TBX PACKAGE

An Add-On Package for Toolbar2000

version 2.1 beta 1
May 29, 2004

Alex A. Denisov

Contents

CHAPTER 1

Introduction

Features

TBX is an add-on package for Toolbar2000 components (Copyright © 1998–2004 Jordan Russell), a shareware package available at http://www.jrsoftware.org/tb2k.htm.

TBX expands Toolbar2000 with the following new features:

- Support for native themes and WindowsXP visual styles
- Customizable layout for toolbar items
- Variations of font size for toolbar items
- Multi-line captions
- Combo and list boxes
- Dockable panels (task panes)
- Tool palettes and color selectors
- A status bar component
- Additional controls

License

This notice is only applicable to TBX package; it does not apply to Toolbar2000 package, where the separate license is provided.

TBX is distributed as freeware. You are free to use it in any kind of application as well as redistribute the source code in original form, provided the following conditions are met:

- All redistributions of source code must be in the original form, they must include all the original files, including but not limited to original documentation, license agreement, copyright notice, and web site address.
- Redistribution of TBX package or any part of TBX code in modified form is allowed only with express written permission from the author (Alex A. Denisov).
- You are not allowed to make modifications to TBX documentation. You may only redistribute the documentation with original unmodified package.
- If TBX is used in your software in the compiled form, an acknowledgement in product documentation and/or in the About box is required. Only registered users are allowed to omit the acknowledgement.
- The origin of this software must not be misrepresented; you must not claim your authorship.

If you do not agree to all of the above terms, you are not permitted to use this package.

This software is provided 'as-is', without warranty of any kind, either expressed or implied. In no event shall the author be held liable for any damages arising from the use of this software.

Alex A. Denisov

alex@g32.org

http://g32.org

Registration and Donations

The TBX package is a freeware product and you can use it for any kind of applications provided the License conditions are satisfied. Registration is required only if you don't want to include the acknowledgement. The registration fee is US$100 for a single user and US$200 per site. Navigate to https://order.kagi.com/?GRH to register.

As long as you include an acknowledgement, you can use the TBX package free of charge even for commercial and shareware applications. However, if you wish to express your appreciation for the time I spend on developing, documenting and supporting it, I do accept and appreciate donations.

If you wish to make your donation, navigate to the same site https://order.kagi.com/?GRH. The base amount is US$20, but if you would like to donate more, feel free to order multiple copies of the "TBX donation" product.

Thank you for your support.

Contacts

TBX home page is located at: http://g32.org

The discussion on TBX package as well as other Toolbar2000 extensions is hosted by Jordan Russel's news server at jrsoftware.toolbar2000.thirdparty. Please, address general questions to the newsgroup.

Direct e-mail contacts: alex@g32.org.

For information on Toolbar2000 components, visit http://www.jrsoftware.org.

CHAPTER 2

Usage

Installing TBX Package

TBX is an add-on package, you have to install it into IDE which already has Toolbar2000 packages installed. However, original Toolbar2000 code is not completely compatible with TBX and several changes should be made to Toolbar2000 sources *before* installation. The list of changes is included into the TBX distribution as a CVS-compatible diff file.

The easiest way to install the modified Toolbar2000 and TBX is a clean installation. If you already have Toolbar2000 (modified or unmodified) installed, it is recommended to uninstall it first. When upgrading to a new version of Toolbar2000 or TBX, you may occasionally run into internal IDE errors which will require complete uninstallation of both packages.

To uninstall existing Toolbar2000 and TBX, complete the following procedure:

1. in Delphi IDE, go to **Component->Install Packages**
2. find and remove all TB2K and TBX packages
3. close Delphi IDE
4. go to **$(DELPHI)\Projects\BPL** and delete **tb2k*.dcp**, **tb2k*.bpl**, **tbx*.dcp**, and **tbx*.bpl** files
5. restart Delphi IDE

Now, when all Toolbar2000 and TBX packages has been completely removed, Delphi should start up without any problems.

To install both Toolbar2000 and TBX:

1. Go to http://www.jrsoftware.org and download the latest version of Toolbar2000. Make sure this version is the same as the version of the patch in **$(tbx)\Diffs**. For example, for Toolbar2000 version 2.0.16, this directory should contain files **_cvs_patch-2_0_16.bat** and **_cvs_patch-2_0_16.diff**.

2. Locate files **Patch.exe**, **_cvs_patch-*.bat**, and **_cvs_patch-*.diff** in the **$(tbx)\Diffs** directory. Copy these files to **$(tb2k)\Source**.
3. Run the **_cvs_patch-*.bat** file to apply the patch. Original files before modification will be saved automatically with the '.orig' extension. You can use these files to undo the patching process.
4. Add Toolbar2000 Source directory to **Tools->Environment Options->Library->Library Path**.
5. Compile and install the **tb2kdsgn_*** package corresponding to IDE version. If a dialog box pops up asking you to save the package, choose 'No'.
6. Add TBX installation directory to **Tools->Environment Options->Library->Library Path**.
7. Compile and install the **tbxdsgn_*** package corresponding to IDE version. If a dialog box pops up asking you to save the package, choose 'No'.
8. Restart IDE.

When upgrading Toolbar2000 and TBX, the procedure of complete uninstall can often be skipped. Chances are that IDE will fire up a series of error messages but the packages will be installed properly. Sometimes, however, this does not work due to unknown internal errors in the IDE and the only way to resolve these problems is to uninstall Toolbar2000 and TBX as described above.

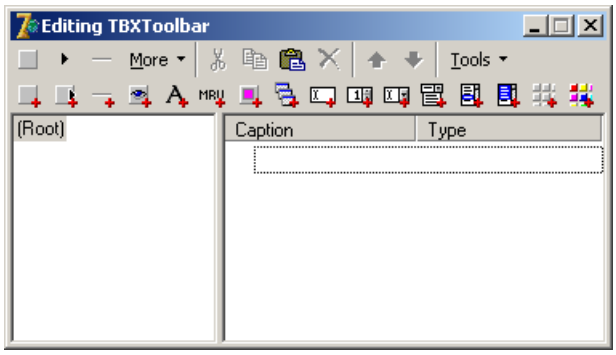
If you use CVS to keep Toolbar2000 up to date, notice that each release of TBX already includes all the changes introduced between the Toolbar2000 release and TBX release dates. Normally, you should be able to continue updating Toolbar2000 sources using Toolbar2000 CVS repository even after the TBX changes have been applied. Unless, of course, those updates do not interfere with TBX modifications.

Getting Started

If you are using the TBX Package for the first time, a good starting point would be to read through the Toolbar2000 documentation (**tb2k.chm** file in the root directory of Toolbar2000 installation). In this section, I will briefly outline the differences when using TBX.

After installing both runtime and design time TBX packages, you can find new VCL components in the Toolbar2000 tab together with the original Toolbar2000 components. All TBX icons are marked with a small red '+' sign.

The toolbar designer contains an additional toolbar with TBX items. These items are also marked with a small '+' sign.



When creating applications with TBX Package, instead of the original Toolbar2000 controls and items, you should use their TBX analogues. The only exceptions are TTBGroup, and TTBItem-Container which remain in TBX projects. You can easily tell where does a component come from: if the class name starts with TTB, the component belongs to Toolbar2000; class names starting from TTBX belong to TBX.

Using TTBImageList is not recommended. TBX uses its own internal routines to display images in normal, hot, and disabled states, therefore the concept of predefined state images does not work. Use the standard TImageList instead.

By default, TBX only supports the 'Default' theme. To access other themes, add the TTBXSwitcher component to the main application form. To change the theme, set the Theme property to 'OfficeXP' or 'Stripes'.

Note: At design time, all the registered themes are accessible through the Object Inspector window in IDE. **To make themes accessible at runtime, you have to include corresponding theme units**.

In Delphi, make sure theme references are included into the uses clause in the project, or the main unit of your application:

```
uses TTBXOfficeXPTheme, TTBXStripesTheme;
```

In C++ Builder, add:

```
#pragma link "TBXOfficeXPTheme"
#pragma include "TBXOfficeXPTheme.pas"
#pragma link "TBXStripesTheme"
#pragma include "TBXStripesTheme.pas"
```

Sorry. The rest is under construction...

TBX Themes

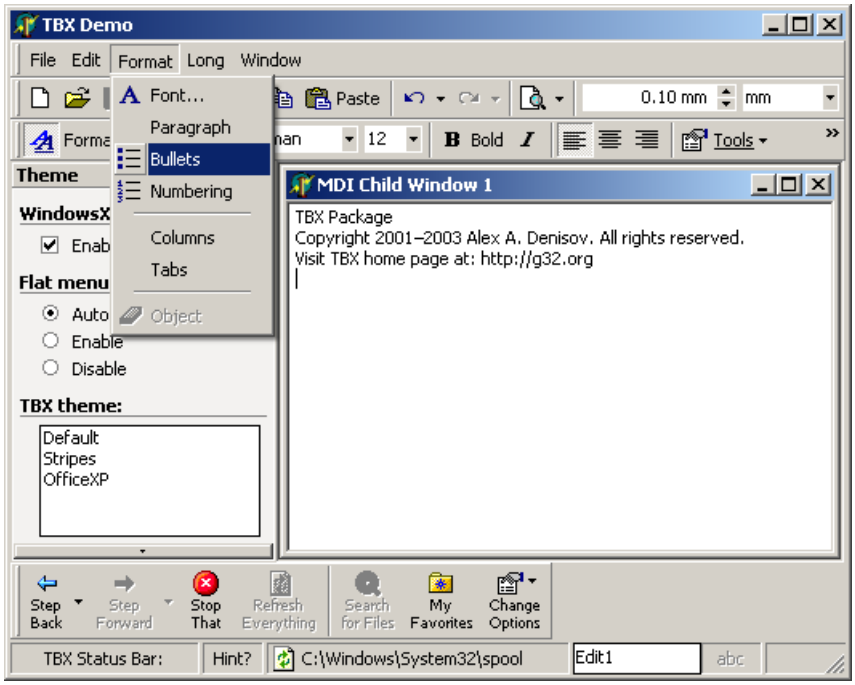
There are three themes included with TBX distribution: Default, OfficeXP, and Stripes. Additional third party themes are also available at http://www.rmklever.com/.

In your application, to make themes available at runtime, include theme units into the 'uses' clause of any unit in your application or into the project file.

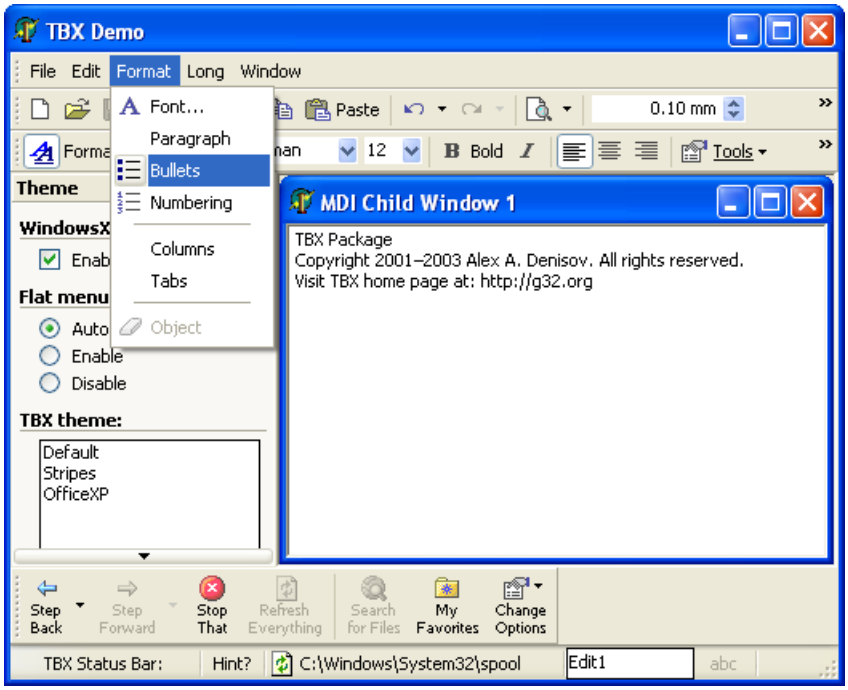
To change the themes, either use the TTBXSwitcher component, or one of the lower level functions TBXCurrentTheme and TBXSetTheme.

