This property contains the name of the field that you want to bind the TfcTrackbar to. If you do not wish to bind the control to a table field, then leave both the *Datafield* and *Datasource* properties blank. The default value is blank (unbound).

Data Type: String

DataSource

Optional: This property contains the name of a TDataSource component that provides the TfcTrackBar control with data. The default value is blank (unbound).

Data Type: TDataSource

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*.

Frequency

Specifies the relative increment between tick marks on the track bar. For example a frequency of 2.0 will put a tick mark at every 2.0 units.

Data Type: Double

Increment

Assign this property to change how much the tracking thumb position is modified when the arrow keys are used. The position is also rounded to a multiple of the increment value.

Data Type: Double

Inverted

Set this property to true to invert the trackbar so that the starting value is on the right instead of the left (horizontal trackbar), or the bottom instead of the top (vertical trackbar).

Max

Specifies the maximum Position of a TTrackBar.

Data Type: Double

Min

Specifies the minimum Position of a TTrackBar.

Data Type: Double

Orientation

When *Orientation* is set to *trfcVertical* then the top of the trackbar is the starting drag position. When *Orientation* is set to *trfcHorizontal* then the left of the trackbar is the starting drag position.

PageSize

Assign this property to change how much the tracking thumb position is modified when the PageUp, PageDown keys are used.

Position

Contains the current position of the slider of a TfcTrackBar.

ReadOnly

Assign this property to prevent the user from changing the value/position of the trackbar.

SelEnd

Assign this property if you wish to display a selected range within the trackbar. This property represents the upper value of the selection. Note: You can specify fractional values for the increment (i.e. 0.5).

Data Type: Double

SelStart

Assign this property if you wish to display a selected range within the trackbar. This property represents the lower value of the selection. See also the *SelEnd* property.

SliderVisible

Assign this property to False to disable the painting of the trackbar thumb

SpacingEdgeTrackbar

Assign this property to change the spacing (pixels) between the edge of the trackbar's boundary and the (top-vertical trackbar, left - vertical trackbar) of the trackbar's dragging rectangle.

SpacingLeftTop

Assign this property to change the spacing (pixels) between the edge of the trackbar's boundary and the (left-horizontal trackbar/top-vertical trackbar) of the trackbar's dragging rectangle.

SpackingRightBottom

Assign this property to change the spacing (pixels) between the edge of the trackbar's boundary and the (right-horizontal trackbar, bottom-vertical trackbar) of the trackbar's dragging rectangle.

TextAttributes

This property allows you to configure the way text is painted in the trackbar. The can display the current track position, and can also display labels next to the tick marks.

This property contains the following sub-properties.

Data Type: TfcTrackBarText

Position Assign this property to control where the text containing the current track position is displayed. Valid values: *tbtLeft*, *tbtRight*, *tbtTop*, *tbtBottom*.

Data Type: TfcTrackBarTextPosition

OffsetX Assign this property to specify the number of pixels to shift the text left or right from its default position.

OffsetY Assign this property to specify the number of pixels to shift the text up or down from its default position.

Font Assign this property to customize the font of the text displayed.

DisplayFormat Assign this property to format the text displaying the position. See the Delphi TNumericField.DisplayFormat property for more details on the format string.

ShowText Set this property to true to display a formatted string revealing the current position

TickLabelFrequency

Assign this property to customize the frequency of the text labels that appear next to the tick marks. The default of 0 means that no tick labels are drawn.

TickDisplayFormat

Assign this property to format the text that appears next to a tick mark See the Delphi TNumericField.DisplayFormat property for more details on the format string.

ThumbColor

Assign this property to change the thumb color. If you are using themes, then this property is ignored and instead the theme paints the thumb

ThumbLength

Assign this property to change the length of the thumb. For a vertical trackbar, this property changes the width of the thumb instead of its height. This property will also change the dimensions of the tracking rectangle.

ThumbThickness

Assign this property to change the thickness of the thumb. For a vertical trackbar, this property changes the height of the thumb instead of its height. This property will also change the dimensions of the tracking rectangle.

TickMarks

Set this property to determine where the tick marks are drawn.

tmfcBottomRight indicates that the tick marks are drawn below (horizontal orientation), or to the right (vertical orientation) of the tracking rectangle.

tmfcTopLeft indicates that the tick marks are drawn above (horizontal orientation), or to the left (vertical orientation) of the tracking rectangle.

tmfcBoth indicates that tick marks are drawn on both sides of the tracking rectangle.

TickStyle

Set *TickStyle* to specify whether the track bar should display tick marks, and if so, how those tick marks are set. *TickStyle* has these possible values:

tsAuto Tick marks are automatically displayed at increments equal to the value of the Frequency property.

tsManual Tick marks are displayed at the Min and Max values. Additional tick marks can be set using the *SetTick* method.

tsNone No tick marks are displayed.

TrackColor

Assign this property to change the color of the tracking rectangle. If themes are enabled, then this property is ignored, and instead the theme paints the tracking rectangle.

TrackPartialFillColor

Assign this property to change the color of the progress rectangle. This rectangle is drawn from the starting edge of the tracking rectangle to the current position.

TrackThumbIcon

Assign this property to use a custom glyph for the thumb

Data Type: TBitmap

Added Events

OnDrawTickText

Use this event to customize how the tick labels are painted. The parameters are defined as follows.

Sender: TObject TfcTrackBr control that is associated with this event.

<i>TickValue</i> : Double	This is the value associated with the tick label
Var <i>ATickText</i> : String	Assign this property to change the text of the tick labels
Var <i>ARect</i> : TRect	This is the default painting rectangle used to paint the tick label
Var <i>DoDefault</i> : Boolean	Set this property to False to disable the default painting of the tick label.

Example: Using this event you can display text descriptions for tick marks instead of numbers. The following example maps the numbers 0, 1, 2, and 3 to the strings *Low*, *Medium*, *High*, and *Top Priority*.

```

procedure TTrackbarForm.fcTrackBar6DrawTickText(Sender:
TObject;
  TickValue: Double; var ATickText: String; var ARect: TRect;
  var DoDefault: Boolean);
var TickInt: integer;
begin
  TickInt:= Round(TickValue);
  if (abs(TickValue-TickInt)<0.01) then
  begin
    case TickInt of
      0: ATickText:= 'Low';
      1: ATickText:= 'Medium';
      2: ATickText:= 'High';
      3: ATickText:= 'Top Priority';
    end;
  end;
end;

```

OnChange

Write an OnChange event handler to take specific action whenever the position of the slider may have changed. For example, if the track bar is being used to control another object, update the other object from an OnChange event handler.

TfcTreeCombo



Use the 1stClass TreeCombo to hierarchically organize and display items in a drop-down list. Similar to the Image Combo found in the Windows Explorer Desktop combo, the component supports most of the functionality found in the 1stClass TfcTreeView, including the display of images.



TfcTreeCombo Screen shot

The drop-down list's nodes and their attributes are defined through the controls *Items* property. These nodes become the drop-down list when the program is executed. Clicking on the *Items* property at design-time brings up the nodes property editor. See the TfcTreeView documentation for information on using the nodes design-time property editor.

To configure the display properties of the drop-down list use the *TreeOptions* property. See the TfcTreeView *Options* property for more information on each option.

InfoPower support: If you are also using Woll2Woll's InfoPower, you can embed the TfcTreeCombo into InfoPower's grid and record-view components. The steps for doing this are the same as with any InfoPower control. See the InfoPower documentation for more information on attaching a custom control to its grid or record-view. See the how-to topics at the end of this section for information on displaying the images of a TreeCombo for all rows in an InfoPower grid.

Use the *ShowMatchText* property to enable incremental searching as the user types. This option also updates the control's display so that the matching text is displayed in the control.

Set the *Style* property to *csDropDownList* to force the entry to come from the list. If *AllowClearKey* is *False*, then the user is not permitted to clear an existing entry.

To restrict the user to only being able to select terminal nodes, set the *Options | icoEndNodesOnly* property. To prevent the drop-down tree from being initially expanded set the *Options | icoExpanded* property to *False*.

Ancestor

TWinControl
 TCustomEdit
 TfcCustomCombo
 TfcCustomTreeCombo
 TfcTreeCombo

Added Properties

Anchors, AutoSelect, AutoSize, BorderStyle, Constraints, and HideSelection

These properties are equivalent to the properties of the same name found in *TEdit*. See the Delphi / C++ Builder docs under *TEdit* for more information on these properties.

AlignmentVertical

The value of this property determines how the Text in the treecombo will be aligned in the combo vertically. This property defaults to *fcavTop*.

Data Type: TfcAlignVertical

Valid Values: fcavTop, fcavCenter

AllowClearKey

When the 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. The default value is *False*.

Data Type: boolean

Valid Values: True or False

ButtonEffects

See TfcButtonEffects for information on this property.

Data Type: TfcButtonEffects

ButtonGlyph

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


Data Type: TBitmap


ButtonStyle

Select the icon to use for this component.

Data Type: TfcComboButtonStyle

Valid Values: cbsEllipsis, cbsDownArrow, cbsCustom

cbsDownArrow The  bitmap is displayed

cbsEllipsis The  bitmap is displayed

cbsCustom: The icon defined by the *ButtonGlyph* property.

ButtonWidth

Determines the width of the Button. Set to zero for the default button width.

Data Type: Integer

Controller

See InfoPower TwwController property

DataField

Optional: This property contains the name of the field that you want to bind the TfcTreeCombo to. If you do not wish to bind the treecombo to a table field, then leave both the *Datafield* and *Datasource* properties blank. The default value is blank (unbound).

Data Type: String

DataSource

Optional: This property contains the name of a TDataSource component that provides the TreeCombo control with data. The default value is blank (unbound).

Data Type: TDataSource

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

The *DropDownCount* property determines how many entries will appear in the dropdown control.

Data Type: Integer

DropDownWidth

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

Data Type: Integer

Frame

See *TfcEditFrame* for more information on this property.

Data Type: *TfcEditFrame*

Images

Images contains a list of images that can appear in the combo. Each node's *ImageIndex* property determines the specific image displayed for the node. If *Images* is unassigned, no images are displayed in the combo.

Data Type: *TCustomImageList*

Items

See the *Items* property of *TfcTreeView*.

Options

This property contains a set of boolean values that control the behavior of the *TreeCombo*.

Data Type: Set of *TfcImgComboOption*

Valid Values: *icoExpanded*, *icoEndNodesOnly*

icoExpanded

Setting *icoExpanded* to *True* expands all nodes in the drop-down list when the *treecombo* is dropped down.

icoEndNodesOnly

Set to *True* to restrict the user's selection to allows only nodes without children to be selectable. Setting this property to *False* allows any node to be selectable. See also the *TreeView Nodes* editor, as it allows you set a selectable property for each node in the drop-down list. See also the *OnCheckValidItem* event to programmatically determine if a node is selectable.

SelectedNode (Runtime and ReadOnly)

Reference this property to retrieve information about the node selected from the drop-down list. See the *TfcTreeView* documentation for more information on the *TfcTreeNode* type.

Data Type: TfcTreeNode

ShowMatchText

When this property is set to *True*, the TreeCombo will perform ‘Quicken’ style incremental searching. As the user enters text, the control will simultaneously search and display the matching text in the control. The default value is *True*.

Data Type: boolean

Sorted

Setting this property to *True* will sort the list alphabetically. Once the control’s drop-down items are sorted, the original hierarchy is lost. That is, setting the *Sorted* property back to *False* will not restore the original order of items. The default value is *False*.

Data Type: boolean

StateImages

StateImages contains a list of images that can appear in the combo’s drop-down list. These images only appear in the drop-down list, and do not appear in the combo’s edit control. Use the *Images* property if you wish to display images in both the edit control and the drop-down list. Each node’s *StateIndex* property determines the specific image displayed for the node. If *StateImages* is unassigned, no state images are displayed in the combo’s drop-down list.

Data Type: TCustomImageList

StoreDataUsing

StoreDataUsing allows you to value stored in the database to one of the node properties. Default is *sdStoreText*.

Data Type: TwwStoreData

Valid Values: *sdStoreText*, *sdStoreData1*, *sdStoreData2*

sdStoreText Store the node’s Text property.

sdStoreData1 Store the node’s Data1 property.

sdStoreData2 Store the node’s Data2 property.

Style

This property determines the style of the TreeCombo. 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.

Data Type: TfcComboStyle

Valid Values: *fcCombo.csDropDown*, *fcCombo.csDropDownList*

TreeOptions

See the *Options* property of the TfcTreeView control.

Data Type: TfcTreeViewOptions

Valid Values: tvoExpandOnDbIClk, tvoExpandButtons3D, tvoFlatCheckBoxes, tvoHideSelection, tvoRowSelect, tvoShowButtons, tvoShowLines, tvoShowRoot, tvoHotTrack, tvoAutoURL, tvoToolTips, tvoEditText, or tvo3StateCheckbox

TreeView (Runtime only)

Use this runtime only property to access the dropdown treeview control.

Data Type: TfcTreeView

Added Events

OnCalcNodeAttributes

Use this event to customize how the drop-down nodes are painted. See the TfcTreeView *OnCalcNodeAttributes* event for more information on this event.

OnCheckValidItem

Use this event to define which nodes are selectable from the drop-down list. Normally all nodes are selectable. See also the property *Options | icoEndNodesOnly*, which allows only nodes without children to be selectable. See also the TreeView Nodes editor, as it allows you set a selectable property for each node in the drop-down list. The parameters for this event are as follows:

<i>Sender</i> :TObject	TfcTreeCombo control that is associated with this event.
<i>Node</i> :TfcTreeNode	<i>Node</i> that is being examined to determine if it is selectable.
<i>Accept</i> :boolean	Set this property to <i>False</i> to prevent this node from being selected.

OnCloseUp

This event is fired immediately after the drop-down list closes. Use this event to perform your own custom action after the drop-down list closes. The parameters for this event are as follows:

<i>Sender</i> :TObject	TfcTreeCombo control that is associated with this event.
------------------------	--

Select: boolean This value is *True* if the user is making a selection. If the user is escaping out of the drop-down list without making a selection, then the value of *Select* is *False*.

OnDropDown

This event is fired immediately before the combo drops down the list. The parameters for this event are as follows:

Sender:TObject TfcTreeCombo control that is associated with this event.

OnSelectionChange

This event is fired when the user selects a new value from the drop-down list. If instead you are trying to detect any kind of change to the control's text property, then use the OnChange event. The parameters for this event are as follows:

Sender:TObject TfcTreeCombo control that is associated with this event.

Added Methods

CloseUp

Call this method if you wish to force the drop-down list to close. Set *Accept* to *True* if you would like the control to accept the last selected entry.

```
procedure CloseUp(Accept: boolean); override;
```

DrawInGridCell

Call this method if you wish to use the TreeCombo information to accurately draw the selected item in an InfoPower grid. See the how-to topics at the end of this section for an example of using this method.

```
Procedure DrawInGridCell(ACanvas:TCanvas; Rect:TRect;  
    State:TGridDrawState); override;
```

DropDown

Call this method if you wish to force the drop-down list to display.

```
Procedure DropDown; override;
```

IsDroppedDown

Call this method when you want to determine if the dropdown control is visible.

```
Function IsDroppedDown:boolean; override;
```

IsValidNode

Override this method if you wish for your own derived component to define its own criteria for determining which nodes are selectable. If you are not subclassing you can use the *OnCheckValidItem* event.

```
function IsValidNode (Node: TfcTreeNode) : boolean; virtual;
```

SetSelectedNode

Call this method if you wish to set the SelectedNode property programmatically. Pass Nil to clear the selectednode property.

```
procedure SetSelectedNode (Node:TfcTreeNode); virtual;
```

How To

Creating a non-hierarchical combobox containing images

To create a combo that displays a non-hierarchical list of items with images, just attach a TImageList to the tree combo using the *Images* property, and set the image index to the appropriate image for each item that you add. Do not add sub-items unless you also want a hierarchical drop-down list. In addition, remove *tvoShowLines* and *tvoShowRoot* from the *TreeOptions* property.



Non-hierarchical list of items

Make Only End Nodes Selectable

Set the *Options | icoEndNodesOnly* property to *True*.

Iterating through a list of nodes in the DropDown Treeview.

You can iterate through the nodes by accessing the *Items* property and referencing the TfcTreeView methods. For instance, the following code iterates through the list by using the *GetFirstNode* method to get the first node in the tree, in conjunction with iterative use of the *GetNext* method of the TfcTreeNode.

```
var Node: TfcTreeComboTreeNode;
```

```

begin
  Node :=
    TfcTreeComboTreeNode(fcTreeCombo1.Items.GetFirstNode);
  while Node<>Nil do
    begin
      { Perform your action here }
      Node := TfcTreeComboTreeNode(Node.GetNext);
    end;
  end;
end;

```

Initializing an unbound TfcTreeCombo selection.

If the combo is not tied to a datasource, you can initialize it by setting the control's *Text* property. If the combo is tied to a datasource, it will take on the value of the field in the TDataSet.

Display the images of the TfcTreeCombo in all rows of an InfoPower Grid.

When you embed the TreeCombo containing images into an InfoPower grid, the image for the control is displayed in the control, but not necessarily for the other rows of the grid. In InfoPower 3000 you can just check the Control Always Paints checkbox in the Edit Control tab page of the Grid's Selected property dialog.