Default Theme

The Default theme replicates the appearance and behavior of toolbars and menus in Microsoft Office 2000.



This theme features behavior almost identical to that of standard Toolbar2000 components. In addition, it has a built in support for Windows XP visual styles:

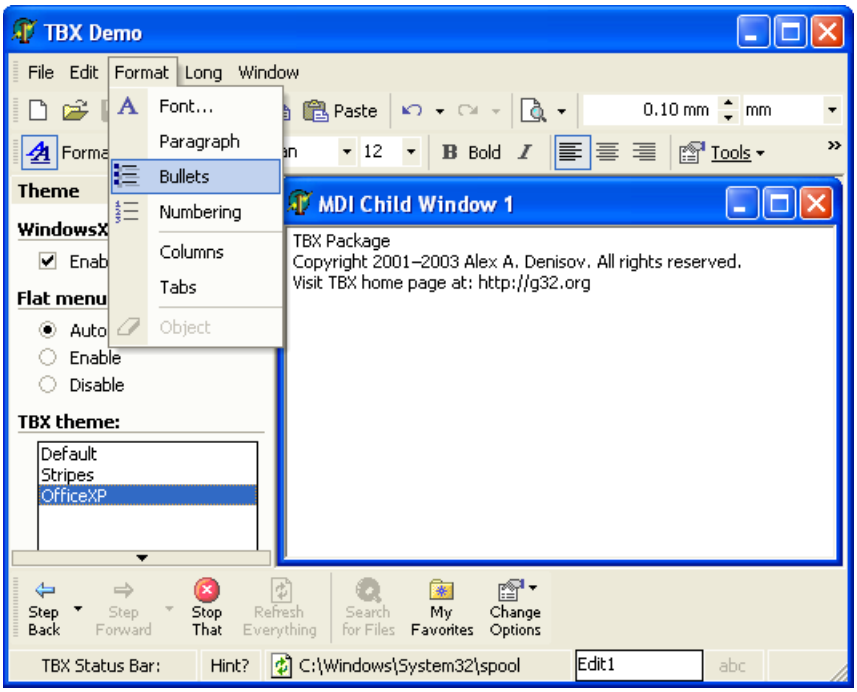


Specific issues:

- Disabled items are highlighted only when user selects them with keyboard cursor. When the mouse hovers over disabled item, it is not highlighted as in most windows applications;
- Support for flat menus in WindowsXP has some differences from that in Toolbar2000 package.

Office XP Theme

The OfficeXP Theme closely recreates the toolbar and menu appearance introduced in Microsoft OfficeXP.

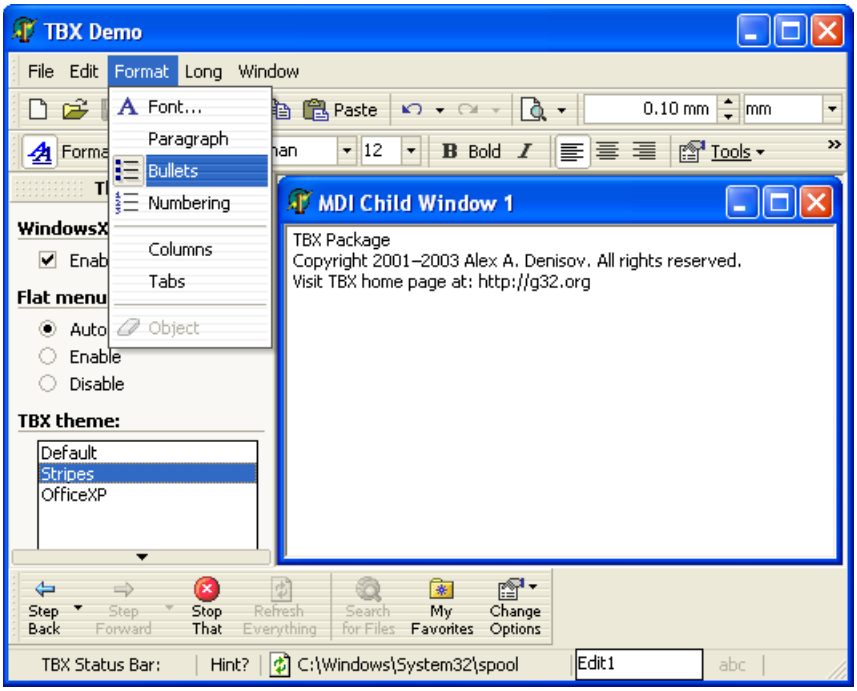


Specific issues:

- The color of item and menu captions is no longer controlled by the Font property. It is chosen automatically by the theme to be in contrast with background
- In high-contrast desktop schemes, this theme may invert some colors in toolbar and menu images to keep images contrast
- The algorithm for choosing contrast colors may be slightly different from that in Microsoft Office XP

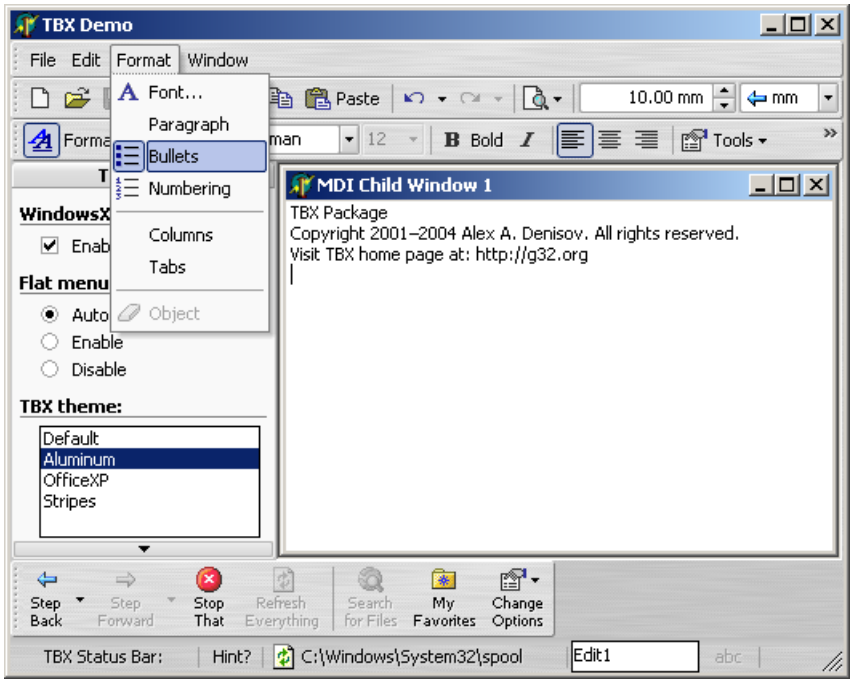
Stripes Theme

This theme was in inspired by the Mac version of MS Office.



Aluminum Theme

Aluminum theme combines the look of brushed aluminum with gradients.



Specific issues:

- Due to an extensive amount of graphical details, this theme may be slow on low end computers.

Converterting and Upgrading Existing Projects

When converting or upgrading existing projects to TBX, there is an elaborate task of renaming existing Toolbar2000 components to their TBX analogues.

To facilitate such a conversion, the TBX Converter has been written. This is a small application located in directory **$(TBX)\Converter**, which assists in renaming components and properties. Several types of files are supported:

pas:
pascal source files

h, hpp:
C++ header files

cpp:
C++ source files

dfm:
form files (both binary and text)

Conversion Rules

The conversion process is defined by the set of replacement operations. These replacement operations can be found in the **conversions.ini** file. This is a simple text file where replacement operations are grouped into sets of conversion rules. Consider, an example of such a rule:

```
[Convert TB2K>TBX]
'Convert TB2K controls:
TTBToolbar -> TTBXToolbar
TTBToolWindow -> TTBXToolWindow
TTBItem -> TTBXItem
```

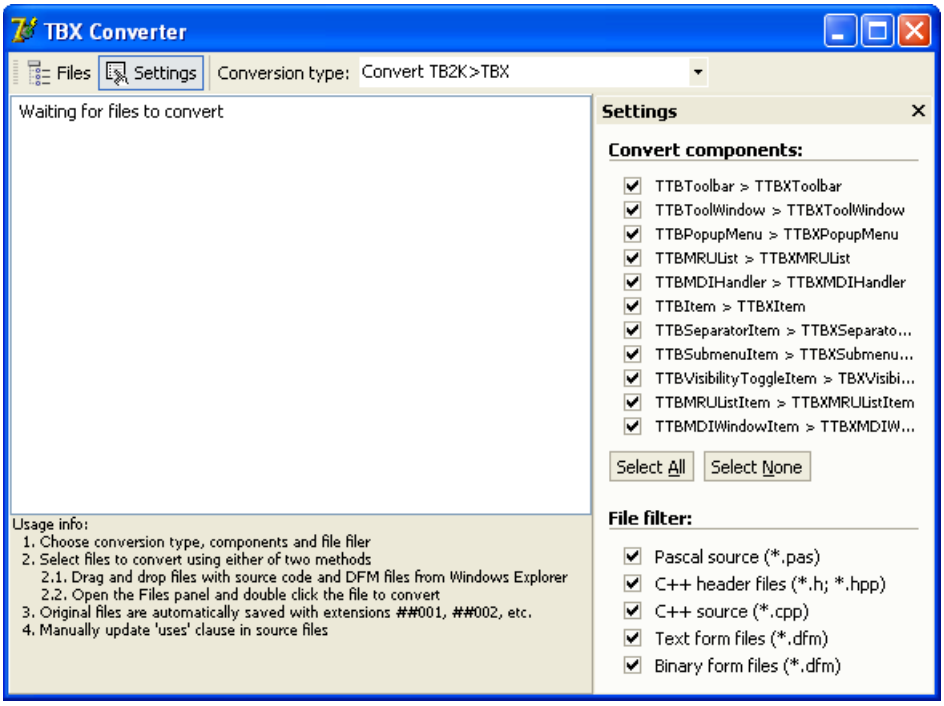
The first line here is the name of the conversion rule. All the content after this line and before the next conversion rule (or the end of the file) defines replacement operations. The line starting with the apostrophe is a comment. The rest of the lines have a simple form of '*old name* -> *new name*'. After reading **conversions.ini** and choosing an appropriate conversion rule, TBX Converter replaces all the occurences of the *old name* with the *new name*.

Note: All the files will be backed up before the conversion, so there is always a way to restore the project to its previous stage if conversion is not successful. However, it is always a good idea to back up the entire project.

Running the TBX Converter through your projects does not mean that those projects will automatically become fully compatible with the TBX Package. Additional modification may still be required but at least, that can be done in IDE without Delphi (or C++ Builder) complaining about opening projects with unknown components.

Using TBX Converter

Let's take a look at the screen shot of the TBX Converter



Before choosing the files for conversion, an appropriate conversion rule must be chosen in the Conversion type combo box:

Upgrade TBX:
Use this rule to upgrade projects with older TBX components (prior to TBX version 2.0)

Convert TB2K>TBX + Upgrade TBX:
Changes Toolbar2000 components to their TBX counterparts and upgrades older TBX components

Convert TB2K>TBX:
Changes Toolbar2000 components to their TBX counterparts

Convert TBX>TB2K:
Changes TBX components back to their Toolbar2000 analogues

The exact set of replacement operations is shown in the 'Settings' side panel. Here you can also enable or disable some of the operations. In the same panel, you can choose file filters. TBX Converter will only process the files which pass through these filters to avoid accidental damage to unsupported files.

After the setup is complete, the TBX Converter is ready to accept files for conversion. You can simply drag and drop files or groups of files from the Windows Explorer, or click the files button and choose the files in the TBX Converter. As the conversion process proceeds, you will see the progress messages in the text area of the application.

Known Issues

Here is the list of issues with TBX Package which have not been yet resolved:

TTBXStatusBar and form inheritance is not functioning properly:
At present, this is unresolved issue.

Support for BiDiMode flags:
The main problem is that Toolbar2000 itself does not support the right-to-left alignment. I started implementing partial support for it, but it looks like substantial changes in Toolbar2000 code are required as well.

Support for OwnerDraw events in toolbar items:
Working on it...

Problems while installing/using TBX with C++ Builder:
I do not use Borland C++ Builder for development, therefore I cannot resolve all the problems. However, I'm trying to collect the information from the newsgroup. Other people who have been able to install TBX in C++ Builder do occasionally provide information on corrections and changes, which I'm trying to include with each new release.

Image lists with alpha channel are not supported:
Most likely, this issue will not be resolved until Borland comes up with a decent TImageList implementation that supports alpha channel

Frequently Asked Questions

Here is the list of common questions and answers which regularly appear in e-mails and newsgroups.

***Q**: Why compiler keeps complaining that TTBPopupPositionRec is not found when I'm trying to install TBX?*

A: You have not patched the original Toolbar2000 sources or the patching procedure was unsuccessful. Make sure you are applying the correct patch to the correct version of Toolbar2000 in the correct directory. See installation instructions for additional information.

***Q**: Why cannot I access themes at runtime while at design time everything is ok?*

A: The only theme which is automatically registered at runtime is the 'Default' theme. You have to include theme units explicitly into the uses clause in your application (this can be done in any file, but I'd recommend including it either in the project file, or in the main unit file.

***Q:** Can I install TBX together with Toolbar2000 and TBSkin?*

A: No. TBX and TBSkin packages are not designed to work together. Patching Toolbar2000 sources with both patches will never succeed and you will end up with a code which will not compile. Visit the support newsgroup from time to time to see if there is any progress in the 'TBSkin++' package development.

***Q**: I want more themes. Where do I get them and how do I install them?*

A: Roy Magne Klever is maintaining a nice web site with additional themes at http://www.rmklever.com/. To make a new theme accessible in your application at runtime, just add the corresponding unit to the uses clause in your main form. To make it accessible at design time as well, install it into any design time package.

***Q**: I cannot access design time item editor after installing, upgrading or recompiling the TBX package. Is this a bug?*

A: Due to some internal IDE bugs you cannot use toolbar item editor after compiling the TBX package. You have to restart the IDE.

***Q**: Why does not TTBImageList work with TBX?*

A: TBX uses its own internal routines to display images in normal, hot, and disabled states, therefore the concept of predefined state images does not work. Use the standard TImageList component instead of TTBImageList.

For other questions and information, please visit the jrsoftware.toolbar2000.thirdparty newsgroup.

CHAPTER 3

Reference

Overview

- TBX.pas – Common TBX items and controls
- TBXDkPanels.pas – Provides support for dockable panels and some other controls
- TBXExtItems.pas – Contains additional controls and toolbar items
- TBXLists.pas – Implements support for list items
- TBXMDI.pas – Provides support for MDI
- TBXStatusBars.pas – Implements support for the TBX status bar
- TBXSwitcher.pas – Contains implementation of then TTBXSwitcher component
- TBXThemes.pas – Implements support for TBX themes
- TBXToolPals.pas – A set of classes to support tool palettes

Unit TBX.pas

TBX.pas contains some common TBX items and controls.

Quick reference

classes:

- TFontSettings – Contains modifications to default font properties
- TTBXCustomItem – A common parent for many other TBX items
- TTBXDock – A replacement for TTBDock
- TTBXItem – A theme-aware replacement for TTBItem

- TTBXPopupMenu – A TBX version of TPopupMenu
- TTBXSeparatorItem – A toolbar item for menu and toolbar separators
- TTBXSubmenuItem – A toolbar item for top level menus and submenus
- TTBXToolbar – A toolbar which supports TBX themes
- TTBXToolWindow – A TBX replacement for TTBToolWindow
- TTBXVisibilityToggleItem – An item which toggles visibility of other controls

routines:

- AddThemeNotification – Adds notification on theme changes to the object
- GetEffectiveColor
- RemoveThemeNotification – Removes notification on theme changes from the object
- TBXCurrentTheme
- TBXSetTheme

types:

- TTBXItemTransparency
- TTriState

variables:

- CurrentTheme

Class TFontSettings

Contains modifications to default font properties

Hierarchy

TPersistent
|
TFontSettings

Properties

Bold

published

```
property Bold: TTriState;
type TTriState = (tsDefault, tsTrue, tsFalse);
```

Controls the fsBold font style.

See also: TTriState

Color

published

```
property Color: TColor;
```

Indicates the color of the font. Default value is clNone, which indicates no color change.

published

Italic

```
property Italic: TTriState;
type TTriState = (tsDefault, tsTrue, tsFalse);
```

Controls the fsItalic font style

published

Name

```
property Name: TFontName;
```

Indicates the name of the font. When the Name property is empty, the name of the default or inherited font is used.

published

Size

```
property Size: TFontSize;
type TFontSize = 25..1000;
```

The size of the font relative to the default font size, in percent. By default, Size is set to 100, indicating the same size. Set size to 200, for example to increase the font size two times, or 50 to make it twice smaller.

published

StrikeOut

```
property StrikeOut: TTriState;
type TTriState = (tsDefault, tsTrue, tsFalse);
```

Controls the fsStrikeOut font style.

published

Underline

```
property Underline: TTriState;
type TTriState = (tsDefault, tsTrue, tsFalse);
```

Controls the fsUnderline font style.

Class TTBXCustomItem

A common parent for many other TBX items

Hierarchy

TTBCustomItem
|
TTBXCustomItem

Description

Derived from TTBCustomItem, this class serves as a common ancestor for many toolbar items in the TBX library. It overrides some methods of TTBCustomItem to provide more flexible painting end measurement.

In addition to standard behavior introduced in TTBCustomItem, TBXCustomItem supports TBX themes, multiline captions, and other enhancements. Unlike TTBXItem, TTBXCustomItem is designed for internal package purposes. It should not be used explicitly in applications.

Properties

AlwaysSelectFirst

property AlwaysSelectFirst: Boolean;

Indicates that the first subitem should always be automatically selected when the child popup opens. Normally, popup menus do not show selection until mouse is moved over the menu. This selection should be forced for combo-style items which have a list as a first subitem.

FontSettings

property FontSettings: TFontSettings;

Use font settings to change the appearance of the item caption.

These font setting are applied regardless of the item state. If you want more flexible control, use the OnAdjustFont event.

See also: Class TFontSettings

Layout

property Layout: TTBXItemLayout;
type TTBXItemLayout = (tbxlAuto, tbxlGlyphLeft, tbxlGlyphTop);

Controls alignment of parts within the item. This property is ignored when the item is shown inside the menu.

MinHeight

property MinHeight: Integer;

Minimum item height in pixels.

MinWidth

property MinWidth: Integer;

Minimum item width in pixels.

Stretch

property Stretch: Boolean;

Indicates whether the item dimensions should be resized to fit the tallest/widest item in the same toolbar.

ToolBoxPopup

protected

property ToolBoxPopup: Boolean;

ToolBoxPopup is used in *TTBXSubmenuItem* to alter the popup menu appearance. Use ToolBoxPopup when submenu contains TTBXStringList, TTBXToolPalette, or similar items.

Events

OnAdjustFont

protected

```
property OnAdjustFont: TAdjustFontEvent;
type TAdjustFontEvent = procedure(Item: TTBCustomItem; Viewer: TTBItemViewer; Font: TFont; StateFlags:
    Integer) of object;
```

The OnAdjustFont event allows customization of the item appearance individually for each toolbar item.

Item
Measured or painted item

Viewer
Corresponding viewer

Font
Modify font parameters to customize the caption appearance

StateFlags
Current state of the item

Before firing this event, the font is reset to the default font of the toolbar, then the FontSettings are applied. Therefore, all modifications are relative to the default font modified with FontSettings.

Note: Changing Font.Color has no effect for disabled items since the color of the font in disabled items is controlled by the theme.

The state of the item is combination of the following constants:

- ISF_DISABLED
- ISF_HOT
- ISF_PUSHED
- ISF_SELECTED
- ISF_MENUCOLOR

OnDrawImage

protected

```
property OnDrawImage: TDrawImageEvent;
type TDrawImageEvent = procedure(Item: TTBCustomItem; Viewer: TTBItemViewer; Canvas: TCanvas; ImageRect:
    TRect; ImageOffset: TPoint; StateFlags: Integer) of object;
```

Using OnDrawImage you can customize how the item image is displayed. The StateFlags parameter is the same as StateFlags in OnAdjustFont.

Class TTBXDock

A replacement for TTBDock

Hierarchy

TTBDock
|
TTBXDock

Description

TTBXDock extends TTBDock with support for TBX themes. This also includes support for transparency.

By default, TTBXDock.Color is clNone and the current theme decides how to paint the dock. Setting the Color property to any other value, paints the dock as a uniform rectangle filled with that color. The Color property is ignored when UseParentBackground = True

Properties

published

UseParentBackground

property UseParentBackground: Boolean;

Indicates that the dock should be "transparent". This feature might not work well with all the types of the parent controls. When the dock is transparent, its Color property is ignored.

Class TTBXItem

A theme-aware replacement for TTBItem

Hierarchy

TTBCustomItem
|
TTBXCustomItem
|
TTBXItem

Description

In popup menus, TTBXItem corresponds to a menu item. In toolbars, TTBXItem corresponds to a toolbar button. TTBXItem is a replacement for TTBItem in Toolbar2000.

TTBXItem implements the generic behavior introduced in TTBXCustomItem, but does not introduce any new behavior.

Class TTBXPopupMenu

A TBX version of TPopupMenu

Hierarchy

TTBPopupMenu
|
TTBXPopupMenu

Description

TTBXPopupMenu is as a replacement for TTBPopupMenu.

See Toolbar2000 documentation for more information.

Properties

ToolBoxPopup

property ToolBoxPopup: Boolean;

When the popup menu contains a tool box, its behavior and appearance is slightly different from the standard popup menu. Set ToolBoxPopup to *True* when the popup menu contains a tool box.

Class TTBXSeparatorItem

A toolbar item for menu and toolbar separators

Hierarchy

TTBSeparatorItem
|
TTBXSeparatorItem

Description

TTBXSeparatorItem is a TBX counterpart for TTBSeparatorItem. In addition, it allows changing the size of the separator.

Properties

published

Size

property Size: Integer;

Controls the size of the separator. Set value to -1 for default size which depends on the current theme settings.

Methods

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXSeparatorItem. After calling the inherited constructor, Create initializes the Size property to -1;

Class TTBXSubmenuItem

A toolbar item for top level menus and submenus

Hierarchy

TTBCustomItem
  |
TTBXCustomItem
  |
TTBXSubmenuItem

Description

TTBXSubmenuItem is a TBX counterpart for TTBSubmenuItem.

Properties

published

DropdownCombo

property DropdownCombo: Boolean;

This property acts the same way as the DropDownCombo in TTBSubmenuItem.

Methods

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXSubmenuItem.

Class TTBXToolbar

A toolbar which supports TBX themes

Hierarchy

TTBCustomToolbar
|
TTBXToolbar

Description

TTBXToolbar should be used in your applications instead of TTBToolbar. TTBXToolbar implements all behavior of TTBCustomToolbar; additionally, it provides support for TBX themes.

By default, the color of the toolbar is set to clNone, and the current theme makes a decision on how to paint the toolbar. By setting the color to any other value, you can customize the color of the toolbar. The transparency for toolbars docked into TTBXDock, or any dock with background is only supported when the color is clNone.

Properties

EffectiveColor

property EffectiveColor: TColor;

When Color=clNone, the actual color of the toolbar is calculated by the current theme, for embedded toolbars the actual color is the color of the parent control. The resulting effective color in these cases can be calculated using the EffectiveColor property. For the rest of Color values, EffectiveColor contains the same value as the Color property.