For InfoPower 2000, the following code allows you to paint the image in all the rows of the grid for the column containing the TreeCombo.

```

procedure TForm1.wwDBGrid1DrawDataCell(Sender: TObject;
  const Rect: TRect; Field: TField; State: TGridDrawState);
var Control: TfcCustomCombo;
begin
  Control := fcGetControlInGrid(self,
    Sender as TwwDBGrid, Field.FieldName);
  if Control <> nil then Control.DrawInGridCell((Sender as
    TwwDBGrid).Canvas, Rect, State);
end;

```

Using the TreeView Items Editor

Refer to the TfcTreeview documentation for a reference to this property editor.

TfcTreeNode (Class)

TfcTreeNode describes an individual node in a tree view control. Each node in a tree view control consists of a number of attributes, and can itself contain 0 or more nodes.

TfcTreeNode is not a design time component you see in your IDE palette, but is created and manipulated internally by the TfcTreeView.

Ancestor

TPersistent
TfcTreeNode

Properties

AbsoluteIndex

AbsoluteIndex is the index of the tree node relative to the first tree node in a tree node.

Use *AbsoluteIndex* to determine the absolute position of a node in a tree nodes object. The first tree node in a tree nodes object has an index of 0 and subsequent nodes are numbered sequentially. If a node has any children, its *AbsoluteIndex* is one less than the index of its first child.

Data Type: integer

CheckboxType

This property returns the type of checkbox being used for the node.

tvctNone Node does not use a checkbox or radio button

tvctCheckbox Node is displayed as a checkbox

tvctRadioGroup Node is displayed as a radio button

Data Type: TfcTreeViewCheckboxType = (tvctNone, tvctCheckbox, tvctRadioGroup);

Checked

Checked indicates if a node's checkbox is currently checked

Data Type: boolean

Count

Use *Count* to determine how many child nodes belong to a tree node. *Count* includes only immediate children, and not their descendants.

Data Type: integer

Cut

Use *Cut* to alter the appearance of the tree node when its *Cut* property is set to *True*. When *Cut* is *True*, the Treeview blends 50% white with any image defined by the *ImageIndex*. This allows the node to appear as if it is marked for removal. The following code attached to the TreeView's *OnKeyDown* event will set the node's *Cut* property to *True* when Ctrl-X is pressed.

```
procedure TForm1.fcTreeView1KeyDown(Sender: TObject; var Key:
Word;
  Shift: TShiftState);
begin
  with (Sender as TfcTreeView) do begin
    if (ssCtrl in Shift) and (Key=ord('X')) and
      (Selected<>nil) then Selected.Cut:= True;
  end
end;
```

Note: The nodes are not actually removed. If you wish to perform some action on the cut nodes at some later time, then you will need to iterate through the nodes in the tree.

Data Type: boolean

Data

Data is a pointer to application-defined data associated with the tree node. Use the *Data* property to associate data with a tree node. *Data* allows applications to quickly access information about the entity represented by the node. Your own code is responsible for allocating memory for data as well as freeing this memory.

Note: For convenience, the *TfcTreeNode* already allocates two string variables into which you can store your own custom data. See the properties *StringData* and *StringData2*. If your needs for storage go beyond this, you will need to use the *Data* property.

Data Type: Pointer

Deleting

Deleting indicates whether a node's *Destroy* method has been called and it is in the process of being deleted.

Use *Deleting* to avoid recursively trying to delete a node in response to events that occur as a result of the node being deleted.

Data Type: boolean

DropTarget

Use *DropTarget* to indicate that the node is a drag and drop target. When *DropTarget* is *True*, the node is drawn in a style used to indicate a drag and drop target.

Note: Setting *DropTarget* to *True* does cause the node to automatically accept dragged objects when they are released. The application must still implement the drop behavior.

Data Type: boolean

Expanded

Expanded specifies whether the tree node is expanded.

When a tree node is expanded, the minus button is shown if the *ShowButtons* property of the tree view is *True* and child nodes are displayed. Set *Expanded* to *True* to display the children of a node. Set *Expanded* to *False* to collapse the node, hiding all of its descendants.

Data Type: boolean

Focused

Focused indicates whether the node appears to have focus.

Tree view nodes are not windowed controls and so can't receive input focus. However, the user can edit them when the tree view control has focus. When *Focused* is *True*, the node is surrounded by a standard focus rectangle and the user can edit the label. Use *Focused* to determine if the user can currently edit a particular node in a tree view.

Data Type: boolean

Grayed

Grayed indicates if a node's checkbox background is currently grayed. See also *Options | two3StateCheckbox* to allow checkboxes to cycle between the following 3 states (Unchecked, Checked, Checked | Grayed).

Data Type: boolean

Handle

Handle is the window Handle of the tree view that contains the node.

Use *Handle* to obtain the handle of the tree view that owns the node. This handle can be passed as a parameter to Windows API function calls that require a handle to the tree view.

Data Type: HWND

HasChildren

HasChildren indicates whether a node has any children.

HasChildren is *True* if the node has subnodes, or *False* if the node has no subnodes. If *ShowButtons* of the tree view is *True*, and *HasChildren* is *True*, a plus (+) button will appear to the left of the node when it is collapsed, and a minus (-) button will appear when the node is expanded.

If a node has no children, setting *HasChildren* to *True* will show a (+) plus button, but will not add any child nodes and the node cannot be expanded.

Data Type: boolean

ImageIndex

ImageIndex specifies which image is displayed when a node is in its normal state and is not currently selected.

Note: If the *ImageIndex* property is set to -1 , then no image is displayed but the text is left-aligned as if there were an image present. If the *ImageIndex* is set to -2 then the text is shifted to the left to take into account the image not being painted. Setting *ImageIndex* to -2 is useful to remove the whitespace preceding the node's text.

Data Type: integer

Index

Index specifies the position of the node in the list of child nodes maintained by its parent node.

Use *Index* to determine the position of the node relative to its sibling nodes. The first child of the parent node has an *Index* value of 0, and subsequent children are indexed sequentially.

Data Type: longint

IsVisible

IsVisible indicates whether the tree node is currently visible in the tree view image.

A node is visible if it is on level 0 or if all its parents are expanded. *IsVisible* indicates whether the node is part of the current tree view image. It does not indicate whether or not the node is scrolled into view when the tree view image is larger than the size of the tree view control.

Data Type: boolean

Item

Item provides access to a child node by its position in the list of child nodes.

property `Item[Index: Integer]: TfcTreeNode;`

Use *Item* to access a child node based on its *Index* property. The first child node has an index of 0, the second an index of 1, and so on.

ItemID

ItemID is a handle of type `HTreeItem` and uniquely identifies each node in a tree view.

Use this property to reference the nodes when making Windows API calls or calling the *GetNode* method of the `TfcTreeNode`s that owns the item.

Data Type: `HWND`

Level

Level indicates the level of indentation of a node within the tree view control.

The value of *Level* is 0 for nodes on the top level. The value of *Level* is 1 for their children, and so on.

Data Type: integer

MultiSelected

This property returns *True* if the node is currently part the multi-selected list in the treeview. You can also assign this property to *True* to multi-select a node.

Data Type: boolean

OverlayIndex (Runtime)

OverlayIndex determines which image from the image list (*Images* property) is used as an overlay mask. An overlay mask is an image drawn transparently over another image in the tree view. For example, to indicate that a node is no longer available, use an overlay image that puts an X over the current node's image. See the Delphi *TImageList Overlay* method to define overlay images. You will also need to use the *TfcTreeView's OnCalcNodeAttributes* method to define the overlay index for a node.

Data Type: integer

Owner

Owner indicates which `TfcTreeNode`s object contains the tree node.

Use the *Owner* property to determine the owner of the tree node. Do not confuse *Owner* with the *TreeView* property. *Owner* is the list of nodes used by the tree view to manage its nodes. The *TreeView* property is the tree view that uses that list.

Data Type: `TfcTreeNode`s

Parent

Parent identifies the parent node of the tree node. A parent node is one level higher than the node and contains the node as a subnode.

Data Type: TfcTreeNode

Selected

Selected determines whether the node is the active node in the tree.

Set *Selected* to *True* to select the node. The appearance of a selected node depends on whether it has the focus and on whether the system colors are used for selection.

When a node becomes selected, the tree view's *OnChanging* and *OnChanged* events are triggered.

Note: This property does not relate to multi-selection. If you wish to check if a node is multi-selected, use the *MultiSelected* property.

Data Type: boolean

SelectedIndex

SelectedIndex is the index in the tree view's image list of the image displayed for the node when it is selected.

Use the *SelectedIndex* property to specify an image to display when the tree node is selected.

Data Type: integer

StateIndex

StateIndex indicates which image from the *StateImages* list to display for the node.

Use *StateIndex* to display an additional image for the node that reflects state information. If *StateIndex* is -1 , or a multiple of 16, then no state image is drawn. The reason that a multiple of 16 is not supported is due to Microsoft not supporting this in their *TreeView* common control.

Note: If a checkbox is displayed for the node, then *StateImages* are disabled for the node.

Data Type: integer

StringData

StringData can be used to store your own information into a string. Often you may want to associate other information for a node besides just its text property. This property can be used for your own needs.

Note: If *Options | tvoAutoURL* is set to *True*, the *TreeView* uses *StringData* as a URL link address. In this mode, when *StringData* is not blank, the *TreeView* automatically displays the *HandPoint* mouse cursor when the mouse moves over the URL link, and automatically opens the address when the user clicks on the link.

Data Type: String

StringData2

StringData2 can be used to store your own information into a string. Often you may want to associate other information for a node besides just its *Text* property. This property can be used for your own needs.

Data Type: String

Text

Text is the label that identifies a tree node.

Use *Text* to specify the string that is displayed in the tree view. The value of *Text* can be assigned directly at run-time or can be set within the TreeView Items Editor while modifying the *Items* property of the TfcTreeView component. If the tree view allows users to edit its nodes, read *Text* to determine the value given the node by the user.

Data Type: string

TreeView

TreeView specifies the tree view that displays the node. Use *TreeView* to determine the tree view associated with the tree node.

Data Type: TfcCustomTreeView

Methods

AlphaSort

AlphaSort sorts the node's children alphabetically based on their *Text* property. Call *AlphaSort* to sort the subtree of the tree view alphabetically. If successful, the method returns *True*.

```
function AlphaSort: boolean;
```

Assign

Assign copies the properties of another tree view node.

```
procedure Assign(Source: TPersistent); override;
```

If the *Source* parameter is a TfcTreeNode object, *Assign* copies the properties from the source node. If *Source* is any other type of object, *Assign* calls its inherited method, which copies information from any object that can copy to a tree node in its *AssignTo* method.

Note: While *Assign* copies the *HasChildren* property, it does not copy the child nodes. Be sure to copy any descendants after assigning the properties of another node that has children.

Collapse

Collapses a node.

```
procedure Collapse (Recurse: boolean);
```

When a node is collapsed, all of its subnodes are hidden. The plus button may be displayed, depending on whether the tree view's *ShowButtons* property is set. If *Recurse* is *True*, then all subnodes will be collapsed as well. When the node is next expanded the children will still be collapsed. If *Recurse* is *False*, the child nodes won't be collapsed and the next time the node is expanded, the children will be in the same state as when *Collapse* was called.

CustomSort

Sorts the descendants based using a customized ordering function.

```
type TTVCompare =  
    function (lParam1, lParam2,  
             lParamSort: Longint): Integer stdcall;  
  
function CustomSort (SortProc: TTVCompare;  
                    Data: Longint): boolean;
```

Use *CustomSort* to sort the descendants of a tree view nodes where the sort order is defined by the *SortProc* parameter. The *lParam1* and *lParam2* parameters of the sort procedure can be cast to *TfcTreeNode* objects are compared. The *lParamSort* parameter of the sort procedure is the value of *Data* parameter of *CustomSort*. The sort procedure should return a value less than 0 if *lParam1* should come before *lParam2*, should return 0 if the two values are equivalent, and should return a value greater than 0 if *lParam1* should follow *lParam2*.

If the *SortProc* parameter is nil, the *AlphaSort* method is called.

Note: See also the *TfcTreeView CustomSort* method.

Delete

Destroys the node and all its children.

```
procedure Delete;
```

Use the *Delete* method to delete a tree node and free all associated memory.

DeleteChildren

Deletes all children of the node.

```
procedure DeleteChildren;
```

Use the *DeleteChildren* method to delete all children of a tree node, freeing all associated memory.

DisplayRect

Returns the bounding rectangle for a tree node.

```
function DisplayRect (TextOnly: boolean): TRect;
```

If the *TextOnly* parameter is *True*, the bounding rectangle includes only the text of the node. Otherwise, it includes the entire line that the node occupies in the tree-view control.

EditText

Begins in-place editing of the specified node's text, replacing the text of the node with a single-line edit control containing the text.

function EditText: **boolean**;

Call *EditText* to allow the user to edit the node's label. This method implicitly sets the *Selected* and *Focused* properties to *True*. When *EditText* is called, the tree view's *OnEditing* event is triggered.

EndEdit

Ends the editing of a node's label.

procedure EndEdit (Cancel : **boolean**) ;

Call *EndEdit* to take the node out of edit mode. If the *Cancel* parameter is *True*, all changes made by the user are discarded. If *Cancel* is *False*, *EndEdit* updates the node's *Text* property and the tree view's *OnEdited* event occurs.

Expand

Expands the node to display all child nodes.

procedure Expand (Recurse: **boolean**) ;

When a node is expanded, its immediate subnodes are displayed. The minus '-' button may be displayed, depending on whether the tree view's *ShowButtons* property is set. If *Recurse* is *True*, all descendants of the immediate subnodes are expanded as well.

GetFirstChild

Returns the first child node of a tree node.

function GetFirstChild: TfcTreeNode;

Call *GetFirstChild* to access the first child node of the tree view node. If the node has no children, *GetFirstChild* returns nil.

Note: *GetFirstChild* returns the same value as *Item[0]*.

GetHandle

Returns the Handle property.

function GetHandle: HWND;

Calling *GetHandle* is the same as reading the Handle property.

GetLastChild

Returns the last immediate child node of the calling node.

function GetLastChild: TfcTreeNode;

Call *GetLastChild* to find the last immediate child of a node. If the calling node has no children, *GetLastChild* returns nil. *GetLastChild* returns the same value as `Item[Count-1]`.

GetNext

Returns the next node after the calling node in the tree view.

function `GetNext: TfcTreeNode;`

If the calling node is the last node, *GetNext* returns nil. It will return the next node including nodes that aren't visible and child nodes. To get the next node at the same level as the calling node, use *GetNextSibling*. To get the next visible node, use *GetNextVisible*.

GetNextChild

Returns the next child node after *Value*.

function `GetNextChild(Value: TfcTreeNode): TfcTreeNode;`

Call *GetNextChild* to locate the next node in the list of immediate children of the tree view node. If the calling node has no children or there is no node after *Value*, *GetNextChild* returns nil.

GetNextSibling

Returns the next node in the tree view at the same level as the calling node.

function `GetNextSibling: TfcTreeNode;`

GetNextSibling will return the next node, regardless of whether it's visible. To find the next node in the tree view including child nodes, use *GetNext*.

GetNextVisible

Returns the next visible node in the tree view after the calling node.

function `GetNextVisible: TfcTreeNode;`

Use *GetNextVisible* when iterating through all the visible nodes in the tree view. A node is visible if all its parent nodes are expanded.

GetPrev

Returns the previous node in the tree view before the calling node.

function `GetPrev: TfcTreeNode;`

GetPrev will return the previous node whether or not it is visible. To get the previous visible node, use *GetPrevVisible*.

GetPrevChild

Returns the previous child node before *Value*.

function `GetPrevChild(Value: TfcTreeNode): TfcTreeNode;`

Call *GetPrevChild* to locate the previous node in the list of immediate children of the tree view node. If the calling node has no children or there is no node before *Value*, *GetPrevChild* returns nil.

GetPrevSibling

Returns the previous node before the calling node and at the same level.

```
function GetPrevSibling: TfcTreeNode;
```

GetPrevSibling returns the previous sibling node, regardless of whether it's visible.

To find the previous node in the tree view including all levels, use *GetPrev*.

GetPrevVisible

Returns the previous visible node before the calling node.

```
function GetPrevVisible: TfcTreeNode;
```

To get the previous node, including non-visible, use *GetPrev*.

HasAsParent

Returns *True* if *Value* is a parent node of the calling node.

```
function HasAsParent (Value: TfcTreeNode): boolean;
```

Use the *HasAsParent* method to determine if a node is a parent to a particular node.

IndexOf

Returns the position of an immediate child node of the calling node.

```
function IndexOf (Value: TfcTreeNode): Integer;
```

Call *IndexOf* to obtain the position of a child node among the children of the calling node. If *Value* isn't an immediate child of the calling node, *IndexOf* returns -1. The first child node has an index of 0, the second an index of 1, and so on.

IsRadioGroup

This method returns *True* if the node is part of a radio group

```
Function IsRadioGroup: boolean;
```

MakeVisible

Expands the parent nodes of a node.

```
procedure MakeVisible;
```

If a node's parent node(s) are collapsed and the node isn't visible, *MakeVisible* will expand the node's parents to make the node visible.