See also: UpdateChildColors

published

ItemTransparency

property ItemTransparency: TTBXItemTransparency;
type TTBXItemTransparency = (itAuto, itEnable, itDisable);

Controls whether the buttons and other items on the toolbar should be transparent or outlined in the normal state. Normally, this property should be of interest for embedded toolbars.

published

SnapDistance

property SnapDistance: Integer;

Set snap distance to a non-zero value to enable toolbar snapping to screen edges.

Methods

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXToolbar. After calling the inherited constructor, Create initializes the Color property to clNone;

Embedded

function Embedded: Boolean;

Returns *True* for non-floating toolbars which are not inserted into the dock. This function is used internally to differentiate the appearance of such embedded toolbars from standard toolbars.

UpdateChildColors

procedure UpdateChildColors;

Updates the colors of all child controls with ParentColor = *True*.

Internally, when a child control, sets ParentColor to *True* the parent color is obtained from the Parent.Color property. Since TBX toolbars by default have Color = clNone, the child will incorrectly obtain clNone instead of using the EffectiveColor. Most of the standard VCL controls display clNone as black which will result in incorrect painting. Unfortunately, there is no way of fixing this issue without modifying Delphi sources. However, with UpdateChildColors, you can update colors of the children after setting ParentColor to *True*.

See also: EffectiveColor

Class TTBXToolWindow

A TBX replacement for TTBToolWindow

Hierarchy

TTBToolWindow
  |
TTBXToolWindow

Description

TTBXPopupMenu is as a replacement for TTBPopupMenu. See Toolbar2000 documentation for more information.

Properties

EffectiveColor

property EffectiveColor: TColor;

When the Color property is set to clNone, the actual color of the window is calculated by the current theme. The effective color of the window in this case can be obtained using the EffectiveColor property. For the rest of Color values, EffectiveColor contains the same value as the Color property.

See also: UpdateChildColors

published

SnapDistance

property SnapDistance: Integer;

Set snap distance to a non-zero value to enable tool window snapping to screen edges.

Methods

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXToolWindow. After calling the inherited constructor, Create initializes the Color property to clNone;

UpdateChildColors

procedure UpdateChildColors;

Updates the colors of all child controls with ParentColor = *True*.

Internally, when a child control, sets ParentColor to *True* the parent color is obtained from the Parent.Color property. Since TBX toolbars by default have Color = clNone, the child will incorrectly obtain clNone instead of using the EffectiveColor. Most of the standard VCL controls display clNone as black which will result in incorrect painting. Unfortunately, there is no way of fixing this issue without modifying Delphi sources. However, with UpdateChildColors, you can update colors of the children after setting ParentColor to *True*.

See also: EffectiveColor

Class TTBXVisibilityToggleItem

An item which toggles visibility of other controls

Hierarchy

TTBCustomItem
  |
TTBXCustomItem
  |
TTBXVisibilityToggleItem

Description

TTBXVisibilityToggleItem should be used instead of TTBVisibilityToggleItem. See the TTBVisibilityToggleItem description in Toolbar2000 documentation for more information.

Properties

Control

published

property Control: TControl;

Set this property to the control whose visibility is affected by the toggle item.

Routines

AddThemeNotification

procedure AddThemeNotification(AObject: TObject);

The objects registered with AddThemeNotification will receive TBM_THEMECHANGE messages when TBX theme changes.

See also: RemoveThemeNotification

GetEffectiveColor

function GetEffectiveColor(C: TControl): TColor;

Returns the effective color of the control. For most controls this function returns the value of Color property, however, some TBX controls may return different effective color if their Color property is set to clNone.

RemoveThemeNotification

procedure RemoveThemeNotification(AObject: TObject);

Removes theme change notification previously registered with AddThemeNotification.

TBXCurrentTheme

function TBXCurrentTheme: **string**;

Returns the name of the active theme.

TBXSetTheme

procedure TBXSetTheme(**const** AThemeName: **string**);

Changes the current theme. If AThemeName is unknown, the 'Default' theme will be used instead.

Types

TTBXItemTransparency

TTBXItemTransparency = (itAuto, itEnable, itDisable);

Specifies appearance of toolbar items

TTriState

TTriState = (tsDefault, tsTrue, tsFalse);

A binary flag with a third state.

tsDefault
Use default or inherited value

tsTrue
Set binary value to True

tsFalse
Set binary value to False

Variables

CurrentTheme

CurrentTheme: TTBXTheme;

Contains the reference to active TBX theme.

Unit TBXDkPanels.pas

TTBXDkPanels implements dockable panels and several simple controls which support TBX themes.

Quick reference

classes:

- TTBXAlignmentPanel – A control which provides alignment margins for its children
- TTBXButton – A push button control
- TTBXCheckBox – A check box control
- TTBXControlMargins – A set of margins used by some controls for alignment purposes
- TTBXCustomButton
- TTBXCustomCheckBox – An internal implementation of TTBXCheckBox
- TTBXCustomLabel – An internal implementation of TTBXLabel
- TTBXCustomLink – An internal implementation of TTBXLink
- TTBXCustomPageScroller – A scrolling control
- TTBXCustomRadioButton – An internal implementation of TTBXRadioButton
- TTBXDockablePanel – An implementation of a resizable dockable panel
- TTBXLabel – A simple label component

- TTBXLink – A hot-link control
- TTBXMultiDock – A special dock for dockable panels
- TTBXPageScroller – A scrolling control
- TTBXPanelObject – A common ancestor for controls to be embedded into dockable panels
- TTBXRadioButton – A radio button control
- TTBXTextObject – A common ancestor for objects with a caption

types:

- TTBXResizingStage

Class TTBXAlignmentPanel

A control which provides alignment margins for its children

Hierarchy

TCustomControl
|
TTBXPanelObject
|
TTBXAlignmentPanel

Description

TTBXAlignment panel provides a set of margins to adjust alignment of child controls.

One of the places where such panels are particularly useable is arrangement of child controls within the TTBXPageScroller container.

Properties

Margins

published

property Margins: TTBXControlMargins;

Defines the child alignment rectangle relative to the panel's boundaries.

See also: Class TTBXControlMargins

Class TTBXButton

A push button control

Hierarchy

TCustomControl
|
TTBXPanelObject
|
TTBXTextObject
|
TTBXCustomButton
|
TTBXButton

Description

TTBXButton implements a push button control. It supports caption wrapping, margins, and other common features of panel objects features. In addition, TTBXButton supports TBX themes.

Note that the Color property controls the *Color* of the background, as described in TTBXPanelObject. In TTBXButton, you can see the background only when Margins are non zero. The color of the button itself is controlled by the theme. TBX does not provide means for changing this color.

Class TTBXCheckBox

A check box control

Hierarchy

TCustomControl
|
TTBXPanelObject
|
TTBXTextObject
|
TTBXCustomCheckBox
|
TTBXCheckBox

Description

TTBXCheckBox is an alternative to the standard check box control. Since it is a panel object, it supports caption wrapping, margins, and other new features. In addition, TTBXCheckBox supports TBX themes.

Class TTBXControlMargins

A set of margins used by some controls for alignment purposes

Hierarchy

TPersistent
|
TTBXControlMargins

Description

TTBXControlMargins is a set of four distances. In some controls, these distances are used to define indents from their boundaries to the internal content.

See also: Class TTBXAlignmentPanel, Class TTBXTextObject

Properties

Bottom

published

property Bottom: Integer;

Bottom margin.

Left

published

property Left: Integer;

Left margin.

Right

published

property Right: Integer;

Right margin.

Top

published

property Top: Integer;

Top margin.

Class TTBXCustomButton

Hierarchy

TCustomControl
|
TTBXPanelObject
|
TTBXTextObject
|
TTBXCustomButton

Description

TTBXCustomButton is an internal base class for TTBXButton. See TTBXButton for more information.

Properties

Alignment

protected

property Alignment: TAlignment;

Defines caption alignment within the button.

AllowAllUnchecked

protected

property AllowAllUnchecked: Boolean;

Indicates whether all the buttons in a given group can be unchecked at the same time.

BorderSize

protected

property BorderSize: Integer;

The size of the border between the button boundary and its contents.

ButtonStyle

protected

property ButtonStyle: TButtonStyle;
type TButtonStyle = (bsNormal, bsFlat);

Controls the button appearance.

Checked

protected

property Checked: Boolean;

Indicates that the button within its group checked.

Note: This property works only when GroupIndex is *not* zero.

DropdownCombo

protected

property DropdownCombo: Boolean;

Controls whether the button with a DropdownMenu appears as a combo button or a simple button with an arrow.

Note: This property works only when DropdownMenu is assigned.

DropdownMenu

protected

property DropdownMenu: TPopupMenu;

Allows to associate a popup menu with the button. It is recommended to use TTBXPopupMenu for better appearance and more proper alignment of the drop down menu.

protected

GlyphSpacing

property GlyphSpacing: Integer;

The distance between the caption and the glyph inside the button.

protected

GroupIndex

property GroupIndex: Integer;

Allows grouping several buttons together. This property works similar to GroupIndex in the standard TSpeedButton.

protected

ImageIndex

property ImageIndex: TImageIndex;

Use ImageIndex to select an image to be displayed inside the button.

protected

Images

property Images: TCustomImageList;

To include an image with the button, set Images to an image list component and choose the corresponding ImageIndex.

protected

Layout

property Layout: TButtonLayout;
type TButtonLayout = (blGlyphLeft, blGlyphTop, blGlyphRight, blGlyphBottom);

Controls relative layout of the image and the caption within the button control.

protected

RepeatDelay

property RepeatDelay: Integer;

See Repeating for more information

protected

Repeating

property Repeating: Boolean;

When Repeating is True, pushing the button with the mouse will repeatedly generate the OnClick event until the mouse is released.

The first OnClick event is generated immediately when mouse button is pushed. Then, in RepeatDelay milliseconds, the second OnClick event is fired, after which OnClick will be called again and again every RepeatInterval milliseconds until the mouse button is released.

protected

RepeatInterval

property RepeatInterval: Integer;

See Repeating for more information

Events

OnDropDown

protected

property OnDropDown: TDropDownEvent;
type TDropDownEvent = **procedure**(Sender: TTBXCustomButton; **var** AllowDropDown: Boolean) **of object**;

OnDropDown is fired immediately before the drop down menu pops up.

Class TTBXCustomCheckBox

An internal implementation of TTBXCheckBox

Hierarchy

TCustomControl
  |
TTBXPanelObject
  |
TTBXTextObject
  |
TTBXCustomCheckBox

Description

TTBXCustomCheckBox is an internal base class for TTBXCheckBox. See TTBXCheckBox for more information.

Properties

AllowGrayed

protected

property AllowGrayed: Boolean;

Determines whether the check box can be in a "grayed" state.

Checked

protected

property Checked: Boolean;

Specifies whether the check box is checked.

This property is somewhat redundant since the checked state can be requested using the State property.

See also: State

protected

State

property State: TCheckBoxState;

Indicates whether the check box is checked, unchecked, or grayed.

Events

protected

OnChange

property OnChange: TNotifyEvent;

OnChange is triggered each time the State property is altered.

Class TTBXCustomLabel

An internal implementation of TTBXLabel

Hierarchy

TCustomControl
|
TTBXPanelObject
|
TTBXTextObject
|
TTBXCustomLabel

Description

TTBXCustomLabel is an internal base class for TTBXLabel. See TTBXLabel for more information.

Properties

protected

FocusControl

property FocusControl: TWinControl;

This property acts the same way as the FocusControl property in the standard TLabel control.

protected

Underline

property Underline: Boolean;

Indicates wether the the label is underlined.

See also: UnderlineColor

protected

UnderlineColor

property UnderlineColor: TColor;

Indicates the color of the underline.

See also: Underline

Methods

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXCustomLabel. After calling the inherited constructor, Create initializes UnderlineColor to clBtnShadow.

Class TTBXCustomLink

An internal implementation of TTBXLink

Hierarchy

TCustomControl
|
TTBXPanelObject
|
TTBXTextObject
|
TTBXCustomLink

Description

TTBXCustomLink is an internal base class for TTBXLink. See TTBXLink for more information.

Properties

protected

ImageIndex

property ImageIndex: TImageIndex;

Use ImageIndex to select the image.