MoveTo

Moves the node to another location in the tree view.

```
Type TfcNodeAttachMode = (fcnaAdd, fcnaAddFirst, fcnaAddChild, fcnaAddChildFirst, fcnaInsert, fcnaInsertAfter);
```

procedure MoveTo (Destination: TfcTreeNode;
Mode: TfcNodeAttachMode);

The *Destination* parameter determines where to move the node. The *Mode* parameter is of type *TfcNodeAttachMode* and specifies how the node is to be reattached. These are the possible values for the *Mode* parameter:

Value	Meaning
fcnaAdd	Adds the node to the end of the list.
fcnaAddFirst	Adds the node at the beginning of the list.
fcnaAddChild	Adds node as a child of the destination at end of the child list.
fcnaAddChildFirst	Adds the node as a child at the beginning of the child list of the destination.
fcnaInsert	Insert the node before the destination node.
fcnaInsertAfter	Insert the node after the destination node.

TfcTreeNodes (Class)

TfcTreeNodes maintains a list of tree nodes in a tree view control. The *Items* property of the tree view control is a TfcTreeNodes object and maintains the collection of nodes in the tree view. Nodes can be added, deleted, inserted and moved within the tree view. Access the nodes in the tree view through the *Items* property of the tree view.

Ancestor

TPersistent
TfcTreeNodes

Properties

Count

Count is the number of nodes maintained by the TfcTreeNodes object. Use *Count* to determine the number of tree nodes in the tree view that owns the tree nodes object. *Count* provides an upper bound when iterating through the entries in the *Item* property array.

Data Type: integer

Handle

Handle is the window handle of the tree view control that owns the tree nodes object. Use *Handle* to obtain the handle of the tree view that owns the tree nodes object.

Data Type: HWND;

Item

Item is an indexed array of all tree nodes managed by the TfcTreeNodes object.

```
property Item[Index: Integer]: TfcTreeNode;
```

Use *Item* to access to a node by its position in the tree view. The first node has an index of 0, the second an index of 1, and so on. *Item* is the default property for TfcTreeNodes. This means that the name of the *Item* property can be omitted when indexing into the set of tree nodes. Thus, the line

```
FirstNode := fcTreeView1.Items.Item[0];
```

can be written

```
FirstNode := fcTreeView1.Items[0];
```

Note Accessing tree view items by index can be time-intensive, particularly when the tree view contains many items. For optimal performance, try to design

applications so that they have as few dependencies on the tree view's item index as possible.

Owner

Owner is the tree view control that uses the tree nodes object to implement its *Items* property.

```
property Owner: TfcCustomTreeView;
```

Use the *Owner* property to access the tree view control that displays the nodes maintained by the *TfcTreeNodes* object.

Data Type: *TfcCustomTreeView*

Methods

Add

Adds a new tree node to a tree view control.

```
function Add(Node: TfcTreeNode; const S: string): TfcTreeNode;
```

The node is added as the last sibling of the *Node* parameter. The *S* parameter specifies the *Text* property of the new node. *Add* returns the node that has been added. If the tree view is sorted, *Add* inserts the node in the correct sort order position rather than as the last child of the *Node* parameter's parent.

AddChild

Adds a new tree node to a tree view.

```
function AddChild(Node: TfcTreeNode;  
    const S: string): TfcTreeNode;
```

The node is added as a child of the node specified by the *Node* parameter. It is added to the end of *Node*'s list of child nodes. The *S* parameter specifies the *Text* property of the new node. *AddChild* returns the node that has been added. If the tree view is sorted, *AddChild* inserts the node in the correct sort order position, rather than as the last child of the *Node* parameter.

AddChildFirst

Adds a new tree node to a tree view. Use *AddChildFirst* to insert a node as the first child of the node specified by the *Node* parameter.

```
function AddChildFirst(Node: TfcTreeNode;  
    const S: string): TfcTreeNode;
```

The *S* parameter specifies the *Text* property of the new node. Nodes that appear after the added node are moved down one row and reindexed with valid *Index* values. *AddChildFirst* returns the node that has been added.

AddChildObject

Adds a new tree node containing data to a tree view.

```
function AddChildObject (Node: TfcTreeNode;  
    const S: string; Ptr: Pointer): TfcTreeNode;
```

The node is added as the first child of the node specified by the *Node* parameter. Nodes that appear after the added node are moved down one row and reindexed with valid Index values. The *S* parameter specifies the *Text* property of the new node. The *Ptr* parameter specifies the *Data* property value of the new node. *AddChildObjectFirst* returns the node that has been added.

Note: The memory referenced by *Ptr* is not freed when the tree nodes object is freed.

AddChildObjectFirst

Adds a new tree node containing data to a tree view.

```
function AddChildObjectFirst (Node: TfcTreeNode; const S:  
string;  
    Ptr: Pointer): TfcTreeNode;
```

The node is added as the first child of the node specified by the *Node* parameter. Nodes that appear after the added node are moved down one row and reindexed with valid Index values. The *S* parameter specifies the *Text* property of the new node. The *Ptr* parameter specifies the *Data* property value of the new node. *AddChildObjectFirst* returns the node that has been added.

Note: The memory referenced by *Ptr* is not freed when the tree nodes object is freed.

AddFirst

Adds a new tree node to a tree view.

```
function AddFirst (Node: TfcTreeNode;  
    const S: string): TfcTreeNode;
```

The node is added as the first sibling of the node specified by the *Node* parameter. Nodes that appear after the added node are moved down one row and re-indexed with valid Index values. The *S* parameter specifies the *Text* property of the new node. *AddFirst* returns the node that has been added.

AddObject

Adds a new node containing data to a tree view.

```
function AddObject (Node: TfcTreeNode;  
    const S: string; Ptr: Pointer): TfcTreeNode;
```

The node is added as the last sibling of the node specified by the *Node* parameter. The *S* parameter specifies the *Text* property of the new node. The *Ptr* parameter specifies the *Data* property value of the new node. *AddObject* returns the node that has been added.

Note: The memory referenced by *Ptr* is not freed when the tree nodes object is freed.

AddObjectFirst

Adds a new node containing data to a tree view.

```
function AddObjectFirst(Node: TfcTreeNode;  
    const S: string; Ptr: Pointer): TfcTreeNode;
```

The node is added as the first sibling of the node specified by the *Node* parameter. Nodes that appear after the added node are moved down one row and reindexed with valid Index values. The *S* parameter specifies the *Text* property of the new node. The *Ptr* parameter specifies the Data property value of the new node. *AddObjectFirst* returns the node that has been added.

Assign

Discards any current property information and replaces it with the information from the Source.

```
procedure Assign(Source: TPersistent); override;
```

Use *Assign* to copy information from one tree nodes object to another. If *Source* is any other type of object, *Assign* calls its inherited method, which will copy properties from any object that can copy to a TfcTreeNodes object in its *AssignTo* method.

BeginUpdate

Prevents the updating of the tree view until the *EndUpdate* method is called.

```
procedure BeginUpdate;
```

BeginUpdate prevents the screen from being repainted when new nodes are added, deleted, or inserted. Tree nodes affected by the changes will have invalid Index values until *EndUpdate* is called.

Use *BeginUpdate* to prevent screen repaints and to speed processing time while adding nodes to the tree view.

Note: Calls to *BeginUpdate* are cumulative; for every call to *BeginUpdate* there must be a corresponding call to *EndUpdate*.

Clear

Deletes all tree nodes contained from the list managed by TfcTreeNodes.

```
procedure Clear;
```

Delete

Delete removes a tree node in the tree view specified by the *Node* parameter.

```
procedure Delete(Node: TfcTreeNode);
```

EndUpdate

Re-enables screen repainting and node reindexing that was turned off with the *BeginUpdate* method.

```
procedure EndUpdate;
```

Use the *EndUpdate* method to enable screen updating after *BeginUpdate* has been called. Calls to *BeginUpdate* are cumulative, so calling *EndUpdate* will only update the tree view if every other call to *BeginUpdate* has already been matched by a call to *EndUpdate*.

GetFirstNode

Returns the first tree node in the tree view.

```
function GetFirstNode: TfcTreeNode;
```

Use the *GetFirstNode* method to retrieve the first node in the tree view. *GetFirstNode* returns the value of *Item[0]*.

GetNode

Returns the tree node given the *ItemId* of the tree node.

```
function GetNode(ItemId: HTreeItem): TfcTreeNode;
```

ItemId is a handle to the node in the tree view.

Insert

Inserts a tree node into the tree view before the node specified by the *Node* parameter.

```
function Insert(Node: TfcTreeNode;  
  const S: string): TfcTreeNode;
```

Call *Insert* to add a new sibling to the *Node* parameter, immediately preceding *Node*. The *S* parameter specifies the *Text* property of the new node. *Insert* returns the new node.

InsertObject

Inserts a tree node containing data into the tree view before the node specified by the *Node* parameter.

```
function InsertObject(Node: TfcTreeNode;  
  const S: string; Ptr: Pointer): TfcTreeNode;
```

Call *InsertObject* to add a new sibling to the *Node* parameter, immediately preceding *Node*. The *S* parameter specifies the *Text* property of the new node and the *Ptr* parameter specifies the *Data* property of the new node. *InsertObject* returns the new node.

Note: The memory referenced by *Ptr* is not freed when the tree nodes object is freed.

FindNode

Searches for a node containing the string *SearchText* in the tree. If a match is found the function returns the matching node. Set *VisibleOnly* to *True* to only search the visible nodes. A node is visible if all the parent nodes are expanded.

```
function FindNode(SearchText: string; VisibleOnly: boolean): TfcTreeNode;
```

TfcTreeHeader



Use the 1stClass TreeHeader control to associate a header with a self-referencing TfcDBTreeView control. See the *how-to* topics in the *TfcDBTreeView* for detailed instructions on setting up a self-referencing tree.

To associate a header with a TfcDBTreeView control, assign its *Header* property. This will cause the tree to display the field information in columns of data, defined by the properties assigned to the header control.

Subject	From Name	Date Posted
Embedding non-InfoPower component in a...	taine gilliam	10/16/2000 4:41:12 PM
DateTimePicker in Grid	Ah-Mee Chan	10/16/2000 11:31:30 PM
Date Picker in Filter Dialog	"Michael L. Sorensen"	10/17/2000 12:07:36 AM
Re: Date Picker in Filter Dialog	"Michael L. Sorensen"	10/17/2000 12:43:06 AM
Re: Date Picker in Filter Dialog	"Paul Woll"	10/17/2000 9:24:18 AM
Re: Date Picker in Filter Dialog	"Paul Woll"	10/17/2000 9:20:30 AM
Using link fields in IP 2000 with SQL Direct	"Stein Olav Øiestad"	10/17/2000 3:41:28 AM
Master/Detail and Grids	"Phil Oneacre"	10/17/2000 4:32:00 AM
ComboBox.Style problem	"Luis Alberto"	10/17/2000 5:29:04 AM
TwwDBRichEdit popupmenu has enabled i...	"Remco Joosse"	10/17/2000 6:50:40 AM

Screenshot of a self-referencing Database treeview using a TfcTreeHeader control.

Ancestor

TCustomControl
 TCustomPanel
 TfcTreeHeader

Required supporting components

TfcDBTreeView

Added Properties

Canvas

Canvas used to paint the header control. You may wish to refer to Canvas when using the OnDrawSection event to customize the header's painting.

Data Type: TCanvas

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*.

HotTrack

Highlights the header sections as the mouse passes over them. If HotTrack is set to True, the text on each header section appears highlighted when the mouse passes over it at runtime.

Data Type: boolean

Images

Lists the images that can appear next to the text in header sections.

Use Images to provide a set of graphic images that can appear in the header sections. Each header section can be associated with one of the images in this list using its ImageIndex property.

Data Type: TCustomImageList;

Options

Specifies options for the header control.

thcoAllowColumnMove When true the header control allows a column to be dragged to another position.

thcoSortTreeOnClick When true column header is clickable and fires the *OnSectionClick* event.

thcoRightBorder When true, a right border is painted at the right of the header control.

Sections

Lists the header sections (column headings).

The Sections property holds a TfcTreeHeaderSections—that is, a collection of TfcTreeHeaderSection objects. At design time, header sections can be added, removed, or modified with the Sections editor. To open the Sections editor, select the Sections property in the Object Inspector, then double-click in the Value column to the right or click the ellipsis (...) button.

Data Type: TfcTreeHeaderSections

Tree

Tree associated with the header control

Data Type: TWinControl

Added Events

OnDrawSection

Use the *OnDrawSection* event to change the default painting of the header. The parameters for this event are as follows:

HeaderControl: TfcTreeHeader Header control being painted
Section: TfcTreeHeaderSection Header section being painted
const Rect: TRect Rectangle area to paint the header section
Pressed: Boolean True if the header is currently pressed.

In the following example if the current section being drawn is not the field that the treeview is sorted on then exit and do default painting, otherwise paint its background in yellow. *OrderByFieldName* is a global variable set in the *OnSectionClick* event.

NOTE: Call *sysutils.abort* if you wish to override the drawing of the section completely with your own code. Otherwise the default drawing of the text and images will occur afterwards.

```
procedure TSelfDBForm.HeaderControl1DrawSection(  
  HeaderControl: TfcTreeHeader; Section: TfcTreeHeaderSection;  
  const Rect: TRect; Pressed: Boolean);  
begin  
  if OrderByFieldName <> Section.FieldName then exit;  
  HeaderControl.Canvas.Brush.Color := clYellow;  
  HeaderControl.Canvas.FillRect(Rect);  
end;
```

OnResize

OnResize occurs when the header control is resized.

OnSectionClick

OnSectionClick occurs after the user clicks on a header section. The parameters for this event are as follows:

HeaderControl: TfcTreeHeader Header control
Section: TfcTreeHeaderSection Header section that was clicked

OnSectionDrag

OnSectionDrag event occurs after a column has been dragged to a new position. The parameters for this event are as follows:

Sender: TObject Header associated with the event
FromSection: TfcTreeHeaderSection Header Section being dragged to a new location

ToSection TfcTreeHeaderSection Header section currently in the position where the user dragged FromSection.

OnSectionMove

OnSectionMove occurs before a header section is dragged to a new location. The parameters for this event are as follows:

HeaderControl: TfcTreeHeader Header control
Section: TfcTreeHeaderSection Header section being moved
DragFrom: integer Section index of header being dragged.
DragTo: integer Section index where header is to be moved to.
var AllowMove: Boolean Set to false to prevent the header from being moved to the new location.

OnSectionResize

The OnSectionResize event occurs when a header section is resized at runtime. This happens when the user positions the mouse pointer between two sections and drags their border to the right or left.

HeaderControl: TfcTreeHeader Header control
Section: TfcTreeHeaderSection Header section that was resized

OnSectionTrack

Occurs as one of the header control's sections is being resized. The OnSectionTrack event tracks the dragging of a header section's border as it happens. OnSectionTrack occurs when the mouse pointer is positioned between two header sections and the left mouse button is depressed.

The parameters for this event are as follows:

HeaderControl: TfcTreeHeader Header control
Section: TfcTreeHeaderSection Header section that was resized
Width: integer Indicate the size of the header section (in pixels)
State: TSectionTrackState Indicates the status of the event. It can be one of the following values:
tsTrackBegin The border has not yet been dragged.
tsTrackMove The border is being dragged.
tsTrackEnd The border is no longer being dragged.

TfcTreeHeaderSection (Class)

TfcTreeHeaderSection represents a section of a tree header control.

TfcTreeHeader uses TfcTreeHeaderSections to maintain a collection of TfcTreeHeaderSection objects. These TfcTreeHeaderSection objects represent resizable column headers. Each column header includes a text label and possibly a graphic image.

Ancestor

TCollectionItem
TfcTreeHeaderSection

Added Properties

Alignment

Specifies how text is aligned within the header section.

Data Type: TAlignment

AllowClick

Allows the section to respond to mouse clicks at runtime.

If AllowClick is set to True (the default), the header section can be clicked with the mouse at runtime. Header sections behave like buttons when clicked: They lose their raised borders and appear depressed on the form. To attach functionality to the clicking of header sections, write an event handler for the header control's OnSectionClick event.

Data Type: Boolean

FieldName

Assign FieldName to associate the header section column with a specific field in the TfcDBTreeView. The data value of the field is displayed in the tree underneath the section column.

Data Type: String

ImageAlignment

Specifies how image is aligned within the header section.

Data Type: TAlignment

ImageIndex

Use ImageIndex to specify an image that appears next to the text on the header section. ImageIndex is the (0-offset) index of the image in the TCustomImageList component that is available through the header control's Images property.

Data Type: Integer

MaxWidth

Sets the maximum width, in pixels, for the header section. If MaxWidth = MinWidth, the header section cannot be resized at runtime.

Data Type: Integer

MinWidth

Sets the minimum width, in pixels, for the header section. If MinWidth = MaxWidth, the header section cannot be resized at runtime.

Data Type: Integer

Style

Determines how the header section's text is displayed. If Style is set to hsText (the default), the string contained in the Text property is displayed in the header section, using the alignment specified by Alignment. The font is determined by the header control's Font property. A graphic image can appear beside the text if the ImageIndex property is set.

If Style is set to hsOwnerDraw, the content displayed in the header section is drawn at run time on the header control's canvas by code in an OnDrawSection event handler.

Data Type: THeaderSectionStyle

Text

The Text property contains a string that identifies the header section or the column beneath it. If Style is set to hsText, the value of Text appears in the header section.

Data Type: String

Width

The Width property determines the default width of the header section, in pixels. Header sections can be resized at runtime by dragging their borders.

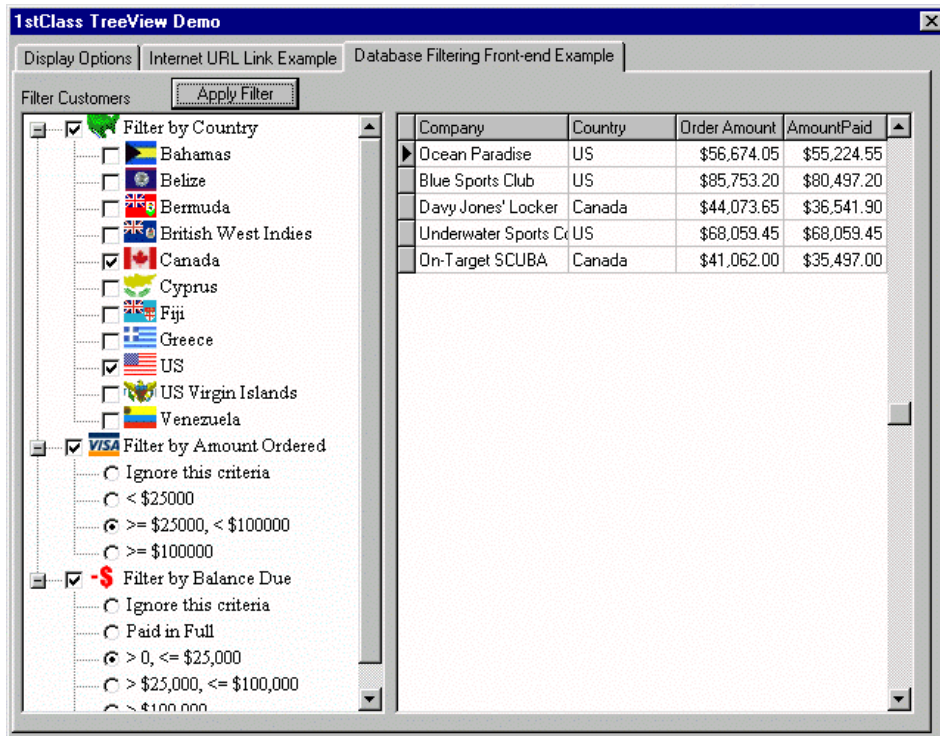
Data Type: Integer

TfcTreeView



A tree view control is a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk.

Each node in a tree view control consists of a label, a number of optional bitmapped images, and optional checkboxes and radio buttons. Each node can have a list of subnodes associated with it. By clicking on a node, the user can expand and collapse the associated list of subnodes



Using a TreeView as a Database Filtering Front-end

Ancestor

TWinControl

TfcCustomTreeView

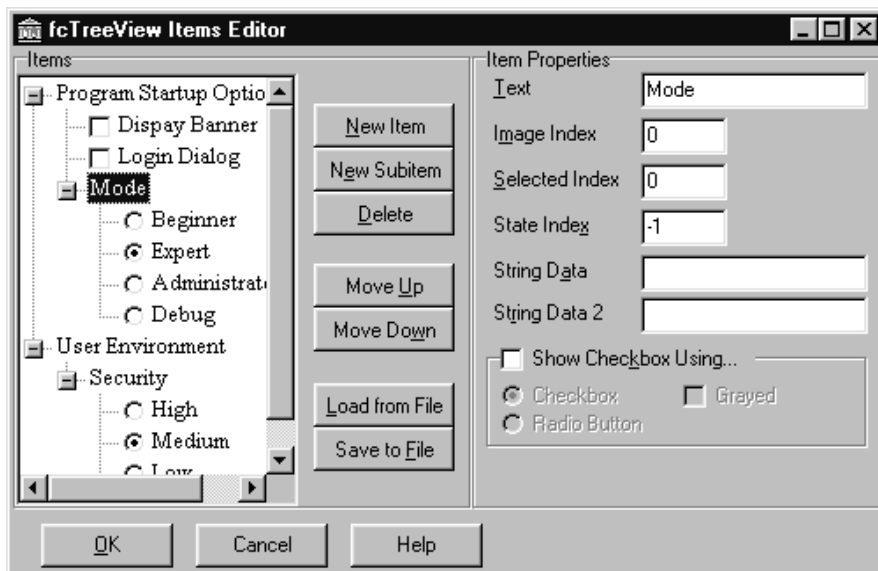
TfcTreeView

Required supporting components

None

TfcTreeView Nodes Editor

Use the TfcTreeView Items editor at design time to add or delete nodes in a treeview component, define node properties such as the text, checkbox attributes, and image attributes. To invoke the TfcTreeView Items editor, you can either double-click the TfcTreeView component or double-click the *Items* property in the Object Inspector. This property editor is also used to define the drop-down list for a TfcTreeCombo component.



Items Group Box

The TfcTreeView Items editor contains an Items group box with an Items list box, a New Item button, a New Sub item button, a Delete button, a Move Up button, a Move Down button, a Load From File button, and a Save to File button. When you first add a treeview control to a form, the Items list box is empty, and the New Sub Item and Delete buttons are disabled.

- **New Item**
Click this button to add a new node to the tree. The new node is inserted as the last child of the currently selected node's parent. If the selected node is a root node, then a node is added to the end of the list.
- **New Subitem**

Click this button to add a new child node to the currently selected node in the tree. The new node is inserted as the last child of the currently selected node.

- **Delete**

Click this button to remove the currently selected node from the tree.

- **Move Up**

Click this button to move the currently selected node so that it is before the prior sibling node. If there is no prior sibling node, then this button does nothing. You can also drag and drop nodes using the mouse to move a node to a new location. Drag and drop will also allow you to move a node up and down the tree hierarchy.

- **Move Down**

Click this button to move the currently selected node so that it is after the next sibling node. If there is no next sibling node, then this button does nothing. You can also drag and drop nodes using the mouse to move a node to a new location. Drag and drop will also allow you to move a node up and down the tree hierarchy.

- **Load From File**

Click this to load the node information from a file. The file must have been generated by calling the *SaveToFile* method of the *TreeView* or by pressing the *Save To File* button of this dialog.

- **Save To File**

Click this to save the node information to a file. You can click on the *Load From File* button reload the node information from a file.

Item Properties

The *TreeView* Items editor also contains an Item Properties group box for setting the properties of the treeview item currently selected in the Items list box. The Item Properties group box contains the following:

- **Text**

Text specifies the text displayed for the node in the tree

- **Image Index**

Image Index specifies which image is displayed when a node is in its normal state and is not currently selected.

Note: If the *Image Index* is set to -1 , then no image is displayed but the text is left-aligned as if there were an image present. If the *Image Index* is set to -2 then the text is shifted to the left to take into account the image not being painted. Setting *Image Index* to -2 is useful to remove the whitespace preceding the node's text.

- **Selected Index**

Selected Index can be set to specify an image to display when the tree node is selected.

- **State Index**

State Index indicates which image from the *StateImages* list to display for the node. Use *State Index* to display an additional image for the node that reflects state information. If *State Index* is -1, or a multiple of 16, then no state image is drawn. The reason that a multiple of 16 is not supported is due to Microsoft not supporting this in their TreeView common control.

Note: If a checkbox is displayed for the node, then *StateImages* are disabled for the node.

- **String Data**

String Data can be used to store your own information into a string. Often you may want to associate other information for a node besides just its text property. This property can be used for your own needs.

Note: If the TreeView's *Options | tvoAutoURL* is set to *True*, the TreeView uses *String Data* as a URL link address. In this mode, when *String Data* is not blank, the TreeView automatically displays the HandPoint mouse cursor when the mouse moves over the URL link, and automatically opens the address when the user clicks on the link.

- **String Data 2**

String Data 2 can be used to store your own information into a string. Often you may want to associate other information for a node besides just its *Text* property. This property can be used for your own needs.

- **Show Checkbox Using**

Set *Show Checkbox Using* to *True* to display a checkbox or a radio-button to the left of the node. The 'checkbox' can be displayed either as a Windows checkbox or a radio-button. The end-user can toggle the checkbox using the mouse or the space key.

Checkbox This mode allows each node to be *checked* or *unchecked* by the end-user. You can later inspect each node's *Checked* property to determine if the node is currently checked.

Radio Button This mode allows at most one node of a current level to have its radio-button checked. When you enable a node as a radio-button all sibling nodes become radio buttons as well. When the end-user selects the radiobutton, all sibling nodes of the current level automatically become unchecked.

Note: When *Show Checkbox Using* is enabled, state images are disabled for the node.

Note: If you wish to enable a checkbox for all nodes, you may instead want to set the *MultiSelectAttributes* | *MultiSelectCheckbox* and *MultiSelectAttributes* | *Enabled* properties to *True*.

- **Selectable**

When the property editor is invoked for a *TfcTreeCombo* object, an additional checkbox is displayed, *Selectable*. Set *Selectable* to *False* to prevent the node from being selected in the *TfcTreeCombo*'s drop-down list.

Required property assignments

None

Added Properties

AutoExpand

Specifies whether the nodes in the tree view automatically expand and collapse depending on the selection. Set *AutoExpand* to *True* to cause the selected item to expand and the unselected item to collapse.

Data Type: boolean

Canvas (Runtime only)

Provides access to the canvas. Use the *Canvas* property to paint to the canvas from the *OnCalcNodeAttributes* and *OnDrawText* event handlers.

Data Type: TCanvas

ChangeDelay

Specifies the delay between when a node is selected and when the *OnChange* event occurs. Set the *ChangeDelay* to 50 milliseconds to emulate the behavior of the tree-view control used in Windows Explorer.

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*.

DropTarget (Runtime only)

DropTarget specifies which item in the tree view appears as the target of a drag and drop operation.

Read *DropTarget* to determine whether a node in the tree view is drawn as the target of a drag and drop operation. Set *DropTarget* when specifying a particular node in the tree view as the drop target of a dragged item.

Note: When *DropTarget* is set, the application must still handle the actual logic of accepting the dragged object by the indicated node.

Data Type: TfcTreeNode

Images

Determines which image list is associated with the tree view. Use *Images* to provide a customized list of bitmaps that can be displayed to the left of a node's label.

Individual nodes specify the image from this list that should appear by setting their *ImageIndex* property. See also the *OnCalcNodeAttributes* event to customize the *ImageIndex* dynamically.

Data Type: TCustomImageList

InactiveFocusColor

Specifies the background color of the node that is selected when the treeview does *not* have focus. This property is not applicable when *Options | tvoHideSelection* is set to *True*.

Data Type: boolean

Indent

Specifies the amount of indentation in pixels when a list of child nodes is expanded. Use *Indent* to determine how far child nodes are indented from their parent nodes when the parent is expanded.

Data Type: integer

Items

Contains the individual nodes that appear in the tree view control. Individual nodes in a tree view are TfcTreeNode objects. Using the *Items* property along with the item's index into the tree view allows you to access these individual nodes. For example, to access the second item in the tree view, you could use the following code.

```
MyTreeNode := TreeView1.Items[1]
```

Note: Accessing tree view items by index can be time-intensive, particularly when the treeview contains many items. For optimal performance, try to design your application so that it has as few dependencies on the tree view's item index as possible.

You should rarely need to use the array syntax, `Items[index]`, as the `TreeView` has more efficient supporting methods for iterating through its internal list.

To efficiently iterate through a `TreeView` use the `GetFirstNode` method to get the first node in the tree, in conjunction with iterative use of the `GetNext` method of the `TfcTreeNode`. For example...

```
var Node: TfcTreeNode;  
begin  
    Node := fcTreeView1.GetFirstNode;  
    while Node<>Nil do  
        begin  
            { Perform your action here }  
            Node := Node.GetNext;  
        end;  
end;
```

To set this property at design time in the Object Inspector, see the how-to topic on using the `TreeView` Items editor

At run-time nodes can be added and inserted by using the `IfcTreeNodes` methods `AddChildFirst`, `AddChild`, `AddFirst`, `Add`, and `Insert`.

Data Type: `TfcTreeNodes`

See the section on `TfcTreeNodes` for a detailed reference to this type.

LineColor

Specifies the color of the lines drawn in the `TreeView`

Data Type: `TColor`

MultiSelectAttributes

Specifies the attributes for enabling and controlling multi-selection in the `TreeView`. This property contains the following sub-properties.

Data Type: `TfcTVMultiSelectAttributes`

AutoUnselect

When *True*, the `TreeView` will automatically unselect all previously selected nodes when the user clicks on a node without using the `Ctrl` or `Shift` keys. In addition the clicked node is selected.

Data Type: `boolean`

Enabled

When *True*, the `TreeView` will automatically use `Ctrl-Click` to select/deselect a node, or `Shift-Click` to select a range of nodes. This provides a convenient way to perform multi-selection. To allow the user to multi-select with a checkbox set both the *Enabled* and *MultiSelectCheckbox* properties to *True*.

Data Type: boolean

MultiSelectLevel

Set this to the level you wish to enable multi-selection for. Defaults to 0, which indicates only the root nodes can be selected. If set to -1, then any node in any level can be selected.

Data Type: integer

MultiSelectCheckbox

When *True*, a checkbox is displayed in each node to allow the end-user a convenient way of selecting nodes. The space key will also select the node when this property is *True*.

Data Type: boolean

MultiSelectList (Runtime only)

When using multi-select in the treeview (See *MultiSelectAttributes*), the user's selected nodes are saved to this array. This allows you to later iterate through the list of selected nodes.

Example: The following example displays the text of the selected records

```
var i: integer;
begin
  with fcTreeView1 do begin
    for i:= 0 to MultiSelectCount-1 do
      ShowMessage (MultiSelectList[i].Text);
    end;
  end;
end;
```

Data Type Array of TfcTreeNode

MultiSelectListCount (Runtime only)

MultiSelectListCount returns the number of multi-selected nodes in the treeview.

Data Type: integer

Options

This property contains a set of boolean values that control the appearance and behavior of the TreeView.

Data Type: Set of TfcTreeViewOption

Valid Values: tvoExpandOnDbIClk, tvoExpandButtons3D, tvoFlatCheckBoxes, tvoHideSelection, tvoRowSelect, tvoShowButtons, tvoShowLines, tvoShowRoot, tvoHotTrack, tvoAutoURL, tvoToolTips, tvoEditText, tvo3StateCheckbox (described below).

tvoExpandOnDbClick

Set to *True* to automatically expand the dbl-clicked node. Defaults to *True*.

tvoExpandButtons3D

Set to *True* to display the expand and collapse buttons as three-dimensional buttons. Defaults to *False*.

tvoFlatCheckBoxes

Set to *True* to display checkboxes. Defaults to *False*, which displays checkboxes in the tree-view as three-dimensional buttons.

tvoHideSelection

This property controls how a treeview displays the selected node when it does not have the focus. Set to *True* to hide the selection when the *TreeView* does not have focus. If set to *False*, the treeview displays the selected node in the color as defined by the *InactiveFocusColor* property. Defaults to *True*.

tvoRowSelect

Set to *True* to highlight the entire row to the left edge of the screen. If set to *False*, only the text is highlighted when a node is selected. Defaults to *False*.

tvoShowButtons

Set to *True* to display the expand and collapse buttons in the treeview. Defaults to *True*.

tvoShowLines

Set to *True* to display the connecting lines in the treeview. Defaults to *True*.

tvoShowRoot

To show lines connecting top-level nodes to a single root, set the tree view's *tvoShowRoot* and *tvoShowLines* properties to *True*.

tvoHotTrack

Specifies whether list items are highlighted when the mouse passes over them. Set *HotTrack* to *True* to provide visual feedback about which item is under the mouse. Set *HotTrack* to *False*, to provide no visual feedback about which item is under the mouse.

See the *tvoAutoURL* property if you wish to define and open URL links from the Treeview.

tvoAutoURL

When this property is set to *True*, the *TreeView* automatically does the following for each node that has a non-blank *StringData* value. See also the *TfcTreeNode.StringData* property.

1. Displays the text of the node in blue and underlined.
2. Displays the HandPoint mouse cursor when the user moves the mouse over the node.
3. Request Windows to open the specified URL link. For instance if a node's *StringData* property was assigned the <http://www.woll2woll.com>, then the *TreeView* would automatically open the registered Internet browser and open the Woll2Woll home page.

tvoToolTips

Set *tvoToolTips* to *True* to automatically display a hint box containing the text of the node, when the entire text of the node cannot otherwise be displayed.

tvoEditText

Set *tvoEditText* to *True* to allow the end-user to edit the text of the node.

tvo3StateCheckbox

Set to *True* to allow checkboxes to cycle between the following 3 states (Unchecked, Checked, Checked | Grayed). If *tvo3StateCheckbox* is *False*, then the checkbox toggles between the 2 states (Unchecked, Checked).

RightClickNode (Runtime only)

This returns the node that was most recently right-clicked. If the right-clicked the tree and the mouse cursor was not over a node, then this property returns nil. For instance this could happen if the tree only had a few nodes, and the user clicked on the whitespace beneath the last tree node.

This property is particularly useful if you are using a *PopupMenu* and wish to determine which node was right-clicked. When the property *RightClickSelects* is *True*, you can directly refer to the *Selected* property to determine the right-clicked node. However when *RightClickSelects* is *False*, you must refer to the *RightClickNode* property.