See also: Images

protected

Images

property Images: TCustomImageList;

To include an image with the link, set Images to an image list component and choose the corresponding ImageIndex.

Class TTBXCustomPageScroller

A scrolling control

Hierarchy

TWinControl
|
TTBXCustomPageScroller

Description

TTBXCustomPageScroller is an internal base implementation for TTBXPageScroller. See TTBXPageScroller for more information.

Properties

AutoRange

property AutoRange: Boolean;

When AutoRange = *True*, the page scroller calculates the range of scrolling automatically.

See also: Range

protected

AutoScroll

property AutoScroll: Boolean;

Controls how the scrolling is activated. If AutoScroll = True (default), the scrolling starts when the mouse cursor is hovered over a scroll button button. When AutoScroll = False, the button needs to be pushed.

protected

ButtonSize

property ButtonSize: Integer;

The size of scrolling buttons. The scrolling buttons are visible only when Range exceeds the size of the control.

protected

Margin

property Margin: Integer;

Specifies the margin between the contained controls and the inside edges of the page scroller.

protected

Orientation

property Orientation: TTBXPageScrollerOrientation;
type TTBXPageScrollerOrientation = (tpsoVertical, tpsoHorizontal);

Defines the direction of scrolling. For page scrollers with Orientation set to tpsoVertical, the scrolling buttons appear at the top and at the bottom edges of the controls. Conversely, for tbsoHorizontal, the buttons are located at the left and right edges and scrolling is performed in horizontal direction.

protected

Position

property Position: Integer;

Specifies the scroll position of the page scroller.

protected

Range

property Range: Integer;

Defines the range of scrolling. Setting range explicitly will also set AutoRange to *False*.

Methods

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXCustomPageScroller.

DisableAutoRange

procedure DisableAutoRange;

Temprorarily disables automatic range recalculation. Use DisableAutoRange when rearranging controls within the page scroller. Use the EnableAutoRange method when rearranging is finished.

EnableAutoRange

procedure EnableAutoRange;

Enables the automatic range recalculation temporarily disabled with DisableAutoRange.

ScrollToCenter

procedure ScrollToCenter(ARect: TRect);
procedure ScrollToCenter(AControl: TControl);

Use ScrollToCenter to adjust scrolling position and bring a specified rectangle (in client coordinates) or the child control to the center of the page scroller.

Class TTBXCustomRadioButton

An internal implementation of TTBXRadioButton

Hierarchy

TCustomControl
|
TTBXPanelObject
|
TTBXTextObject
|
TTBXCustomRadioButton

Properties

Checked

protected

property Checked: Boolean;

Determines whether the radio button is selected. Setting Checked to *True* will automatically unselect the rest of the radio buttons within the same windowed control container provided they have the same GroupIndex.

GroupIndex

protected

property GroupIndex: Integer;

Use GroupIndex to separate radio buttons contained within the same windowed control container into several logical groups.

See also: Checked

Events

OnChange

protected

property OnChange: TNotifyEvent;

OnChange is fired each time when the Checked state of the radio button is changed.

Class TTBXDockablePanel

An implementation of a resizable dockable panel

Hierarchy

TTBCustomDockableWindow
  |
TTBXDockablePanel

Description

TTBXDockablePanel implements a resizable panel similar to task panes in Microsoft Office XP products.

Properties

BorderSize

published

property BorderSize: Integer;

Specifies an indent from the edge of the client area for alignment of child controls.

published

CaptionRotation

property CaptionRotation: TDPCaptionRotation;
type TDPCaptionRotation = (dpcrAuto, dpcrAlwaysHorz, dpcrAlwaysVert);

Indicates the alignment of the caption for docked panels.

published

DockedHeight

property DockedHeight: Integer;

Controls the height of client area when the panel is docked into the horizontal dock.

published

DockedWidth

property DockedWidth: Integer;

Controls the width of client area when the panel is docked into the vertical dock.

EffectiveColor

property EffectiveColor: TColor;

When Color=clNone, the actual color of the dockable panel is calculated by the current theme. The resulting effective color of the panel in these cases can be obtained using the EffectiveColor property. For the rest of Color values, EffectiveColor contains the same value as the Color property.

published

FloatingHeight

property FloatingHeight: Integer;

Controls the height of the client area when the panel is in the floating mode. If FloatingHeight is set to zero while the panel is docked, the floating height will be copied from the docked height the next time it is dragged off the dock.

published

FloatingWidth

property FloatingWidth: Integer;

Controls the width of client area when the panel is in the floating mode. If FloatingWidth is set to zero while the panel is docked, the floating width will be copied from the docked width the next time it is dragged off the dock.

published

MaxClientHeight

property MaxClientHeight: Integer;

Maximum height of client area of the panel. MaxClientHeight is effective only when it is larger than MinClientHeight.

See also: MinClientHeight, MaxClientWidth

published

MaxClientWidth

property MaxClientWidth: Integer;

Maximum width of client area of the panel. MaxClientWidth is effective only when it is larger than MinClientWidth.

See also: MinClientWidth, MaxClientHeight

published

MinClientHeight

property MinClientHeight: Integer;

Minimum height of the client area of the panel.

See also: MaxClientHeight, MinClientWidth

published

MinClientWidth

property MinClientWidth: Integer;

Minimum width of the client area of the panel.

See also: MaxClientWidth, MinClientHeight

published

ShowCaptionWhenDocked

property ShowCaptionWhenDocked: Boolean;

Controls the caption visibility when the panel is docked.

published

SmoothDockedResize

property SmoothDockedResize: Boolean;

Controls resizing behavior when the panel is docked.

published

SnapDistance

property SnapDistance: Integer;

Set snap distance to a non-zero value to enable toolbar snapping to screen edges.

published

SplitHeight

property SplitHeight: Integer;

Determines the client area height of the panel when it is docked into a vertical TTBXMultiDock.

Then the dock is resized (or panels are inserted/removed), the sizes of all visible panels are redistributed to cover the whole dock area. The dock first tries to use the SplitHeight size for all panels and then the remaining gap is proportionally distributed amongst the panels.

When user resizes panels, the dock tries to keeps positions and sizes for the rest of the docked panels and SplitHeight is simultaneously updated for *all* the panels in the dock to match their immediate on-screen location.

See also: SplitWidth

published

SplitWidth

property SplitWidth: Integer;

Determines the client area width of the panel when it is docked into a horizontal TTBXMultiDock.

See also: SplitHeight

published

SupportedDocks

property SupportedDocks: TDockKinds;
type TDockKinds = **set of** (dkStandardDock, dkMultiDock);

Controls the types of docks into which the panel can be docked.

When using multi-docks, it is highly recommended to exclude dkStandardDock from SupportedDocks to prevent the panel from docking into standard TB2K docks.

Methods

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXDockablePanel.

After performing necessary initialization, Create sets the following properties:

- MinClientWidth to 32
- MinClientHeight to 32
- DockedWidth to 128
- DockedHeight to 128
- ShowCaptionWhenDocked to *True*
- SmoothDockedResize to *True*

UpdateChildColors

procedure UpdateChildColors;

Updates the colors of all child controls with ParentColor = *True*.

Internally, when a child control, sets ParentColor to *True* the parent color is obtained from the Parent.Color property. Since TBX toolbars by default have Color = clNone, the child will incorrectly obtain clNone instead of using the EffectiveColor. Most of the standard VCL controls display clNone as black which will result in incorrect painting. Unfortunately, there is no way of fixing this issue without modifying Delphi sources. However, with UpdateChildColors, you can update colors of the children after setting ParentColor to *True*.

Events

published

OnDockedResizing

property OnDockedResizing: TTBXDockedResizing;
type TTBXDockedResizing = **procedure**(Sender: TObject; Vertical: Boolean; **var** NewSize: Integer; Stage: TTBXResizingStage; **var** AllowResize: Boolean) **of object**;

This event is fired during the panel resizing when it is docked. It allows imposing additional constraints on the window dimensions.

When the user stretches a panel docked at the horizontal dock, Vertical is *True* and the NewSize parameter indicates the height of the panel. When resizing a panel docked at a vertical dock, NewSize indicates the width of the panel.

See also: TTBXResizingStage

Class TTBXLabel

A simple label component

Hierarchy

TCustomControl
|
TTBXPanelObject
|
TTBXTextObject
|
TTBXCustomLabel
|
TTBXLabel

Description

TTBXLabel is a panel object with a single purpose—displaying a text label. The label may optionally be underlined.

TTBXLabel implements the generic behavior introduced in TTBXCustomLabel, but does not introduce any new behavior.

Class TTBXLink

A hot-link control

Hierarchy

TCustomControl
  |
TTBXPanelObject
  |
TTBXTextObject
  |
TTBXCustomLink
  |
TTBXLink

Description

TTBXLink is similar to TTBXLabel with the OnClick event published. In addition, the link can include an image and the caption is underlined when the mouse cursor is positioned over the link. As a descendant of TTBXTextObject, it supports caption wrapping, margins, and other features.

Class TTBXMultiDock

A special dock for dockable panels

Hierarchy

TTBDock
  |
TTBXMultiDock

Methods

ResizeVisiblePanels

```
procedure ResizeVisiblePanels(NewSize: Integer);
```

Resizes all the panels that are currently visible in the dock.

Class TTBXPageScroller

A scrolling control

Hierarchy

TWinControl
  |
TTBXCustomPageScroller
  |
TTBXPageScroller

Description

TTBXPageScroller is an alternative to the standard TPageScroller control. This control has been designed to be placed into TTBXDockablePanel. It can be placed onto many other controls as well.

TTBXPageScroller serves the same purpose as standard TPageScroller and TScrollBox—allowing access to child controls when the client area is larger when the window of the page scroller itself. However, there are some substantial differences in the way it is handled. For example, TTBXPage-Scroller allows having child controls with Align set to alClient or alBottom.

Currently, the proper alignment of child controls can be obtained if their Align property is set to alTop, alClient, or alBottom for vertical page scrollers, and alLeft, alClient, or alRight for horizontal page scrollers. The further restriction is that only one control amongst children can have alClient alignment.

One of the ways to arrange child controls within the page scroller is to place them in TTBXAlign-mentPanel, or other container with a proper alignment.

Class TTBXPanelObject

A common ancestor for controls to be embedded into dockable panels

Hierarchy

TCustomControl
  |
TTBXPanelObject

Description

TTBXPanelObject is a common ancestor for some of the additional controls designed to be embed-ded into dockable panels and page scrollers.

You can use it as child of other controls as well.

By default, when the *Color* property of a panel object is *clNone*, it simulates transparency by rendering parent's background. This is similar to the ParentBackground property introduced in Delphi 7, but works independent from XP themes.

Technically this transparency simulation is done by redirecting WM_ERASEBKGND event to the parent control, if the parent control does some painting in WM_PAINT event handler, these details will not be seen in the panel object's background.

Properties

SmartFocus

protected

property SmartFocus: Boolean;

Using the SmartFocus property, alters the focusing behavior of the panel object. When SmartFocus is *false*, the control has the standard focusing behavior. For controls in dockable panels, tool windows, and toolbars, such standard behavior is often not the best choice since it will steal the focus from the main area of your application. Due to this reason, the SmartFocus property was introduced.

Set SmartFocus to *true* to prevent unwanted focusing of panel objects. While the focus is within the main area of the application, clicking the control with SmartFocus retains the focus within the main area. To simplify the keyboard navigation, such a control will still get focused if the focus already belongs to one of its siblings.

SpaceAsClick

protected

property SpaceAsClick: Boolean;

When the panel object is focused, SpaceAsClick indicates that hitting the space key has the same result as clicking the control with the mouse.

Methods

MakeVisible

procedure MakeVisible;

If the panel object is a child of TTBXCustomPageScroller or TScrollingWinControl, calling MakeVisible will adjust scroll position of the parent to bring it into the visible area.

Events

OnMouseEnter

protected

property OnMouseEnter: TNotifyEvent;

OnMouseEnter is fired each time the mouse cursor enters the boundaries of the object.