RightClickSelects

Setting this property to *True* will cause the tree to make the right-clicked node the selected node. If this property is *False*, you can still gain access to the right-clicked node by referring to the *RightClickNode* runtime property.

Note: Do not confuse this property with the native Delphi/Builder *TTreeView* *RightClickSelect* property. That property has no relationship to this property.

Data Type: boolean

Selected (Runtime only)

Selected returns the selected node in the tree view.

Read *Selected* to access the selected node of the tree view. If there is no selected node, the value of *Selected* is nil.

Set *Selected* to set a node in the tree view. When a node becomes selected, the tree view's *OnChanging* and *OnChanged* events occur. Also, if the specified node is the child of a collapsed parent item, the parent's list of child items is expanded to reveal the specified node. In this case, the tree views *OnExpanded* and *OnExpanding* events occur as well.

Data Type: TfcTreeNode

SortType

Determines if and how the nodes in a tree view are automatically sorted.

Once a tree view is sorted, the original hierarchy is lost. That is, setting the *SortType* back to *fcstNone* will not restore the original order of items. These are the possible values:

<u>Value</u>	<u>Meaning</u>
fcstNone	No sorting is done.
fcstData	The items are sorted when the <i>Data</i> object or <i>SortType</i> is changed.
fcstText	The items are sorted when the <i>Caption</i> or <i>SortType</i> is changed.
fcstBoth	The items are sorted when either the <i>Data</i> object, the <i>Caption</i> or <i>SortType</i> is changed.

Optionally, the *OnCompare* event can be hooked to handle comparisons. The *OnCompare* event will be called to compare two nodes for sorting.

Data Type: TfcSortType

Valid Values: fcstNone, fcstData, fcstText, fcstBoth

StateImages

Determines which image list to use for state images. Use *StateImages* to provide a set of bitmaps that reflect the state of tree view nodes. The state image appears as an additional image to the left of the item's icon.

Data Type: TCustomImageList

TopItem (Runtime only)

TopItem is the topmost node that appears in the tree view. When *TopItem* is changed, the tree view scrolls vertically so that the specified node is topmost in the list view.

Data Type: TfcTreeNode

Added Events

OnCalcNodeAttributes

This event allows you to change the node and painting canvas attributes before the node is painted by the *TreeView*. Use this event to change the font, background color, the node's text, and other node attributes.

TreeView: *TfcCustomTreeView* *TreeView* associated with the node to be painted.
If you wish to access the painting canvas to change the painting attributes of the node, refer to the *TreeView's Canvas* property.

Node: *TfcTreeNode* *Node* that is about to be painted

ItemState: *TfcItemStates* State is the item's current state: one or more of *fcisSelected*, *fcisGrayed*, *fcisDisabled*, *fcisChecked*, *fcisFocused*, *fcisDefault*, *fcisHot*, *fcisMarked*, *fcisIndeterminate*.

Example: The following code causes the root nodes to paint with boldfaced text.

```
Procedure TreeViewDemoForm.fcTreeView1CalcNodeAttributes (  
  TreeView: TfcCustomTreeView;  
  Node: TfcTreeNode; State: TfcItemStates);  
begin  
  if Node.Level=0 then TreeView.Canvas.Font.Style:= [fsBold];  
end;
```

OnChange

OnChange occurs whenever the selection has changed from one node to another.

The parameters for this event are as follows:

TreeView: *TfcCustomTreeView* *TreeView* associated with the event

Node: *TfcTreeNode* The *Node* parameter is the node whose selection state has changed.

OnChanging

OnChanging occurs whenever the selection is about to be changed from one node to another.

The parameters for this event are as follows:

TreeView: *TfcCustomTreeView* *TreeView* associated with the event

Node: *TfcTreeNode* The *Node* parameter specifies the currently selected node.

AllowChange: boolean Set *AllowChange* to *False*, to prevent selection from moving to a new node.

OnCollapsed

OnCollapsed occurs after a node has been collapsed. Write an *OnCollapsed* event handler to respond after a node in the tree view collapses.

The parameters for this event are as follows:

TreeView: TfcCustomTreeView *TreeView* associated with the event

Node: TfcTreeNode The *Node* parameter is the node whose children are no longer visible.

OnCollapsing

OnCollapsing occurs when a node is about to be collapsed.

The parameters for this event are as follows:

TreeView: TfcCustomTreeView *TreeView* associated with the event

Node: TfcTreeNode The *Node* parameter specifies the node that is about to be collapsed.

AllowCollapse: boolean Set the *AllowCollapse* parameter to *False* to prevent the node specified by the *Node* parameter from being collapsed.

OnCompare

OnCompare occurs when two nodes must be compared during a sort of the nodes in the tree view. Write an *OnCompare* event handler to customize the sort order of the nodes in the tree view. If an *OnCompare* event handler is not provided, tree view nodes are sorted alphabetically, based on their labels.

The parameters for this event are as follows:

TreeView: TfcCustomTreeView *TreeView* associated with the event

Node1, Node2: TfcTreeNode The Nodes being compared

Data: Integer *Not currently used*

Compare: Integer Set the *Compare* parameter to a value less than 0 if *Node1* is less than *Node2*. Set *Compare* to 0 if *Node1* is equivalent to *Node2*, and set *Compare* to a value greater than 0 if *Node1* is greater than *Node2*.

OnDbClick

See *OnMouseDown* event.

OnDeletion

OnDeletion occurs when a node in the tree view is deleted. Write an *OnDeletion* event handler to respond when a node is deleted from the tree view control.

The parameters for this event are as follows:

TreeView: TfcCustomTreeView *TreeView* associated with the event

Node: TfcTreeNode The *Node* parameter is the node to be deleted

OnDrawText

This event allows you to change the default text painting of the node. You will rarely need to use this event, as the *OnCalcNodeAttributes* is the preferred event to use to change a node's font, text, background, and other attributes. You will only need this event if you wish to override the actual painting of the node's text.

The parameters for this event are as follows:

TreeView: TfcCustomTreeView *TreeView* associated with the node. If you wish to access the painting canvas, refer to the *TreeView's Canvas* property.

Node: TfcTreeNode *Node* that is about to be painted

ARect: TRect Default rectangle where the text is to be painted.

AItemState: TfcItemStates State is the item's current state: one or more of *fcisSelected*, *fcisGrayed*, *fcisDisabled*, *fcisChecked*, *fcisFocused*, *fcisDefault*, *fcisHot*, *fcisMarked*, *fcisIndeterminate*.

DefaultDrawing: boolean Specifies whether the control should paint the item.

Example: The following example underlines the character following ampersands in the node's text by using the TCanvas *DrawText* method. Note that the code below does NOT make the node accessible through an accelerator key. If you desire this behavior you would need to write the code to trap the keys on your own using events such as the *OnKeyDown* event.

```
Procedure TForm1.fcTreeView1DrawText(  
  TreeView: TfcCustomTreeView;  
  Node: TfcTreeNode; ARect: TRect; AItemState: TfcItemStates;  
  var DefaultDrawing: boolean);  
begin  
  { Underlines characters following ampersand }  
  TreeView.Canvas.DrawText(Node.Text, ARect, 0);  
  if fcisFocused in AItemState then begin { Draw focus rect }  
    InflateRect(ARect, 1, 1);  
    TreeView.Canvas.DrawFocusRect(ARect);  
end;
```

```
    DefaultDrawing := False;  
end;
```

OnEdited

Occurs after the user edits the *Text* property of a node. This event can occur only if *Options | tvcEditText* is set to True.

The parameters for this event are as follows:

<i>TreeView</i> : TfcCustomTreeView	<i>TreeView</i> associated with the event
<i>Node</i> : TfcTreeNode	The <i>Node</i> parameter is the node whose label was edited
<i>S</i> : string	The <i>S</i> parameter is the new value of the node's <i>i</i> property. The node's label can be changed in an <i>OnEdited</i> event handler before the user's edits are committed

OnEditing

Occurs when the user starts to edit the *Text* property of a node. Write an *OnEditing* event handler to determine whether the user is allowed to edit the label of a specific node in the tree view.

The parameters for this event are as follows:

<i>TreeView</i> : TfcCustomTreeView	<i>TreeView</i> associated with the event
<i>Node</i> : TfcTreeNode	The <i>Node</i> parameter is the node whose label is about to be edited
<i>AllowEdit</i> : boolean	Set the <i>AllowEdit</i> parameter to <i>False</i> to prevent the user from editing the node specified by the <i>Node</i> parameter. To disallow editing of all nodes in the tree view, set the <i>Options tvcEditText</i> property to <i>False</i> instead.

OnExpanding

Occurs when a node is about to be expanded. Write an *OnExpanding* event handler to determine whether a node can be expanded.

The parameters for this event are as follows:

<i>TreeView</i> : TfcCustomTreeView	<i>TreeView</i> associated with the event
<i>Node</i> : TfcTreeNode	The <i>Node</i> parameter is the node that is about to be expanded
<i>AllowExpansion</i>	Set the <i>AllowExpansion</i> parameter to <i>False</i> to prevent the node from expanding.

OnExpanded

Occurs after a node is expanded. Write an *OnExpanded* event handler to respond when a node in the tree view is expanded.

The parameters for this event are as follows:

TreeView: TfcCustomTreeView *TreeView* associated with the event

Node: TfcTreeNode The *Node* parameter specifies the node whose children are now displayed to the user.

OnGetImageIndex

Occurs when the tree view looks up the *ImageIndex* of a node. Write an *OnGetImageIndex* event handler to change the image index for the particular node before it is drawn. For example, the bitmap of a node can be changed to indicate a different state for the node.

Note: If the Node's *ImageIndex* property is set to -1, then no image is displayed but the text is left-aligned as if there were an image present. If the Node's *ImageIndex* is set to -2 then the text is shifted to the left to take into account the image not being painted. Setting *ImageIndex* to -2 is useful to remove the whitespace preceding the node's text.

The parameters for this event are as follows:

TreeView: TfcCustomTreeView *TreeView* associated with the event

Node: TfcTreeNode The node whose *ImageIndex* you wish to change.

OnGetSelectedIndex

Occurs when the tree view looks up the *SelectedIndex* of a node. Write an *OnGetSelectedIndex* event handler to change the selected image index of a node before it is drawn.

The parameters for this event are as follows:

TreeView: TfcCustomTreeView *TreeView* associated with the event

Node: TfcTreeNode The *Node* whose *Selected* image index you wish to change.

OnItemChange

This event allows you to take some custom action when a node's text or image index is changed, or if a node is added or deleted from the *TreeView*. Do not confuse this event with the *OnChange* event, which is fired when you change the active node.

The parameters for this event are as follows:

TreeView: TfcCustomTreeView *TreeView* associated with the node that is changed.

<i>Node</i> : TfcTreeNode	<i>Node</i> that is changed
<i>Action</i> : TfcItemChangeAction	<i>Action</i> associated with the <i>OnItemChange</i> event. It can be one of the following values. <i>icaAdd</i> : New node added to the TreeView <i>icaDelete</i> : Node is removed from the TreeView <i>icaText</i> : Text is modified for node <i>icaImageIndex</i> : <i>ImageIndex</i> property for node is modified.
<i>NewValue</i> : Variant	Data associated with the action. If <i>Action=icaText</i> , then <i>NewValue</i> is the new text. If <i>Action=icaImageIndex</i> , then <i>NewValue</i> is the new <i>ImageIndex</i> value. Otherwise this value is Null.

OnMouseDown, OnMouseUp, OnDbClick

Use this event to perform some custom action when the mouse is pressed, released, or Double-clicked over the TreeView. The parameters for this event are as follows:

<i>TreeView</i> : TfcCustomTreeView	<i>TreeView</i> associated with the event
<i>Node</i> : TfcTreeNode	<i>Node</i> that the mouse is over at the time of the event.
<i>Button</i> : TMouseButton	Distinguishes which mouse button generated the mouse event. Can be <i>mbLeft</i> , <i>mbRight</i> , or <i>mbMiddle</i> .
<i>Shift</i> : TShiftState	Use the <i>Shift</i> parameter to respond to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations.
<i>X, Y</i> : Integer	X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the TreeView.

OnMouseMove

Use this event to perform some custom action when the mouse moves over a node.

The parameters for this event are as follows:

<i>TreeView</i> : TfcCustomTreeView	<i>TreeView</i> associated with the event
<i>Node</i> : TfcTreeNode	<i>Node</i> that the mouse is over
<i>Shift</i> : TShiftState	Use the <i>Shift</i> parameter to respond to the state of the shift keys and mouse buttons. Shift keys are

the Shift, Ctrl, and Alt keys or shift key-mouse button combinations.

X, Y: Integer

X and *Y* are pixel coordinates of the new location of the mouse pointer in the client area of the `TreeView`.

OnToggleCheckbox

Use this event to perform your own custom action after a node's checkbox or radiobutton has been toggled.

The parameters for this event are as follows:

TreeView: `TfcCustomTreeView` *TreeView* associated with the event

Node: `TfcTreeNode` *Node* associated with the checkbox being toggled.

Added Methods

AlphaSort

`AlphaSort` sorts all nodes alphabetically by label in the tree view control.

```
function AlphaSort: boolean;
```

Call `AlphaSort` to sort the nodes of the tree view. If successful, `AlphaSort` returns `True`. To have the tree view maintain all nodes in a sorted order (for example, when the user edits the labels), use the `SortType` property.

CustomSort

`CustomSort` sorts the nodes in the tree view into a customized sort order.

```
function CustomSort (SortProc: TTVCompare;  
    Data: Longint): boolean;
```

Use `CustomSort` to sort the nodes of a tree view, where the sort order is defined by the `SortProc` parameter. The `IParam1` and `IParam2` parameters of the sort procedure can be cast to `TfcTreeNode` objects are compared. The `IParamSort` parameter of the sort procedure is the value of `Data` parameter of `CustomSort`. The sort procedure should return a value less than 0 if `IParam1` should come before `IParam2`, should return 0 if the two values are equivalent, and should return a value greater than 0 if `IParam1` should follow `IParam2`.

If the `SortProc` parameter is nil, the `AlphaSort` method is called.

Note: To have the tree view sort all nodes automatically (for example, when the user edits the labels), use the `SortType` property instead, and provide an `OnCompare` event handler.

Example: This example shows how to use the `CustomSort` method to order a tree view in reverse alphabetical order. The application must provide a callback function such

as *CustomSortProc* below, which calls the global *AnsiStrIComp* function and negates its return value.

```
function CustomSortProc(Node1, Node2: TfcTreeNode;  
    Data: integer): integer; stdcall;  
begin  
    Result := -AnsiStrIComp(PChar(Node1.Text),  
        PChar(Node2.Text));  
end;
```

This procedure can then be used as a parameter to *CustomSort* to sort the nodes of the tree view:

```
TreeView1.CustomSort(@CustomSortProc, 0);
```

FullCollapse

Call *FullCollapse* to hide all the nodes in the tree view except those at the first level.

```
procedure FullCollapse;
```

FullExpand

FullExpand expands all nodes within the tree view control.

```
procedure FullExpand;
```

GetFirstSibling

Returns the first sibling of a node. If the parameter *Node* is set to Nil, then *GetFirstSibling* returns the first node in the tree.

```
Function GetFirstSibling(Node: TfcTreeNode): TfcTreeNode;
```

GetHitTestInfoAt

GetHitTestInfoAt returns information about the location of a point relative to the client area of the tree view control.

```
function GetHitTestInfoAt(X, Y: Integer): TfcHitTests;
```

Call *GetHitTestInfoAt* to determine what portion of the tree view, if any, sits under the point specified by the *X* and *Y* parameters. For example, use *GetHitTestInfoAt* to provide feedback about how to expand or collapse nodes when the mouse is over the relevant portions of the tree view.

GetHitTestInfoAt returns a TfcHitTests type. The possible return values are:

Value	Location of (X,Y)
fchtAbove	Above the client area of the tree view control.
fchtBelow	Below the client area of the tree view control.
fchtNowhere	In the client area of the tree view control but below the last item.
fchtOnItem	On the bitmap or label associated with an item.

<code>fchtOnButton</code>	On the button associated with an item.
<code>fchtOnIcon</code>	On the bitmap associated with an item.
<code>fchtOnIndent</code>	On the indentation associated with an item.
<code>fchtOnLabel</code>	On the label (text) associated with an item.
<code>fchtOnRight</code>	In the area to the right of an item.
<code>fchtOnStateIcon</code>	On the state icon for a tree view item that is in a user-defined state.
<code>fchtToRight</code>	To the right of the client area of the tree view control.
<code>fchtToLeft</code>	To the left of the client area of the tree view control.

GetNodeAt

GetNodeAt returns the node that is found at the specified position.

```
function GetNodeAt (X, Y: Integer): TfcTreeNode;
```

Call *GetNodeAt* to access the node at the position specified by the X and Y parameters. X and Y specify the position in pixels relative to the top left corner of the tree view. If there is no node at the location, *GetNodeAt* returns nil.

InvalidateNode

Call *InvalidateNode* to force the TreeView to repaint a particular node.

```
Procedure InvalidateNode (Node: TfcTreeNode);
```

IsEditing

IsEditing indicates whether a node is currently being edited by the user. *IsEditing* returns *True* if any node label in the tree view is being edited.

```
function IsEditing: boolean;
```

LoadFromFile

LoadFromFile reads the file specified in FileName and loads the data into the tree view.

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

Use the *LoadFromFile* method to retrieve tree view data from a file and load it into a tree view.

LoadFromStream

LoadFromStream reads tree view data from a stream and stores the contents in the tree view.

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

Use *LoadFromStream* to read the nodes of the tree view from the specified stream. For example, an application can save the information displayed in a tree view as the

data in a Binary Large Object (BLOB) field. *LoadFromStream* can retrieve the data using a *TBlobStream* object.

SaveToFile

SaveToFile saves the tree view to the file specified in *FileName*. Use the *SaveToFile* method to store tree view data to a text file. The nodes can later be reloaded from the file into a new tree view object using the *LoadFromFile* method.

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

SaveToStream

SaveToStream writes the data in the tree view to the stream passed as the *Stream* parameter. Use the *SaveToStream* method to stream out tree view data. It can be streamed back in to another tree view object using the *LoadFromStream* method.

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

UnselectAll

When *MultiSelectAttributes | Enabled* is *True*, then this method unselects all previously selected nodes.

```
Procedure UnselectAll;
```

How To

How to use multi-selection in the TreeView

To allow the user to multi-select in the Tree, set the *MultiSelectAttributes | Enabled* to *True*. Then the *TreeView* will automatically use *Ctrl-MouseClick* to select/deselect a node, or *Shift-MouseClick* to select a range of nodes. This provides a convenient way to perform multi-selection. If you wish for any node to be selected, set the *MultiSelectAttributes | MultiSelectLevel* to -1 . If you wish for only the root nodes to be selectable, then set this property to 0 . If you wish to display a checkbox next to each node that can be multi-selected then set the *MultiSelectAttributes | MultiSelectCheckbox* property to *True*.

To iterate through the list of selected nodes, see the example documented under the *TfcTreeView MultiSelectList* property.

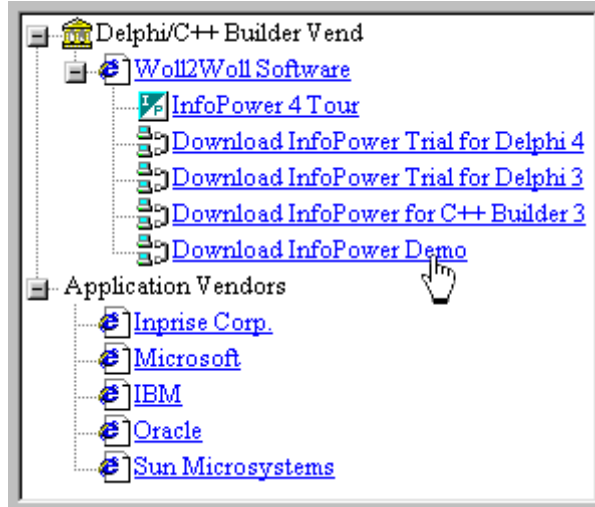
To select/unselect all nodes with code, you can call the *SelectAll* or *UnselectAll* methods.

To automatically unselect all the previously selected nodes when the user clicks on the text of a node (with the mouse not in its up state), set the *MultiSelectAttributes | AutoUnselect* property to *True*. You may wish to set this property to *False* so that the end-user does not inadvertently unselect all previously selected nodes.

How to define URL Links in the TreeView

The *1stClass TreeView* provides a convenient way to display the node's text as URL links, and open the respective URL link when the user clicks on the node. In addition the mouse

cursor will automatically change to a HandPoint when the mouse is moved over a node containing a URL link.



URL Links with hand point

The following are the steps required to enable this functionality.

1. Set the TreeView's *Options* | *twoAutoURL* property to *True*.
2. Dbl-click the TreeView at design time to bring up the Node's editor. Select the node you wish to relate a URL link address to. Enter the URL link address using the *StringData* property. If *StringData* is left unassigned, then the node is not considered to contain a URL link and is displayed normally.

How to iterate through all the nodes.

To efficiently iterate through a TreeView use the *GetFirstNode* method to get the first node in the tree, in conjunction with iterative use of the *GetNext* method of the *TfcTreeNode*. For example...

```
var Node: TfcTreeNode;  
begin  
  Node := fcTreeView1.GetFirstNode;  
  while Node<>Nil do  
  begin  
    { Perform your action here }  
    Node := Node.GetNext;  
  end;  
end;
```

How to iterate through all the immediate children of a node.

To efficiently iterate through all the child nodes of a given node, use the *GetFirstChild* method to get the first node in the tree, in conjunction with iterative use of the *GetNextSibling* method of the *TfcTreeNode*. For example the following code iterates through all the child nodes of *StartingNode* (*TfcTreeNode*).

```
Node := StartingNode.GetFirstChild;
while Node<>Nil do
begin
  { Perform your action here }
  Node := Node.GetNextSibling;
end;
```


Index

I

IstClass	
benefits	4
component overview	11
components	<i>See</i> Component overview
demonstration/sample project	11
description	3
installing	5
introduction	1
license agreement	2
on-line help	<i>See</i> on-line help
source code	14
technical support	3
uninstalling	10
IstClass 3000	
custom framing	73
transparency	73

A

AbsoluteIndex property	
TfcTreeNode	155
Action property	
TfcOutlookListItem	112
ActiveDataSet property	
TfcDBTreeView	57
ActiveNode property	
TfcDBTreeView	57
ActivePage property	
TfcOutlookBar	107
Add method	
TfcButtonGroupItems	22
TfcOutlookPages	110
TfcStatusPanels	127
TfcTreeNodes	167
AddChild method	
TfcTreeNodes	167
AddChildFirst method	
TfcTreeNodes	167

AddChildObject method	
TfcTreeNodes	168
AddChildObjectFirst method	
TfcTreeNodes	168
AddFirst method	
TfcTreeNodes	168
Adding color dialog support	
TfcColorList	46
AddObject method	
TfcTreeNodes	168
AddObjectFirst method	
TfcTreeNodes	169
Aligning colors on the right	
TfcColorList	46
Alignment event	
TfcTreeHeaderSection	175
Alignment property	
TfcCalcEdit	26
TfcColorList	40
TfcText	134
AlignmentVertical property	
TfcColorCombo	33
TfcTreeCombo	147
AllColors property	
TfcColorList	40
AllowClearKey property	
TfcColorCombo	33
TfcFontCombo	77
TfcTreeCombo	147
AllowClick event	
TfcTreeHeaderSection	175
AllowNull property	
TfcCalcEdit	26
AlphaBlend property	
TfcImager.BitmapOptions	97
AlphaSort method	
TfcTreeNode	161
TfcTreeView	193
Animation property	
TfcOutlookBar	107
ApplyBitmapRegion method	
TfcImageForm	93
Assign method	
TfcTreeNode	161

TfcTreeNodeNodes.....	169
AutoBold property	
TfcButtonGroup	19, 23
AutoDropDown property	
TfcColorCombo	33
AutoExpand property	
TfcTreeView	181
TfcTreeView.Options	185
Automatically display all hints	
TfcStatusBar	128
TfcStatusPanel	132
AutoSize property	
TfcImageForm	92
TfcImager	96
AutoSizeHeightAdjust property	
EditFrame	74
Frame	74
AutoUnselect property	
TfcDBTreeView.MultiSelectAttributes60	
TfcTreeView.MultiSelectAttributes..	183

B

Background property	
TfcCalcEdit	27
BackgroundStyle property	
TfcCalcEdit	27
bcsCheckList property	
TfcButtonGroup.ClickStyle	20
bcsClick property	
TfcButtonGroup.ClickStyle	20
bcsRadioGroup property	
TfcButtonGroup.ClickStyle	20
BeginUpdate method	
TfcTreeNodeNodes.....	169
benefits	4
Bevel property	
TfcStatusPanel	130
BitmapOptions property	
TfcImager	49, 97
Blending bitmaps	
TfcImager	102
BlockColor property	
TfcProgressBar	119
BlockSize property	
TfcProgressBar	119
boAutoBold property	
TfcImageBtn.Options.....	87

boFocusable property	
TfcImageBtn.Options.....	86
boFocusRect property	
TfcImageBtn.Options.....	87
BorderAroundLabel property	
TfcGroupBox	81
boToggleOnUp property	
TfcImageBtn.Options.....	90
TfcImageBtn.Options.....	86
Btn3DLight property	
TfcImageBtn.ShadeColors	87
BtnBlack property	
TfcImageBtn.ShadeColors	87
BtnFocus property	
TfcImageBtn.ShadeColors	88
BtnHighlight property	
TfcImageBtn.ShadeColors	87
BtnShadow property	
TfcImageBtn.ShadeColors	87
button effects	
Flat property.....	17
supporting components	73
Transparent property	17
ButtonClassName property	
TfcButtonGroup	18, 19
TfcOutlookBar	107
ButtonEffects property	
TfcColorCombo	33
ButtonGlyph property	
TfcColorCombo	33
ButtonItem property	
TfcButtonGroup	19, 22, 23
ButtonMargin property	
TfcCalcEdit	27
Buttons property	
TfcButtonGroup	19
ButtonSize property	
TfcOutlookBar	108
ButtonStyle property	
TfcCalcEdit	26
TfcColorCombo	34
TfcTreeCombo	147
ButtonWidth property	
TfcCalcEdit	26
TfcTreeCombo	148

C

CalcOptions property	
TfcCalcEdit	27
Canvas property	
TfcDBTreeView	57
TfcTreeHeader	171
TfcTreeView	181
CaptionBarControl property	
TfcImageForm	92, 94, 95
CaptionIndent property	
TfcGroupBox	81
cboAutoCreateOutlookList property	
TfcOutlookBar.Options	108
cboCloseOnEquals property	
TfcCalcEdit	28
cboDigitGrouping property	
TfcCalcEdit	28
cboFlatButtons property	
TfcCalcEdit	27
cboFlatDrawStyle property	
TfcCalcEdit	28
cboHideBorder property	
TfcCalcEdit	27
cboHideEditor property	
TfcCalcEdit	27
cboHideMemory property	
TfcCalcEdit	28
cboHotTrackButtons property	
TfcCalcEdit	27
cboRoundedButtons property	
TfcCalcEdit	28
cboSelectOnEquals property	
TfcCalcEdit	28
cboShowDecimal property	
TfcCalcEdit	28
cboShowStatus property	
TfcCalcEdit	28
cboSimpleCalc property	
TfcCalcEdit	28
cboTransparentPanels property	
TfcOutlookBar.Options	109
ccoGroupSystemColors property	
TfcColorList.Options	43
ccoShowColorNames property	
TfcColorList.Options	43
ccoShowColorNone property	
TfcColorList.Options	42
ccoShowCustomColors property	
TfcColorList.Options	42
ccoShowGreyScale property	
TfcColorList.Options	43
ccoShowStandardColors property	
TfcColorList.Options	43
ccoShowSystemColors property	
TfcColorList.Options	42
Centering captions	
TfcLabel	104
ChangeDelay property	
TfcTreeView	181
Changing buttons ability to receive focus	
TfcShapeBtn	125
Changing node color based on field	
TfcDBTreeView	71
Changing selected button color	
TfcButtonGroup	24
CheckboxType property	
TfcTreeNode	155
Checked property	
TfcTreeNode	155
Clear method	
TfcBitmap	16
TfcButtonGroupItems	23
TfcTreeNodes	169
ClickStyle property	
TfcButtonGroup	18, 20
TfcOutlookList	111
CloseUp method	
TfcCalcEdit	30
TfcColorCombo	37
TfcFontCombo	80
TfcTreeCombo	152
Col property	
TfcStatusPanel	130
Collapse method	
TfcDBTreeView	67
TfcTreeNode	161
Color property	
TfcImageBtn	84
TfcImager.BitmapOptions	97
TfcShapeBtn	122
TfcStatusPanel	130
TfcText.Shadow	136
ColorAlignment property	
TfcColorCombo	34
TfcColorList	41, 46
ColorDialog property	
TfcColorCombo	34
ColorDialogOptions property	

TfcColorCombo	34
ColorFromIndex method	
TfcColorList	45
ColorListOptions property	
TfcColorCombo	34
ColorMargin property	
TfcColorList	41
ColorWidth property	
TfcColorList	41
Columns property	
TfcButtonGroup	20
Component Hierarchy	11, 12
Complete	12
Component overview	11
Component property	
TfcStatusPanel	130
Conserve resources with TfcImageBtn	
TfcButtonGroup	23
Constrain button width	
TfcButtonGroup	23
Contrast property	
TfcImager.BitmapOptions	97
ControlSpacing property	
TfcButtonGroup	20
Count property	
TfcTreeNode	155
TfcTreeNodes	166
CreateOutlookList method	
TfcOutlookPage	110
Creating a drag control for the form	
TfcImageForm	94
Creating a nonrectangular form	
TfcImageForm	94
csClick property	
TfcOutlookList.ClickStyle	111
csoByIntensity property	
TfcColorList.SortBy	44
csoByName property	
TfcColorList.SortBy	43
csoByRGB property	
TfcColorList.SortBy	43
csoNone property	
TfcColorList.SortBy	43
csSelect property	
TfcOutlookList.ClickStyle	111
custom framing	
key properties and events	73
supporting components	73
custom framing and transparency effects	73
CustomColors property	

TfcColorCombo	34
TfcColorList	41, 42, 45, 46
CustomSort method	
TfcTreeNode	161
TfcTreeView	194
Cut property	
TfcTreeNode	156

D

Data property	
TfcTreeNode	156
DataField property	
TfcCalcEdit	28
TfcColorCombo	34
TfcDBImager	50
TfcLabel	103
TfcProgressBar	118
TfcShapeBtn	122
TfcTrackbar	140
TfcTreeCombo	148
DataLink property	
TfcDBTreeNode	52
DataSet property	
TfcDBTreeNode	52
DataSource property	
TfcCalcEdit	29
TfcColorCombo	35
TfcDBImager	50
TfcLabel	103
TfcProgressBar	118
TfcShapeBtn	122
TfcTrackbar	140
TfcTreeCombo	148
DataSourceFirst property	
TfcDBTreeView	57
DataSourceLast property	
TfcDBTreeView	57
DataSources property	
TfcDBTreeView	57
Defining a custom shaped button	
TfcShapeBtn	124
Delete method	
TfcTreeNode	162
TfcTreeNodes	169
DeleteChildren method	
TfcTreeNode	162
deleting 1stClass ..See uninstalling 1stClass	

Deleting property		TfcColorCombo.....	37
TfcTreeNode.....	156	TfcTreeCombo.....	152
Depth property		DropDownCount property	
TfcText.ExtrudeEffects.....	134, 135	TfcColorCombo.....	35
Design-time aids		TfcFontCombo.....	77
TfcButtonGroup.....	18, 23	TfcTreeCombo.....	148
TfcDBTreeView.....	59	DropDownWidth property	
TfcImageBtn.....	83	TfcColorCombo.....	35
TfcOutlookBar.....	106	TfcFontCombo.....	77
TfcShapeBtn.....	121	TfcTreeCombo.....	148
TfcTreeView.....	178	DropTarget property	
DisabledColors property		TfcTreeNode.....	156
TfcText.....	134	TfcTreeView.....	181
DisableThemes property		dsBlendDither property	
TfcDBTreeView.....	58	TfcImageBtn.DitherStyle.....	85
TfcProgressBar.....	118	dsCenter property	
TfcShapeBtn.....	122	TfcImager.DrawStyle.....	100
TfcStatusBar.....	126	dsDither property	
TfcTrackBar.....	140	TfcImageBtn.DitherStyle.....	84
TfcTreeCombo.....	148	dsFill property	
TfcTreeHeader.....	171	TfcImageBtn.DitherStyle.....	85
TfcTreeview.....	181	dsNormal property	
DisplayFields property		TfcImager.DrawStyle.....	100
TfcDBTreeView.....	54, 58	dsProportional property	
DisplayFormat property		TfcImager.DrawStyle.....	100
TfcCalcEdit.....	29	dsProportionalCenter property	
TfcProgressBar.....	118	TfcImager.DrawStyle.....	100
Displaying in InfoPower Grid		dsStretch property	
TfcColorCombo.....	38, 153	TfcImager.DrawStyle.....	100
DisplayRect method		dsTile property	
TfcTreeNode.....	162	TfcImager.DrawStyle.....	100
DitherColor property		dtvoAutoExpandOnDSScroll property	
TfcImageBtn.....	84	TfcDBTreeView.Options.....	61
DitherStyle property		dtvoExpandButtons3D property	
TfcImageBtn.....	84	TfcDBTreeView.Options.....	62
DoubleBuffered property		dtvoFlatCheckBoxes property	
TfcText.....	134	TfcDBTreeView.Options.....	62
Drag and Drop		dtvoHideSelection property	
TfcColorList.....	47	TfcDBTreeView.Options.....	59, 62
DragTolerance property		dtvoHotTracking property	
TfcImageForm.....	92	TfcDBTreeView.Options.....	63, 71
DrawInGridCell method		dtvoKeysScrollLevelOnly property	
TfcColorCombo.....	37	TfcDBTreeView.Options.....	61
TfcTreeCombo.....	152	dtvoRowSelect property	
DrawStyle property		TfcDBTreeView.Options.....	62
TfcImager.....	100	dtvoShowButtons property	
DropDown		TfcDBTreeView.Options.....	62
TfcFontCombo.....	80	dtvoShowHorzScrollBar property	
DropDown method		TfcDBTreeView.Options.....	62
TfcCalcEdit.....	30	dtvoShowLines property	