See also: OnMouseLeave

OnMouseLeave

protected

property OnMouseLeave: TNotifyEvent;

OnMouseLeave is fired each time the mouse cursor leaves the boundaries of the object.

See also: OnMouseEnter

Class TTBXRadioButton

A radio button control

Hierarchy

TCustomControl
|
TTBXPanelObject
|
TTBXTextObject
|
TTBXCustomRadioButton
|
TTBXRadioButton

Description

TTBXRadioButton is an alternative to the standard radio button control. Since it is a panel object, it supports caption wrapping, margins, and other new features. In addition, TTBXRadioButton supports TBX themes.

Class TTBXTextObject

A common ancestor for objects with a caption

Hierarchy

TCustomControl
|
TTBXPanelObject
|
TTBXTextObject

Description

TTBXTextObject is a base ancestor for other panel objects which defines some common features for displaying a caption.

Properties

Alignment

protected

property Alignment: TLeftRight;

Defines the alignment of the caption.

protected

Margins

property Margins: TTBXControlMargins;

Use margins to control the padding between the content and the boundaries of the object.

protected

ShowAccelChar

property ShowAccelChar: Boolean;

Determines how an ampersand in the caption is displayed.

When ShowAccelChar is *False*, any character preceded by an ampersand (&) appears underlined. To display an ampersand use two ampersands (&&) to stand for the single ampersand that is displayed or set ShowAccelChar to *True*.

protected

Wrapping

property Wrapping: TTextWrapping;
type TTextWrapping = (twNone, twEndEllipsis, twPathEllipsis, twWrap);

Wrapping controls how the caption is displayed when the text does not fit within the allowed boundaries.

Types

TTBXResizingStage

TTBXResizingStage = (rsBeginResizing, rsResizing, rsEndResizing);

Indicates the stage in the process of panel resizing.

See also: OnDockedResizing

Unit TBXExtItems.pas

TTBXExtItems contains various additional controls with extended functionality.

Quick reference

classes:

- TTBXColorItem – A toolbar item which displays color information.
- TTBXComboBoxItem – A combo box item
- TTBXCustomDropDownItem – An internal implementation of TTBXDropDownItem
- TTBXCustomSpinEditItem – An internal implementation of TTBXSpinEditItem
- TTBXDropDownItem – A combination of the edit item with the drop down button
- TTBXEditItem – A TBX analogue to TTBEditItem.
- TTBXLabelItem – A toolbar item which displays a line of text
- TTBXMRUList – A container for MRU list entries
- TTBXMRUListItem – An implementation of MRU list item
- TTBXSpinEditItem – An edit item for numeric values

Class TTBXColorItem

A toolbar item which displays color information.

Hierarchy

TTBCustomItem
|
TTBXCustomItem
|
TTBXColorItem

Description

TTBXColorItem is a very simple extension of TTBXCustomItem. Instead of displaying an icon, it shows a rectangle filled with a particular color.

Properties

published

Color

property Color: TColor;

Specifies a color displayed by the item.

Class TTBXComboBoxItem

A combo box item

Hierarchy

TTBEditItem
|
TTBXEditItem
|
TTBXCustomDropDownItem
|
TTBXComboBoxItem

Description

TTBXComboBoxItem is a combination of TTBXDropDownItem and TTBXStringList.

A few remarks on handling images with TTBXComboBoxItem.

The list entries use the same image list as the combo item. However, images in the list can be set independently from the images in the edit box area. The ShowImage property controls the appearance of the image in the edit area, while ShowListImages controls images in the list area.

The ImageIndex property only defines the image in the edit area when the combo box cannot find the entry in the list. When the text in the edit area coincides with one of the entries in the list, the image is taken from that entry.

See OnAdjustImageIndex property if you need to customize images both in the edit area and in

the list.

Properties

AutoComplete

published

property AutoComplete: Boolean;

Specifies whether the combo box item automatically completes words that the user types by selecting the first item that begins with the currently typed string.

ItemIndex

property ItemIndex: Integer;

Indicates current selection in the list. If ItemIndex = -1, then there is no selection.

MaxListWidth

published

property MaxListWidth: Integer;

Specifies the minimum width of the drop down area.

See also: MinListWidth

MaxVisibleItems

published

property MaxVisibleItems: Integer;

Specifies the maximum number of visible items. The scrollbar appears when the total number of items in the list exceeds MaxVisibleItems.

MinListWidth

published

property MinListWidth: Integer;

Specifies the minimum width of the drop down area.

See also: MaxListWidth

ShowListImages

published

property ShowListImages: Boolean;

Indicates wether the images in the list area are visible. This property corresponds to ShowImages.

Strings

published

property Strings: TStrings;

Provides access to the list of items (strings) in the list portion of the combo box item.

Methods

Create

constructor Create(AOwner: TComponent);

Creates and initializes an instance of TTBXComboBoxItem.

Events

OnAdjustImageIndex

published

property OnAdjustImageIndex: TTBXCAdjustImageIndex;
type TTBXCAdjustImageIndex = **procedure**(Sender: TTBXComboBoxItem; **const** AText: **string**; AIndex: Integer; **var** ImageIndex: Integer) **of object**;

With the OnAdjustImageIndex event you can customize the image in the edit area of the combo box as well as the images for list entries.

When the combo box fires this event for painting the image in the edit area, the AIndex parameter contains -1; for the list entries, AIndex contains the index of the entry index in the list. The AText parameter contains either the text in the edit area or corresponds to the list entry.

See also: Strings

OnClearItem

published

property OnClearItem: TTBXLPaintEvent;
type TTBXLPaintEvent = **procedure**(Sender: TTBXCustomList; ACanvas: TCanvas; ARect: TRect; AIndex, AHoverIndex: Integer; **var** DrawDefault: Boolean) **of object**;

Each item is painted in two stages: first the background is filled with the default list color, or with highlight color if the item is selected; when, the text is painted on top of it. It is possible to customize any of these stages.

Handle the OnClearItem event to draw a customized background of the item. Set the DrawDefault parameter to *False* if you don't want the default background.

OnDrawItem

published

property OnDrawItem: TTBXLPaintEvent;
type TTBXLPaintEvent = **procedure**(Sender: TTBXCustomList; ACanvas: TCanvas; ARect: TRect; AIndex, AHoverIndex: Integer; **var** DrawDefault: Boolean) **of object**;

OnDrawItem is fired after the OnClearItem event. Handle OnDrawItem to customize the second stage of the painting process and set DrawDefault to *False* if default painting is not required.

Tip: Font settings are not restored after this event. Therefore, a simple way to assign a custom font for each item is to leave the DrawDefault = *True*, and change the necessary properties in ACanvas.Font.

Note, that the item width can be different for different fonts and some adjustments might be required in OnMeasureWidth.

published

OnItemClick

```
property OnItemClick: TNotifyEvent;
```

Handle OnItemClick to respond to item clicking in the list area.

published

OnMeasureHeight

```
property OnMeasureHeight: TTBXLMeasureHeight;
type TTBXLMeasureHeight = procedure(Sender: TTBXCustomList; ACanvas: TCanvas; var AHeight: Integer) of
    object;
```

OnMeasureHeight allows changing the item height in the list. All items have the same height, and by default this height is determined by the current font size.

published

OnMeasureWidth

```
property OnMeasureWidth: TTBXLMeasureWidth;
type TTBXLMeasureWidth = procedure(Sender: TTBXCustomList; ACanvas: TCanvas; AIndex: Integer; var AWidth:
    Integer) of object;
```

Handle this property to specify a custom width for each item. This width is used by the component internally to determine the width of the list. Note, that unlike OnMeasureHeight, this event is fired for each item.

Class TTBXCustomDropDownItem

An internal implementation of TTBXDropDownItem

Hierarchy

TTBEditItem
|
TTBXEditItem
|
TTBXCustomDropDownItem

Description

TTBXCustomDropDownItem is an internal base class for TTBXDropDownItem. See TTBXDropDownItem for more information.

Class TTBXCustomSpinEditItem

An internal implementation of TTBXSpinEditItem

Hierarchy

TTBEditItem
|
TTBXEditItem
|
TTBXCustomSpinEditItem

Description

TTBXCustomSpinEditItem is an internal base class for TTBXSpinEditItem. See TTBXSpinEditItem for more information.

Properties

AsInteger

property AsInteger: Integer;

When working with integer and hex values, instead of the Value property, use AsInteger to avoid additional type conversion. When ValueType = evtFloat, reading AsInteger returns the value rounded to the nearest integer.

Decimal

property Decimal: TDecimal;
type TDecimal = 0..10;

Number of digits after the decimal separator.

Increment

property Increment: Extended;

Specifies the amount by which the Value property changes when one of the spin buttons is pressed or when either the up or down arrow key is pressed.

MaxValue

property MaxValue: Extended;

Use MaxValue to limit the largest Value that can be entered.

See also: MinValue

MinValue

property MinValue: Extended;

Use MinValue to limit the smallest Value that can be entered.

See also: MaxValue

Postfix

property Postfix: **string**;

The string of characters which appears after the number. Use Postfix to display units, percentage, etc. for evtInteger and evtFloat values. Postfix is ignored when working with evtHex values.

Postfix may consist of letters and symbols, excluding: #0..#31, *space*, '*', '+', ',', '-', '.', '/', '0'..'9', and '^'.

In addition, for floating point values, it should not include symbols 'e' and 'E' to avoid ambiguities with scientific notation.

See also: ValueType, SpaceBeforePostfix, Prefix

Prefix

property Prefix: **string**;

The string of characters which appears before the number. For example, prefix can be used to display a currency symbols. Prefix is ignored when working with evtHex values.

Restrictions on symbols which may be in Prefix are shown in the Postfix property description.

See also: SpaceAfterPrefix

Snap

property Snap: Boolean;

Snap controls how the number is altered using the spin buttons. When the snapping is activated, the new Value after pressing the button is snapped to the grid with the step equal to Increment.

For example, consider the situation when the current Value is 0.32 and Increment is 0.1. When snapping is disabled, pressing the 'Up' button will change Value to 0.42, but with snapping the new Value will be 0.4. Correspondingly, pressing the 'Down' button in the first case, will change Value to 0.22, and with snapping it will be 0.2.

SpaceAfterPrefix

property SpaceAfterPrefix: Boolean;

When SpaceAfterPrefix = *True*, the spin edit item inserts a single space character to separate Prefix from the number.

SpaceBeforePostfix

property SpaceBeforePostfix: Boolean;

When SpaceBeforePostfix = *True*, the item inserts a single space character to separate the number from the Postfix.

Value

property Value: Extended;

This property conains the numeric value displayed by the control in floaging point format.

When ValueType if set to evtInteger or evtHex, the Value property is automatically rounded to the nearest integer. In this case it may also be convenient to access the numeric value using the AsInteger property.

See also: MinValue, MaxValue

ValueType

property ValueType: TSEValueType;
type TSEValueType = (evtInteger, evtFloat, evtHex);

Specifies the type of the value. This affects both displaying and editing of the number.

See also: Value, AsInteger

Methods

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXCustomSpinEditItem.

After calling the inherited constructor Create initializes the following properties:

- Alignment to taRightJustify;
- Decimal to 2;
- Increment to 1;
- Snap to *True*;
- Text to '0';

Events

OnConvert

protected

```
property OnConvert: TSEConvertEvent;
type TSEConvertEvent = procedure(Sender: TTBXCustomSpinEditItem; const APrefix, APostfix: string; var
    AValue: Extended; var CanConvert: Boolean) of object;
```

Using the OnConvert event it is possible to convert between different units.

The following example performs conversion between centimeters, millimeters, and inches:

```
procedure MyForm.SpinEditOnConvert(Sender: TTBXCustomSpinEditItem; const APrefix, APostfix: String; var
    AValue: Extended; var CanConvert: Boolean);
var
    S: string;
begin
    S := APostfix;

    { use current units if user did not type units explicitly }
    if Length(S) = 0 then S := Sender.Postfix;

    { convert everything to mm }
    if CompareText(S, 'in') = 0 then AValue := AValue * 25.4
    else if CompareText(S, 'cm') = 0 then AValue := AValue * 10;

    { convert mm to current units }
    if CompareText(Sender.Postfix, 'in') = 0 then AValue := AValue / 25.4
    else if CompareText(Sender.Postfix, 'cm') = 0 then AValue := AValue * 0.1;
end;
```

OnStep

protected

```
property OnStep: TSEStepEvent;
type TSEStepEvent = procedure(Sender: TTBXCustomSpinEditItem; Step: Integer; const OldValue: Extended; var
    NewValue: Extended) of object;
```

Use OnStep to change the default behavior of the spin buttons. The Step parameter is +1 to the 'Up' button and -1 for the 'Down' button.

OnTextToValue

protected

```
property OnTextToValue: TSETextToValueEvent;
type TSETextToValueEvent = procedure(Sender: TTBXCustomSpinEditItem; const AText: string; out AValue:
    Extended; var CanConvert: Boolean) of object;
```

Allows to interpret an input string as a number.

In you find in the OnTextToValue handler, that AText is meaningful, set AValue to the corresponding value and change the CanConvert parameter to *True*. If event handler does not change the CanConvert, the spin edit proceeds to parsing the string as a number or expression.

For example:

```
procedure MyForm.MySpinEditTextToValue(Sender: TTBXCustomSpinEditItem; const AText: string; out AValue:
    Extended; var CanConvert: Boolean);
begin
    if CompareText(AText, 'pi') = 0 then
    begin
        AValue := 3.14;
        CanConvert := True;
    end;
end;
```

See also: OnValueToText

protected

OnValueChange

```
property OnValueChange: TSEChangeEvent;
type TSEChangeEvent = procedure(Sender: TTBXCustomSpinEditItem; const AValue: Extended) of object;
```

OnChange is fired every time the Value changes.

protected

OnValueToText

```
property OnValueToText: TSEValueToTextEvent;
type TSEValueToTextEvent = procedure(Sender: TTBXCustomSpinEditItem; const AValue: Extended; var Text:
    string) of object;
```

Use OnValueToText to customize the textual representation of the Value property by changing the Text parameter.

See also: OnTextToValue

Class TTBXDropDownItem

A combination of the edit item with the drop down button

Hierarchy

TTBEditItem
|
TTBXEditItem
|
TTBXCustomDropDownItem
|
TTBXDropDownItem

Description

A combination of TTBXEditItem with the combo button which works similarly to TTBXSubmenuItem.

The contents of the drop down menu is determined by the Items property, or can be linked from other tb2k or TBX items; the same way as submenu items work.

TTBXDropDownItem implements the generic behavior introduced in TTBXCustomDropDownItem, but does not introduce any new behavior.

Class TTBXEditItem

A TBX analogue to TTBEditItem.

Hierarchy

TTBEditItem
|
TTBXEditItem

Description

TTBXEditItem is a single-line text input control. Most of the item's behavior is inherited from TTBEditItem. Besides TBX theme support, TTBXEditItem allows changing the alignment of the text, using password characters, displaying an image next to the text, etc.

Properties

Alignment

published

property Alignment: TAlignment;

Use Alignment to specify whether values should be left-justified, right-justified, or centered in the column.

FontSettings

published

property FontSettings: TFontSettings;

Controls the font used by the edit item. This property works similar to FontSettings

PasswordChar

published

property PasswordChar: Char;

This property is analogous to PasswordChar in the standard TEdit control.

ReadOnly

published

property ReadOnly: Boolean;

To restrict the item to display only, set the ReadOnly property to *True*. See description for TEdit.ReadOnly in Delphi Reference.

ShowImage

published

property ShowImage: Boolean;

Specifies wether an image should be shown next to the text. The image is obtained from Images and ImageIndex properties the same way as it is done in TTBCustomItem.

Methods

StartEditing

function StartEditing(AView: TTBView): Boolean;

Use StartEditing to put the item into the editing mode. The function does not exit until the editing is complete. It returns *True* if the Text property has been changed while the item was in the editing mode.

Events

OnChange

published

property OnChange: TTBXEditChange;
type TTBXEditChange = **procedure**(Sender: TObject; **const** Text: **string**) **of object**;

Add the OnChange handler to respond to changes in the text while the user is typing or changing the contents of the edit item at runtime.

Class TTBXLabelItem

A toolbar item which displays a line of text

Hierarchy

TTBCustomItem
|
TTBXLabelItem

Description

TTBXLabelItem is a simple line of text which can be incorporated anywhere on a toolbar or in popup menu.

Properties

Caption

published

property Caption: TCaption;

Determines the actual text displayed in the label.

FontSettings

published

property FontSettings: TFontSettings;

This property works similarly to FontSettings

Margin

published

property Margin: Integer;

Specifies the distance (in pixels) to other items on a toolbar.

Orientation

published

property Orientation: TTBXLabelOrientation;
type TTBXLabelOrientation = (tbxoAuto, tbxoHorizontal, tbxoVertical);

Controls orientation of the text.

Methods

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXLabelItem.

UpdateCaption

procedure UpdateCaption(**const** Value: TCaption);

Sets the Caption property without invalidating an item size. Using this method helps to avoid flickering when updating the item within the popup menu while the menu is visible.

Events

published

OnAdjustFont

property OnAdjustFont: TAdjustFontEvent;
type TAdjustFontEvent = **procedure**(Item: TTBCustomItem; Viewer: TTBItemViewer; Font: TFont; StateFlags: Integer) **of object**;

This event works similarly to OnAdjustFont

Class TTBXMRUList

A container for MRU list entries

Hierarchy

TTBMRUList
|
TTBXMRUList

Description

TTBXMRUList is a replacement for the TTBMRUList component. See Toolbar2000 documentation for more information on TTBMRUList.

Properties

published

KeyShift

property KeyShift: Integer;

Using KeyShift you can change the numbering of items in the MRU list. This is useful, for example, when two or more MRU list items are located in the same submenu. By changing the KeyShift value you can make items in the second list to start their key numbering from other than “1” symbol.

Class TTBXMRUListItem

An implementation of MRU list item

Hierarchy

TTBCustomItem
|
TTBXCustomItem
|
TTBXMRUListItem

Description

Use TTBXMRUListItem in as a placeholder for the list of most recently used items. TTBXMRUListItem should be used in conjunction with the TTBXMRUList component, which will fill the list with corresponding items at runtime.

When upgrading from standard Toolbar2000, to provide correct support for TBX themes, TTBXMRUListItem should be used instead of TTBMRUListItem and corresponding TTBMRUList components should be replaced with TTBXMRUList.

Properties

MRUList

published

property MRUList: TTBMRUList;

Point MRUList to a valid TTBXMRUList object.

Methods

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXMRUListItem.

Class TTBXSpinEditItem

An edit item for numeric values

Hierarchy

TTBEditItem
|
TTBXEditItem
|
TTBXCustomSpinEditItem
|
TTBXSpinEditItem

Description

TTBXSpinEditItem is a special edit item for entering numeric values. It allows displaying and editing integer and floating point numbers and provides a set of spin buttons. In addition, it works as a primitive calculator and facilitates displaying, editing and converting values with different units (e.g. converting between inches and centimeters). Finally, TTBXSpinEditItem allows displaying some particular values as text and interpret text input as a value.

The calculator function works as very simple, one operator calculator. For example, when user enters '5 + 2', spin edit recognizes it as 7. At present, only '+', '-', '*', and '/' operators can be used and only one operator can be used. Each value in the expression can have its own prefix and postfix. Provided the correct OnConvert handler is written, expressions like '1.2 mm + 3 in' also become valid. Operators and units do not work with hex values.

Using OnTextToValue and OnValueToText, one can define literal constants. Note however, that constants do not work with operators.

TTBXSpinEditItem implements the generic behavior introduced in TTBXCustomSpinEditItem, but does not introduce any new behavior.

Unit TBXLists.pas

Quick reference

classes:

- TTBXCustomList – A common ancestor for toolbar items displaying string lists
- TTBXStringList – A toolbar item which displays a collection of strings in a scrollable list
- TTBXUndoList – A toolbar list item for displaying list of undo actions etc.

Class TTBXCustomList

A common ancestor for toolbar items displaying string lists

Hierarchy

TTBCustomItem
|
TTBXCustomItem
|
TTBXCustomList

Description

TTBXCustomList is an ancestor for scrollable list controls embedded in popup menus. This is an internal abstract class, which only handles list-specific properties without carrying any particular list elements.

TTBXCustomList supports images on the list entries.

The images are taken from the Images property and similarly to TTBCustomItem, TTBXCustomList obtains images from parent when it's own Images property is *nil*.

By default, the image index for each entry in the list is the same as item index. This can be changed by handling the OnAdjustImageIndex property.

See also: Class TTBXStringList, Class TTBXUndoList

Properties

ItemIndex

property ItemIndex: Integer;

Indicates the current selected item in the list. If ItemIndex is -1, there is no selection.

MaxVisibleItems

property MaxVisibleItems: Integer;

Specifies the maximum number of visible items. The scrollbar appears when the total number of items in the list exceeds MaxVisibleItems.

MaxWidth

property MaxWidth: Integer;

Normally, the list will set its horizontal size automatically, depending on the width of the largest item in it. Using MaxWidth and MinWidth, it is possible to set the limits on this behavior.

MinWidth

property MinWidth: Integer;

See MaxWidth description.

ShowImages

property ShowImages: Boolean;

When ShowImages is set to *True*, the list displays its items with images.

Methods

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXCustomList.

MakeVisible

procedure MakeVisible(AIndex: Integer);

When an item referenced by AIndex is scrolled outside the visible area, calling MakeVisible will adjust the scroll bar position to make the item visible.

Events

OnAdjustImageIndex

```
property OnAdjustImageIndex: TTBXLAdjustImageIndex;
type TTBXLAdjustImageIndex = procedure(Sender: TTBXCustomList; AItemIndex: Integer; var ImageIndex:
    Integer) of object;
```

Handle OnAdjustImageIndex to customize the images appearing next to entries in the list.

See also: ShowImages

OnChange

```
property OnChange: TNotifyEvent;
```

OnChange is fired each time the new item is selected.

OnClearItem

```
property OnClearItem: TTBXLPaintEvent;
type TTBXLPaintEvent = procedure(Sender: TTBXCustomList; ACanvas: TCanvas; ARect: TRect; AIndex,
    AHoverIndex: Integer; var DrawDefault: Boolean) of object;
```

Each item is painted in two stages: first the background is filled with the default list color, or with highlight color if the item is selected; when, the text and image is painted on top of it. It is possible to customize these stages. Handle the OnClearItem event to draw a customized background of the item and OnDrawItem to paint image and text.

Set the DrawDefault parameter to *False* if you don't want the default background.

OnDrawItem

```
property OnDrawItem: TTBXLPaintEvent;
type TTBXLPaintEvent = procedure(Sender: TTBXCustomList; ACanvas: TCanvas; ARect: TRect; AIndex,
    AHoverIndex: Integer; var DrawDefault: Boolean) of object;
```

OnDrawItem is fired after the OnClearItem event. Handle OnDrawItem to customize the second stage of the painting process and set DrawDefault to False if the default painting is not required.

Tip: Font settings are not restored after this event. Therefore, a simple way to assign a custom font for each item is to leave the DrawDefault = *True*, and change the necessary properties in ACanvas.Font. Note, that the item width can be different for different fonts and some adjustments might be required in OnMeasureWidth.

OnMeasureHeight

```
property OnMeasureHeight: TTBXLMeasureHeight;
type TTBXLMeasureHeight = procedure(Sender: TTBXCustomList; ACanvas: TCanvas; var AHeight: Integer) of
    object;
```

OnMeasureHeight allows changing the item height in the list. All items have the same height, and by default this height is determined by the current font size.

OnMeasureWidth

```
property OnMeasureWidth: TTBXLMeasureWidth;
type TTBXLMeasureWidth = procedure(Sender: TTBXCustomList; ACanvas: TCanvas; AIndex: Integer; var AWidth:
    Integer) of object;
```

Handle this property to specify a custom width for each item. This width is used by the component internally to determine the width of the list. Note, that unlike OnMeasureHeight, this event is fired for each item.