TfcDBTreeView.Options	62
dtvoShowNodeHint property	
TfcDBTreeView.Options	62
dtvoShowRoot property	
TfcDBTreeView.Options	62
dtvoShowVertScrollBar property	
TfcDBTreeView.Options	62

E

edit controls	
custom framing	73
transparency effects	73
EditText method	
TfcTreeNode	162
Embossed property	
TfcImager.BitmapOptions	97
Emulating Outlook Express's OutlookBar	
TfcOutlookList	116
Enabled property	
EditFrame	74
Frame	74
TfcDBTreeView.MultiSelectAttributes60	
TfcOutlookBar.Animation	107
TfcOutlookListItem	112
TfcStatusPanel	130
TfcText.ExtrudeEffects	134, 135
TfcText.Shadow	136
TfcTreeView.MultiSelectAttributes..	183
EndEdit method	
TfcTreeNode	162
EndUpdate method	
TfcTreeNode	169
ExecuteColorDialog method	
TfcColorCombo	37
Expand method	
TfcDBTreeView	67
TfcTreeNode	163
Expanded property	
TfcDBTreeNode	52
TfcTreeNode	157
ExtImage property	
TfcImageBtn	23, 85, 89
ExtImageDown property	
TfcImageBtn	23, 85, 89
ExtrudeEffects property	
TfcText	134

F

FarColor property	
TfcText.ExtrudeEffects	135
fbsFlat property	
TfcImageBtn.ShadeStyle	88
fbsHighlight property	
TfcImageBtn.ShadeStyle	88
fbsNormal property	
TfcImageBtn.ShadeStyle	88
fbsRaised property	
TfcImageBtn.ShadeStyle	88
fclsDefault property	
TfcText.Style	137
fclsLowered property	
TfcText.Style	137
fclsOutline property	
TfcText.Style	137
fclsRaised property	
TfcText.Style	137
fcstBoth property	
TfcTreeView.SortType	187
fcstData property	
TfcTreeView.SortType	187
fcstText property	
TfcTreeView.SortType	187
FieldName event	
TfcTreeHeaderSection	175
FindButton method	
TfcButtonGroupItems	22
FindNode method	
TfcTreeNode	170
Flat property	17
Flat-style buttons	
TfcShapeBtn	125
Focus color outline	
TfcShapeBtn	125
Focusable property	
TfcImager	100
FocusBorders property	
EditFrame	74
Frame	74
Focused property	
TfcTreeNode	157
FocusStyle property	
EditFrame	74
Frame	74
Font property	
TfcStatusPanel	131

Formatting display of date/time	
TfcStatusBar	128
frame effects	
key properties and events.....	73
supporting components	73
Frame property	
TfcColorCombo.....	35
TfcFontCombo	77
TfcGroupBox	82
TfcPanel.....	117
TfcTreeCombo	148
Frequency property	
TfcTrackBar	140
FullBorder property	
TfcGroupBox	82
FullCollapse method	
TfcTreeView	194
FullExpand method	
TfcTreeView	194

G

GaussianBlur property	
TfcImager.BitmapOptions.....	98
GetColorFromRGBString method	
TfcColorCombo.....	38
GetFieldValue method	
TfcDBTreeNode	54
GetFirstChild method	
TfcTreeNode	163
GetFirstNode method	
TfcTreeNodes.....	170
GetFirstSibling method	
TfcTreeView	194
GetHandle method	
TfcTreeNode	163
GetHitTestInfoAt method	
TfcDBTreeView	67
TfcTreeView	195
GetLastChild method	
TfcTreeNode	163
GetNext method	
TfcTreeNode	163
GetNextChild method	
TfcTreeNode	163
GetNextSibling method	
TfcTreeNode	164
GetNextVisible method	

TfcTreeNode	164
GetNode method	
TfcTreeNodes.....	170
GetNodeAt method	
TfcDBTreeView.....	68
TfcTreeView	195
GetPanelFromPt method	
TfcStatusBar	128
GetPrev method	
TfcTreeNode	164
GetPrevChild method	
TfcTreeNode	164
GetPrevSibling method	
TfcTreeNode	164
GetPrevVisible method	
TfcTreeNode	164
GetRect method	
TfcStatusPanel	133
GlyphOffset property	
TfcOutlookList	112
GlyphX property	
TfcImageBtn.Offsets.....	86
GlyphY property	
TfcImageBtn.Offsets.....	86
Grayed property	
TfcTreeNode	157
Grayscale property	
TfcImager.BitmapOptions.....	98
GreyScaleIncrement property	
TfcColorList.....	41
TfcColorList.Options	43

H

Handle property	
TfcTreeNode	157
TfcTreeNodes.....	166
HasAsParent method	
TfcTreeNode	164
HasChildren property	
TfcDBTreeNode.....	52
TfcTreeNode	157
Header property	
TfcDBTreeView. <i>See TfcTreeHeader, See TfcTreeHeader, See TfcTreeHeader</i>	
Help	13
Exhaustive Index.....	13
How-To & Tips Sections	13

Implementation & Coding Examples...	13	TfcOutlookListItem	112
On-line help	13	TfcStatusPanel	131
Troubleshooting Section	13	TfcTreeNode	158
Hiding expand for childless nodes		Imager property	
TfcDBTreeView	71	TfcDBTreeView	59
Highlight property		TfcOutlookBar	108
TfcText	135	Images property	
Hint property		TfcDBTreeView	59
TfcOutlookListItem	112	TfcOutlookList	112
TfcStatusPanel	131	TfcStatusBar	126
HorizontallyFlipped property		TfcTreeCombo	149
TfcImager.BitmapOptions.....	98	TfcTreeHeader	172
Hot property		TfcTreeView	182
TfcDBTreeNode.....	53	ImmediateHints property	
HotTrack property		TfcFontCombo	77, 78
TfcTreeHeader	172	InactiveFocusColor property	
Hot-tracking		TfcDBTreeView	59
TfcImageBtn.....	89	TfcDBTreeView	62
TfcLabel.....	104	TfcTreeView	182
Hot-tracking specific nodes		Increment property	
TfcDBTreeView	71	TfcTrackBar	141
HotTrackStyle property		Indent property	
TfcOutlookList	111	TfcStatusPanel	131
hslconHilite property		TfcTreeView	182
TfcOutlookList.HotTrackStyle	112	Index method	
hslItemHilite property		TfcTreeNode	165
TfcOutlookList.HotTrackStyle	112	Index property	
		TfcTreeNode	158
		InfoPower support	
		TfcCalcEdit.....	25
		TfcColorCombo.....	32
		TfcTreeCombo	146
		InitColorList method	
		TfcColorList.....	45
		Initializing	
		TfcCalcEdit.....	30
		TfcColorCombo.....	38
		Insert method	
		TfcTreeNodes.....	170
		InsertObject method	
		TfcTreeNodes.....	170
		installation	5
		requirements	5
		step-by-step.....	6
		Installation	
		On-line Help	
		Delphi.....	8
		Tip	9
		Integration	
		TfcDBImager.....	51

I

icoEndNodesOnly property			
TfcTreeCombo.Options	149		
icoExpanded property			
TfcTreeCombo.Options	149		
ifUseWindowsDrag property			
TfcImageForm.Options	92		
Image combo			
TfcTreeCombo	152		
Image property			
TfcImageBtn.....	85		
ImageAlignment event			
TfcTreeHeaderSection	175		
ImageDown property			
TfcImageBtn.....	85		
ImageIndex event			
TfcTreeHeaderSection	176		
ImageIndex property			
TfcDBTreeNode.....	53		

TfcImager.....	102
Interval property	
TfcOutlookBar.Animation.....	107
Invalidate method	
TfcStatusBar.....	128
InvalidateClient method	
TfcDBTreeView.....	68
InvalidateNode method	
TfcDBTreeView.....	68
TfcTreeView.....	195
InvalidateRow method	
TfcDBTreeView.....	68
Inverted property	
TfcImager.BitmapOptions.....	98
TfcTrackBar.....	141
IsCustomColor method	
TfcColorCombo.....	38
IsDroppedDown method	
TfcCalcEdit.....	30
TfcColorCombo.....	38
TfcTreeCombo.....	152
IsEditing method	
TfcTreeView.....	196
IsRadioGroup method	
TfcTreeNode.....	165
IsSelectedRecord method	
TfcDBTreeView.....	68
IsValidNode method	
TfcTreeCombo.....	152
IsVisible property	
TfcTreeNode.....	158
Item property	
TfcTreeNode.....	158
TfcTreeNodes.....	166
ItemDisabledTextColor property	
TfcOutlookListItem.....	113
ItemHighlightColor property	
TfcOutlookList.....	114
ItemHotTrackColor property	
TfcOutlookList.....	114
ItemID property	
TfcTreeNode.....	158
ItemIndex property	
TfcColorList.....	41
ItemLayout property	
TfcOutlookList.....	114
113	
Items property	
TfcColorList.....	40, 42

TfcOutlookList.....	112
TfcTreeCombo.....	149
TfcTreeView.....	182
ItemShadowColor property	
TfcOutlookList.....	114
ItemSpacing property	
TfcOutlookList.....	114
ItemsWidth property	
TfcOutlookList.....	114
Iterating through colors	
TfcColorCombo.....	31, 38
Iterating through descendants	
TfcTreeView.....	198
Iterating through items	
TfcButtonGroup.....	23
Iterating through multiselection	
TfcDBTreeView.....	72
Iterating through nodes	
TfcTreeView.....	198

L

LastVisibleDataSet property	
TfcDBTreeView.....	59
Layout property	
TfcButtonGroup.....	21
TfcOutlookList.....	114
Level property	
TfcDBTreeNode.....	53
TfcTreeNode.....	159
LevelIndent	
TfcDBTreeView.....	60
License Agreement.....	2
Lightness property	
TfcImager.BitmapOptions.....	98
Limiting color choices	
TfcColorList.....	45
LineColor property	
TfcDBTreeView.....	60
TfcTreeView.....	183
LineSpacing property	
TfcText.....	135
LoadFromBitmap method	
TfcBitmap.....	16
LoadFromFile method	
TfcTreeView.....	196
LoadFromStream method	
TfcTreeView.....	196

M

MakeActiveDataSet method	
TfcDBTreeView.....	68
MakeVisible method	
TfcTreeNode.....	165
Margin property	
TfcStatusPanel.....	131
Max property	
TfcProgressBar.....	118
TfcTrackBar.....	141
MaxControlSize property	
TfcButtonGroup.....	21, 23, 24
MaxMRU property	
TfcFontCombo.....	78
MaxWidth event	
TfcTreeHeaderSection.....	176
Min property	
TfcProgressBar.....	118
TfcTrackBar.....	141
MinWidth event	
TfcTreeHeaderSection.....	176
MouseEnterSameAsFocus property	
Frame.....	75
MouseOnItem property	
TfcOutlookListItem.....	113
MoveTo method	
TfcDBTreeView.....	68
TfcTreeNode.....	165
Multiline captions	
TfcImageBtn.....	89
TfcLabel.....	104
MultiSelect property	
TfcColorList.....	46
MultiSelectAttributes property	
TfcDBTreeView.....	60
TfcTreeView.....	183
MultiSelectCheckbox property	
TfcTreeView.MultiSelectCheckbox..	184
MultiSelectCheckBox property	
TfcDBTreeView.MultiSelectAttributes60	
MultiSelected property	
TfcDBTreeNode.....	53
TfcTreeNode.....	159
Multi-selection	
TfcTreeView.....	196
MultiSelectLevel property	
TfcDBTreeView.MultiSelectAttributes60	
TfcTreeView.MultiSelectAttributes..	183

MultiSelectList property	
TfcDBTreeView.....	61
TfcTreeView.....	184
MultiSelectListCount property	
TfcDBTreeView.....	61
TfcTreeView.....	184

N

Name property	
TfcStatusPanel.....	131
NearColor property	
TfcText.ExtrudeEffects.....	135
Nodes editor	
TfcTreeView.....	178
NoneString property	
TfcColorList.....	42
NonFocusBorders property	
EditFrame.....	74
Frame.....	74
NonFocusColor property	
EditFrame.....	74
Frame.....	74
NonFocusFontColor property	
EditFrame.....	74
Frame.....	74
NonFocusStyle property	
EditFrame.....	74
Frame.....	74
NonFocusTextOffsetX property	
EditFrame.....	74
Frame.....	74
NonFocusTextOffsetY property	
EditFrame.....	74
Frame.....	74
NonFocusTransparentColor property	
EditFrame.....	75
Frame.....	75
Nonrectangular forms.....	91
NumGlyphs property	
TfcImageBtn.....	85

O

Offsets	
TfcImageBtn.....	90
Offsets property	

TfcImageBtn.....	86	OnDrawText event	
OnAddFont event		TfcDBTreeView.....	65
TfcFontCombo.....	79	TfcStatusPanel.....	133
OnAddNewColor event		TfcTreeView.....	189
TfcColorCombo.....	36	OnDrawTickText event	
TfcColorList.....	44	TfcTrackbar.....	144
OnCalcNodeAttributes event		OnDropDown event	
TfcDBTreeView.....	53, 54, 63, 71, 72	TfcColorCombo.....	37
TfcTreeCombo.....	151	TfcTreeCombo.....	151
TfcTreeView.....	187	OnEdited event	
OnCalcPictureType event		TfcTreeView.....	190
TfcDBImager.....	50	OnEditing event	
OnChange event		TfcTreeView.....	191
TfcButtonGroup.....	22, 24	OnExpanded event	
TfcDBTreeView.....	64	TfcTreeView.....	191
TfcOutlookBar.....	109	OnExpanding event	
TfcProgressbar.....	119	TfcTreeView.....	191
TfcTrackbar.....	145	OnFilterColor event	
TfcTreeHeader.....	173	TfcColorCombo.....	31, 37, 38
TfcTreeView.....	188	TfcColorList.....	40, 44
OnChanging event		OnGenerateFontHint event	
TfcButtonGroup.....	22	TfcFontCombo.....	80
TfcOutlookBar.....	109	OnGetImageIndex event	
TfcTreeView.....	188	TfcTreeView.....	191
OnCheckValidItem event		OnGetSelectedIndex event	
TfcTreeCombo.....	151	TfcTreeView.....	192
OnCloseColorDialog event		OnInitColorDialog event	
TfcColorCombo.....	36	TfcColorCombo.....	37
OnCloseUp event		OnItemChange event	
TfcColorCombo.....	36	TfcOutlookList.....	116
TfcTreeCombo.....	151	TfcTreeView.....	192
OnCollapsed event		OnItemClick event	
TfcTreeView.....	188	TfcOutlookList.....	116
OnCollapsing event		On-line help.....	13
TfcTreeView.....	188	OnMouseDown event	
OnCompare event		TfcDBTreeView.....	66
TfcTreeView.....	189	TfcTreeView.....	192
OnDblClick event		OnMouseEnter event	
TfcDBTreeView.....	64, 66	TfcImageBtn.....	89
TfcTreeView.....	192	TfcLabel.....	104
OnDeletion event		OnMouseLeave event	
TfcTreeView.....	189	TfcImageBtn.....	89
OnDrawItem event		TfcLabel.....	104
TfcOutlookList.....	115	OnMouseMove event	
OnDrawKeyBoardState event		TfcDBTreeView.....	54, 66
TfcStatusBar.....	127	TfcTreeView.....	193
OnDrawPanel event		OnMouseUp event	
TfcStatusBar.....	127	TfcDBTreeView.....	66
OnDrawSection event		TfcTreeView.....	192
TfcDBTreeView.....	64	OnSectionClick event	

TfcTreeHeader	173
OnSectionDrag event	
TfcTreeHeader	173
OnSectionResize event	
TfcTreeHeader	174
OnSectionTrack event	
TfcTreeHeader	174
OnSelChange event	
TfcImageBtn.....	89
OnSelectionChange event	
TfcFontCombo	80
TfcTreeCombo	151
OnSetCalcButtonAttributes event	
TfcCalcEdit	29
OnTextChanged event	
TfcStatusPanel	133
OnToggleCheckbox event	
TfcTreeView	193
OnUserCollapse event	
TfcDBTreeView.....	66
OnUserExpand event	
TfcDBTreeView.....	67
Options property	
TfcCalcEdit	27
TfcColorList.....	42
TfcDBTreeView.....	61
TfcImageBtn.....	86
TfcImageForm.....	92
TfcOutlookBar	108
TfcText	136
TfcTreeCombo	149
TfcTreeHeader	172
TfcTreeView	184
Orientation property	
TfcProgressBar.....	119
Orientation property	
TfcShapeBtn	122
Orientation property	
TfcText.ExtrudeEffects.....	135
Orientation property	
TfcTrackBar	141
OutlineColor property	
TfcText	136
OutlookBar property	
TfcOutlookPage.....	108
OutlookItems property	
TfcOutlookBar	108
OutlookList property	
TfcOutlookPage.....	108
Owner property	

TfcTreeNode	159
TfcTreeNodes.....	167

P

paBottom property	
TfcOutlookBar.PanelAlignment.....	109
paDynamic property	
TfcOutlookBar.PanelAlignment.....	109
PageSize property	
TfcTrackBar	141
PaintCanvas property	
TfcOutlookList	114
Painting performance	
TfcImageBtn.....	90
Panel property	
TfcOutlookPage.....	108
PanelAlignment property	
TfcOutlookBar	109
PanelByName method	
TfcStatusPanels.....	127
PanelColor property	
TfcCalcEdit.....	28
Panels property	
TfcStatusBar	127
paperless forms.....	73
Parent property	
TfcDBTreeNode.....	53
TfcTreeNode	159
ParentClipping property	
TfcImageBtn.....	87, 90
paTop property	
TfcOutlookBar.PanelAlignment.....	109
picture masks	
supporting components	73
Picture property	
TfcDBImager.....	50
TfcImageForm.....	92
TfcImager.....	100
PictureType property	
TfcDBImager.....	50
PointList property	
TfcShapeBtn	123
PopupMenu property	
TfcStatusPanel	131
Position property	
TfcTrackBar	141
PreLoad property	

TfcFontCombo	78
PreProcess property	
TfcImager	100
Preventing multiselect highlighting	
TfcDBTreeView	72
Progress property	
TfcProgressBar	119
Proportional Sizing	
TfcStatusBar	129
psCapsLock property	
TfcStatusPanel.Style	132
psComputerName property	
TfcStatusPanel.Style	132
psControl property	
TfcStatusPanel.Style	132
psDate property	
TfcStatusPanel.Style	132
psDateTime property	
TfcStatusPanel.Style	132
psHint property	
TfcStatusPanel.Style	132
psNumLock property	
TfcStatusPanel.Style	132
psOverWrite property	
TfcStatusPanel.Style	132
psRichEditStatus property	
TfcStatusPanel.Style	132
psScrollLock	
TfcStatusPanel.Style	132
psTextOnly property	
TfcStatusPanel.Style	132
psTime property	
TfcStatusPanel.Style	132
psUserName property	
TfcStatusPanel.Style	132

R

ReadOnly property	
TfcTrackBar	141
RecentFonts property	
TfcFontCombo	78
Reformatting displayed text	
TfcStatusBar	128
RefreshList method	
TfcColorCombo	38
Reload method	
TfcFontCombo	80

Removing region	
TfcImageBtn	90
requirements	5
RespectPalette property	
TfcImageBtn	87
TfcImager	101
Retrieving multiselect colors	
TfcColorList	46
RichEdit Line/Column info	
TfcStatusPanel	132
RightClickNode property	
TfcTreeView	186
RightClickSelect property	
TfcTreeView	186
Rotation property	
TfcImager.BitmapOptions	98
TfcText	136
RoundRectBias property	
TfcShapeBtn	123

S

Saturation property	
TfcImager.BitmapOptions	98
SaveToBitmap method	
TfcBitmap	16
SaveToFile method	
TfcTreeView	196
SaveToStream method	
TfcTreeView	196
Saving resources	
TfcImageBtn	89
ScrollButtonsVisible property	
TfcOutlookList	115
ScrollInterval property	
TfcOutlookList	115
Sections property	
TfcTreeHeader	172
Selected property	
TfcButtonGroup	21
TfcDBTreeNode	54
TfcOutlookList	115
TfcOutlookListItem	113
TfcTreeNode	159
TfcTreeView	186
SelectedColor property	
TfcCalcEdit	29
TfcColorCombo	35, 38

TfcColorList	43	ShowProgressText property	
SelectedFont property		TfcProgressBar	119
TfcFontCombo	78	Simulate windows taskbar	
SelectedIndex property		TfcButtonGroup	24
TfcTreeNode	160	SizeToDefault method	
SelectedNode property		TfcImageBtn	89
TfcTreeCombo	149	TfcShapeBtn	124
Selecting a button at Design time		SliderVisible property	
TfcButtonGroup	23	TfcTrackBar	142
SelectRecord method		Smooth property	
TfcDBTreeView	68	TfcProgressBar	119
SelEnd property		SmoothStretching property	
TfcTrackBar	142	TfcImager	101
SelStart property		SortBy property	
TfcTrackBar	142	TfcColorList	43
Seperation property		Sorted property	
TfcOutlookListItem	113	TfcFontCombo	78
SetSelectedNode method		TfcTreeCombo	149
TfcTreeCombo	152	SortList method	
Setting shade colors		TfcColorList	45
TfcShapeBtn	125	SortMultiSelectList method	
ShadeColor property		TfcDBTreeView	69
TfcText	136	SortType property	
ShadeColors property		TfcTreeView	186
TfcImageBtn	87	source code..... <i>See</i> 1stClass source code	
ShadeStyle property		SpacingEdgeTrackbar property	
TfcImageBtn	88	TfcTrackBar	142
Shadow property		SpacingLeftTop property	
TfcImageBtn.ShadeColors	88	TfcTrackBar	142
TfcText	136	SpacingRightBottom property	
Shape property		TfcTrackBar	142
TfcShapeBtn	124	Sponge property	
Sharpen property		TfcImager.BitmapOptions	99
TfcImager.BitmapOptions	99	StateImages property	
ShowButton property		TfcDBTreeView	63
TfcCalcEdit	29	TfcTreeCombo	150
TfcColorCombo	35	TfcTreeView	187
ShowButtons property		StateIndex property	
TfcOutlookBar	109	TfcDBTreeNode	54
ShowDownAsUp property		TfcTreeNode	160
TfcButtonGroup	21	StaticCaption property	
TfcOutlookBar	109	TfcShapeBtn	124
ShowFocusRect property		Step property	
TfcImager	101	TfcProgressBar	119
ShowFontHint property		StepIt method	
TfcFontCombo	78	TfcProgressbar	119, 120
ShowMatchText property		Steps property	
TfcColorCombo	35	TfcOutlookBar.Animation	107
TfcFontCombo	78	StoreDataUsing property	
TfcTreeCombo	149		

TfcTreeCombo	150
Striated property	
TfcText.ExtrudeEffects	135
StringData property	
TfcTreeNode	160
StringData2 property	
TfcTreeNode	160
Style event	
TfcTreeHeaderSection	176
Style property	
TfcColorCombo	36
TfcFontCombo	79
TfcStatusPanel	131
TfcText	137
TfcTreeCombo	150

T

TabOrder property	
TfcImager	101
TabStop property	
TfcImager	101
Technical Support	3
Text event	
TfcTreeHeaderSection	176
Text property	
TfcCalcEdit	29
TfcDBTreeNode	54
113	
TfcTreeNode	160
TextAlignment property	
TfcOutlookListItem	113
TextAttributes property	
TfcTrackBar	142
TextDownX property	
TfcImageBtn.Offsets	86
TextDownY property	
TfcImageBtn.Offsets	86
TextOptions	
TfcLabel	103
TextOptions property	
TfcImageBtn	88
TfcStatusPanel	133
TextX property	
TfcImageBtn.Offsets	86
TextY property	
TfcImageBtn.Offsets	86
TfcBitmap	16

Added methods	16
TfcButtonEffects	17
Properties	17
TfcButtonGroup	3, 18
Added Events	22
Added Methods	22
Added Properties	19
Changing selected button color	24
Conserving resources when using Image button	23
Constraining width or height	23
Design-time aids	18, 23
How To	23
Iterating through items	23
Removing auto bold effect	23
Selecting a button at design time	23
Simulating a windows task bar	24
Using a TfcImageBtn	23
TfcButtonGroupItem	20
TfcButtonGroupItems	19
Added Methods	22
TfcCalcEdit	3, 25
Added Events	29
Added Methods	30
Added Properties	26
How To	30
InfoPower support	25
Initializing color	30
TfcColorCombo	3, 32
Added Events	36
Added Methods	37
Added Properties	33
Displaying in InfoPower Grid	38, 153
How To	38
InfoPower support	32
Initializing color	38
Iterating through colors	31, 38
ScreenShot	32
Tips	39
TfcColorList	3, 40
Added Events	44
Added Methods	45
Added Properties	40
Adding color dialog support	46
Aligning colors on the right	46
Dragging and Dropping a Color	47
How To	45
Limiting color choices	45
OnFilterColor event example	45
Retrieving multiselect colors	46

Tips.....	48	Handling tabbing and focus issues.....	90
Using the OnAddNewColor event.....	44	Hot-tracking.....	89
TfcDBImager.....	3, 49	How To.....	89
Added Events.....	50	Multiline captions.....	89
Added Properties.....	49	Painting appearance.....	87
How To.....	51	Painting performance.....	90
Integrating with TDBCtrlGrid.....	51	Removing region.....	90
Loading Pictures.....	51	Resource optimization.....	85
Using the OnCalcPictureType event....	51	Saving resources.....	89
TfcDBTreeNode.....	52	Switches.....	86
Added Methods.....	54	Text offsets.....	86
Added Properties.....	52	Tips.....	90
Hot-tracking.....	53	Using as TButton or TBitBtn.....	90
TfcDBTreeView.....	3, 4, 56	Using as TSpeedButton.....	90
Added Events.....	63	TfcImageForm.....	3, 91
Added Methods.....	67	Added Properties.....	92
Added Properties.....	57	Decreasing size of executable.....	94
Background image.....	59	Designing the image.....	94
Changing node color based on field....	71	Drag control for the form.....	92, 94
Changing painting attributes.....	63	How To.....	94
Design-time aids.....	59	Loading images at runtime.....	93
Hiding expand icon for childless nodes	71	Tips.....	94
Hot-tracking specific nodes.....	71	TfcImager.....	3, 96
How To.....	69	256 Color palette issues.....	101
Iterating through multiselection.....	72	Accessing the working bitmap.....	102
Preventing multiselect highlighting....	72	Added Properties.....	96
Text and display format.....	58	Bitmap effects.....	49, 97
Using PopupMenu on selected node....	72	Blending two bitmaps.....	102
TfcEditFrame.....	73	Centering an image.....	100
TfcFontCombo.....	3, 76	Different Drawing Styles.....	100
Added Events.....	79	Effects	
Added Methods.....	80	AlphaBlend.....	97
Added Properties.....	76	Amount.....	97
TfcGroupBox.....	3, 81	Bitmap.....	97
Added Properties.....	81	Transparent.....	97
Focus/NonFocus Colors.....	82	Brightness.....	98
Tips.....	82	Color.....	97
TfcImageBtn.....	3, 83	Contrast.....	97
256 Color palette issues.....	87	Embossed.....	97
Added Events.....	89	Flip Horizontally.....	98
Added Methods.....	89	Flip Vertical.....	99
Added Properties.....	84	GaussianBlur.....	98
Caption and glyph offsets.....	90	Grayscale.....	98
Clipping.....	90	Invert.....	98
Design-time aids.....	83	Rotation.....	98
Set shade colors.....	83	Angle.....	98
Size to default.....	84	Saturation.....	98
Different up/down image shapes... 86,	90	Sharpness.....	99
Down text offsets.....	86	Sponge.....	99
Glyph offsets.....	86	Tint.....	99

Wave.....	99	Custom Shapes.....	123
How To.....	102	Defining a custom shaped button.....	124
Integrating with TfcDBTreeView.....	102	Design-time aids.....	121
Integrating with TfcOutlookBar.....	102	Flat-style buttons.....	125
Painting tips.....	100, 101	Focus color outline.....	125
Performance.....	100	How To.....	124
Proportional stretching.....	100	Setting shade colors automatically....	125
Stretching an image.....	100	Standard Shapes.....	124
Tiling an image.....	100	Tips.....	125
TfcLabel.....	3, 103	TfcStatusBar.....	4, 126
Added Events.....	104	Added Events.....	127
Added Properties.....	103	Added Methods.....	127
Centering captions.....	104	Added Properties.....	126
Hot-tracking.....	104	Automatic hints.....	128
Hot-tracking example.....	104	Formatting display of date/time.....	128
How To.....	104	How To.....	128
Multiline captions.....	104	Menu hints.....	128
Tips.....	105	Proportional Sizing.....	129
TfcOutlookBar.....	3, 106	Reformatting displayed text.....	128
Added Events.....	109	Tips.....	129
Added Methods.....	110	Using OnDrawText event.....	128
Added Properties.....	107	TfcStatusPanel.....	130
Animation performance issues.....	110	Added Events.....	133
Design-time aids.....	106	Added Methods.....	133
Create OutlookList.....	106	Added Properties.....	130
New button.....	106	Automatic Hints.....	132
Paste.....	106	Computer Info.....	132
Embedding your own controls.....	110	Current Date/Time.....	132
How To.....	110	Custom Controls.....	132
Tips.....	110	Keyboard States.....	132
TfcOutlookList.....	111	RichEdit Line/Column info.....	132
Added Events.....	115	TfcText.....	134
Added Properties.....	111	Added Properties.....	134
Customizing the appearance of items.....	115	Adding shadows to text.....	137
Emulating Outlook Express's OutlookBar.....	116	Disabled colors.....	134
How To.....	116	Extrusion effects.....	134
TfcOutlookPage.....	108	How To.....	137
TfcPanel.....	3, 117	Rotation.....	136
Added Properties.....	117	Text Shadow Settings.....	136
TfcProgressbar.....	118	Text Styles.....	137
TfcProgressBar.....	3	Tips.....	138
Added Events.....	119	TfcTrackbar.....	139
Added Methods.....	119	Added Events.....	144
Added Properties.....	118	Added Properties.....	140
TfcShapeBtn.....	3, 121	TfcTreeCombo.....	4, 146
Added Methods.....	124	Added Events.....	151
Added Properties.....	122	Added Methods.....	152
Changing buttons ability to receive focus.....	125	Added Properties.....	147
		How To.....	152
		Image combo.....	152

InfoPower support	146	TrackColor property	
Initializing an unbound TfcTreeCombo	153	TfcTrackBar	144
Iterating through list of nodes	153	TrackPartialFillColor property	
Make only end nodes selectable	153	TfcTrackBar	144
Non-heiarchical combobox	152	TrackThumbIcon property	
TfcTreeHeader		TfcTrackBar	144
Added Events	172	transparency	
Added Properties	171	supporting components	73
TfcTreeHeaderSection		Transparent property	17
Added Properties	175	EditFrame	75
TfcTreeNode	155	Frame	75
Methods	161	TfcButtonGroup	21
Properties	155	TfcGroupBox	82
TfcTreeNodes	166	TfcImager	101
Methods	167	TfcOutlookList	115
Properties	166	TfcPanel	117
TfcTreeView	4, 177	TransparentColor property	
Added Events	187	TfcImageBtn	88
Added Methods	193	TfcImageForm	93
Added Properties	181	TfcImager	101
Design-time aids	178	Tree property	
Item Properties	179	TfcTreeHeader	172
Items group box	178	TreeOptions property	
How To	196	TfcFontCombo	79
Iterating through descendants	198	TfcTreeCombo	150
Iterating through nodes	198	TreeView property	
Multi-selection	196	TfcFontCombo	79
Nodes editor	178	TfcTreeCombo	150
URL Links	197	TfcTreeNode	161
ThumbColor property		Turning off AutoBold	
TfcTrackBar	143	TfcButtonGroup	23
ThumbLength property		tvctCheckbox property	
TfcTrackBar	143	TfcTreeNode.CheckboxType	155
ThumbThickness property		tvctNone property	
TfcTrackBar	143	TfcTreeNode.CheckboxType	155
TickMarks property		tvctRadioGroup property	
TfcTrackBar	143	TfcTreeNode.CheckboxType	155
TickStyle property		tvto3StateCheckbox property	
TfcTrackBar	144	TfcTreeView.Options	186
TintColor property		tvtoAutoURL property	
TfcImager.BitmapOptions	99	TfcTreeView.Options	160
toFullJustify property		tvtoEditText property	
TfcText.Options	136	TfcTreeView.Options	186
TopItem property		tvtoExpandButtons3D property	
TfcTreeView	187	TfcTreeView.Options	184
toShowAccel property		tvtoExpandOnDbClick property	
TfcText.Options	136	TfcTreeView.Options	184
toShowEllipsis property		tvtoFlatCheckBoxes property	
TfcText.Options	136	TfcTreeView.Options	184
		tvtoHideSelection property	

TfcTreeView.Options	185
twoHotTrack property	
TfcTreeView.Options	185
twoRowSelect property	
TfcTreeView.Options	185
twoShowButtons property	
TfcTreeView.Options	185
twoShowLines property	
TfcTreeView.Options	185
twoShowRoot property	
TfcTreeView.Options	185
twoToolTips property	
TfcTreeView.Options	185

U

UnboundAlignment property	
TfcColorCombo	36
underline controls	73
uninstalling 1stClass	10
UnselectAll method	
TfcDBTreeView	69
TfcTreeView	196
UnselectRecord method	
TfcDBTreeView	69
UpdateShadeColors method	
TfcImageBtn	89, 125
URL Links	
TfcTreeView	197
Using a TfcImageBtn	
TfcButtonGroup	23
Using as TButton or TBitBtn	
TfcImageBtn	90
Using as TSpeedButton	
TfcImageBtn	90
Using PopupMenu	
TfcDBTreeView	72

V

Valignment property	
TfcText	137
VerticallyFlipped property	
TfcImager.BitmapOptions	99
Visible property	
TfcImageForm	93
TfcOutlookListItem	113

W

Wave property	
TfcImager.BitmapOptions	99
Width event	
TfcTreeHeaderSection	176
Width property	
TfcStatusPanel	133
WordWrap property	
TfcText	137
WorkBitmap property	
TfcImager	102

X

XOffset property	
TfcText.Shadow	137

Y

Yoffset property	
TfcText.Shadow	137