Class TTBXStringList

A toolbar item which displays a collection of strings in a scrollable list

Hierarchy

TTBCustomItem
|
TTBXCustomItem
|
TTBXCustomList
|
TTBXStringList

Description

TTBXStringList is an internal implementation of a scrollable list that uses a standard TStringList object for list element storage.

See also: Class TTBXUndoList

Properties

published

Strings

```
property Strings: TStrings;
```

A string list which stores the elements of the list.

Class TTBXUndoList

A toolbar list item for displaying list of undo actions etc.

Hierarchy

TTBCustomItem
|
TTBXCustomItem
|
TTBXCustomList
|
TTBXStringList
|
TTBXUndoList

Description

TTBXUndoList is a modified version of TTBXStringList. It is designed to control undo/redo/history lists.

The only difference is that all the entries in the list prior to selected are automatically highlighted.

Unit TBXMDI.pas

The classes defined in this unit provide support for applications which implement Multi-Document Interface (MDI).

Quick reference

classes:

- TTBXMDIHandler – A replacement for TTBMDIHandler
- TTBXMDIWindowItem – A replacement for TTBMDIWindowItem

Class TTBXMDIHandler

A replacement for TTBMDIHandler

Hierarchy

TComponent
|
TTBXMDIHandler

Description

TTBXMDIHandler should be used instead of TTBMDIHandler to provide the correct support for MDI in your application. For more information, see TTBMDIHandler description in Toolbar2000 documentation.

See also: Class TTBXMDIWindowItem

Class TTBXMDIWindowItem

A replacement for TTBMDIWindowItem

Hierarchy

TTBCustomItem
  |
TTBXCustomItem
  |
TTBXMDIWindowItem

Description

Place the TTBXMDIWindowItem into a submenu to mark the location of MDI list which will be filled in with captions of MDI child windows at runtime.

See the description for TTBMDIWindowItem in Toolbar2000 documentation for more information.

See also: Class TTBXMDIHandler

Unit TBXStatusBars.pas

This unit contains implementation of TBX status bars and their internal classes.

Quick reference

classes:

- TTBXCustomStatusBar – An internal implementation of TTBXStatusBar
- TTBXStatusBar – A status bar control with support for TBX themes
- TTBXStatusPanel – A panel in TTBXStatusBar
- TTBXStatusPanels – A container for TTBXStatusPanel objects

Class TTBXCustomStatusBar

An internal implementation of TTBXStatusBar

Hierarchy

TCustomControl
  |
TTBXCustomStatusBar

Description

TTBXCustomPageScroller is an internal base implementation for TTBXStatusBar. See TTBXStatusBar for more information.

Properties

FixAlign

property FixAlign: Boolean;

Use FixAlign in applications that support hiding and showing of the status bar to avoid status bar exchanging places with other controls aligned at the bottom of the form.

Images

property Images: TCustomImageList;

Lists the images that can appear in status bar panels.

See also: Class TTBXStatusPanel

Panels

property Panels: TTBXStatusPanels;

The Panels property holds a collection of TTBXStatusPanel objects.

See also: Class TTBXStatusPanels

SimplePanel

property SimplePanel: Boolean;

Determines whether the status bar displays a single panel or multiple panels. This property works the same way as the standard TPanel.SimplePanel.

SimpleText

property SimpleText: TCaption;

Contains the string that is displayed in the status bar when SimplePanel is set to *true*.

SizeGrip

property SizeGrip: Boolean;

Determines whether the status bar is resizable at runtime.

UseSystemFont

property UseSystemFont: Boolean;

UseSystemFont specifies whether the status bar uses the system font.

Methods

GetPanelAt

```
function GetPanelAt(const Pt: TPoint): TTBXStatusPanel;
function GetPanelAt(X, Y: Integer): TTBXStatusPanel;
```

Returns the panel at specified location relative to the status bar's top-left corner. If there is no panel at the specified point, this function returns *nil*.

GetPanelRect

```
function GetPanelRect(APanel: TTBXStatusPanel): TRect;
```

Returns a rectangle occupied by the panel. If the panel is hidden, this function returns an empty rectangle. Coordinates are specified relative to the top-left corner of the status bar.

Events

OnAdjustContentRect

```
property OnAdjustContentRect: TSBAdjustContentRect;
type TSBAdjustContentRect = procedure(Sender: TTBXCustomStatusBar; Panel: TTBXStatusPanel; var ARect:
    TRect) of object;
```

OnAdjustContentRect allows to customize the content rectangle of the panel. This is the rectangle to which internal parts of the panel are aligned.

OnAdjustFont

```
property OnAdjustFont: TSBAdjustFont;
type TSBAdjustFont = procedure(Sender: TTBXCustomStatusBar; Panel: TTBXStatusPanel; AFont: TFont) of
    object;
```

By processing the OnAdjustFont event, you can customize the font in each panel.

OnPanelClick

```
property OnPanelClick: TSBPanelEvent;
type TSBPanelEvent = procedure(Sender: TTBXCustomStatusBar; Panel: TTBXStatusPanel) of object;
```

Handle OnPanelClick to respond to mouse clicks on that panel.

OnPanelDblClick

property OnPanelDblClick: TSBPanelEvent;
type TSBPanelEvent = **procedure**(Sender: TTBXCustomStatusBar; Panel: TTBXStatusPanel) **of object**;

Handle OnPanelDblClick to respond to double clicks on that panel.

Class TTBXStatusBar

A status bar control with support for TBX themes

Hierarchy

TCustomControl
|
TTBXCustomStatusBar
|
TTBXStatusBar

Description

The TTBXStatusBar component is a replacement for the Delphi's standard TStatusBar component. The status bar consists of a row of panels. Each panel is represented by a TTBXStatusPanel object listed in the Panels property.

Class TTBXStatusPanel

A panel in TTBXStatusBar

Hierarchy

TCollectionItem
|
TTBXStatusPanel

Description

Each TTBXStatusPanel object is a member of a TTBXStatusPanels collection. It represents a panel in a TTBXStatusBar control.

Properties

published

Alignment

property Alignment: TAlignment;

Indicates the alignment of the caption within the status panel.

published

Caption

property Caption: TCaption;

Specifies the text appearing in the status panel.

published

Control

property Control: TControl;

A panel can control the size of one of a child control inside the status bar by constraining its location to the panel's boundaries.

published

Enabled

property Enabled: Boolean;

Indicates whether the panel is rendered in enabled or disabled style.

published

FontSettings

property FontSettings: TFontSettings;

Allows customizing caption font for each panel. This property works similarly to FontSettings

published

Framed

property Framed: Boolean;

Indicates that the status bar panel is painted with the frame or a separator, depending on the current theme.

published

Hint

property Hint: **string**;

Set the hint property to attribute a hint for the panel.

published

ImageIndex

property ImageIndex: TImageIndex;

Set ImageIndex to display an image in the panel. The images are obtained from the image list referenced by Images property.

published

MaxSize

property MaxSize: Integer;

Specifies the maximum size of the panel. When panels are stretching their width cannot exceed MaxSize. MaxSize is only active when it is greater than Size otherwise the panel can be stretched without limitations.

published

Size

property Size: Integer;

Indicates the normal size of the status bar panel. The actual size may become greater when the panel is stretched, but it may never exceed MaxSize.

published

StretchPriority

property StretchPriority: TPercent;
type TPercent = 0..100;

When the client area of the status bar exceeds the total sum of normal sizes of its panels, the status bar starts stretching some of the panels so that they continue to fill it from edge to edge. First, the panel with a higher stretch priority is stretched untill it width reaches MaxSize. It that is not enough, the panel with the highest priority is chosen amongst the rest of the panels, and the procedure proceeds untill the necessary stretching is achieved.

Panels with StretchPriority = 0 are never stretched.

published

Tag

property Tag: Integer;

Tag has no predefined meaning. It can be used for storing an additional integer value or it can be typecast to any 32-bit value such as a component reference or a pointer.

published

TextTruncation

property TextTruncation: TTextTruncation;
type TTextTruncation = twNone..twPathEllipsis;

Controls how the caption is displayed when it does not fit inside the panel.

twNone:
the caption is clipped

twEndEllipsis:
the caption is truncated and ellipses are added

twPathEllipsis:
replaces characters in the middle with ellipses so that the result fits in the specified rectangle. If the string contains backslash characters, twPathEllipsis preserves as much as possible of the text after the last backslash

published

ViewPriority

property ViewPriority: TPercent;
type TPercent = 0..100;

When the the status bar cannot fit all the panels, that is the client width is less than the total sum of all panels' Size properties, the panels with lower view priority become hidden. The status bar hides panels starting from those with lower ViewPriority until the rest of the panels can fit inside the client area. The panels with ViewPriority = 100 are never hidden. The panels with ViewPriority = 0 are always hidden.

Visible

property Visible: Boolean;

When the status bar shrinks due to resizing of the parent window, some panels may become hidden. Use the Visible property to determine which panels are visible.

See also: ViewPriority

Class TTBXStatusPanels

A container for TTBXStatusPanel objects

Hierarchy

TCollection
|
TTBXStatusPanels

Description

TStatusPanels holds a collection of TTBXStatusPanel objects in a TTBXStatusBar.

The inherited Count property contains the number of status panels in the collection.

Properties

Items

property Items[Index: Integer]: TTBXStatusPanel;

Lists the status panels in the collection.

StatusBar

property StatusBar: TTBXCustomStatusBar;

Contains a reference to the TTBXStatusBar control which owns the collection of panels.

Methods

FindPanel

function FindPanel(AControl: TControl): TTBXStatusPanel;

Finds the panel which contains a specified control.

Unit TBXSwitcher.pas

Quick reference

classes:

- TTBXSwitcher – A component which simplifies switching of themes

Class TTBXSwitcher

A component which simplifies switching of themes

Hierarchy

TComponent
|
TTBXSwitcher

Description

Using TTBXSwitcher, you can change the current TBX theme at runtime and at design time. Alternatively, you can use TBXCurrentTheme and TBXSetTheme routines.

Properties

published

EnableXPStyles

property EnableXPStyles: Boolean;

EnableXPStyles is effective only for applications running in Windows XP environment. This property controls the support for native Windows XP visual styles.

published

FlatMenuStyle

property FlatMenuStyle: TFlatMenuStyle;
type TFlatMenuStyle = (fmsAuto, fmsEnable, fmsDisable);

In automatic mode (fmsAuto), the appearance of popup menus corresponds to system settings; that is, flat menus will appear only on Windows XP with themes which support the new menu style. The other two options allow overriding system preferences. This setting does not have effect in the Office XP theme.

published

Theme

property Theme: **string**;

Set this property to control the appearance of toolbars and toolbar items. When reading, this property returns the name of the currently selected theme. Writing to this property will set the specified theme.

Standard TBX distribution includes the following themes:

- “Default”
- “OfficeXP”

- "Stripes"

See also: TBXSetTheme, TBXCurrentTheme

ThemeCount

property ThemeCount: Integer;

Returns the number of available themes.

Themes

property Themes[Index: Integer]: **string**;

Returns the name of the specified theme. The zero-based index should be in the 0..ThemeCount - 1 range.

Events

published

OnThemeChange

property OnThemeChange: TNotifyEvent;

The OnThemeChange event is fired when the current TBX theme is changed.

Unit TBXThemes.pas

TBXThemes declares types and classes which are necessary for TBX themes functionality. This unit should only be of interest for advanced users who want to implement custom themes or want to apply TBX themes for their custom controls.

Quick reference

types:

- TTBXItemLayout

Types

TTBXItemLayout

TTBXItemLayout = (tbxlAuto, tbxlGlyphLeft, tbxlGlyphTop);

TTBXItemLayout indicates relative alignment of caption and image in a toolbar item.

tbxlAuto
: The image is located below caption in vertical toolbars and left of the caption in the rest of toolbars. In addition, caption is rotated for vertical toolbars.

tbxlGlyphLeft
: Image is always left of the caption. Caption is always upright.

tbxlGlyphTop
: Image is always above the caption. Caption is always upright.

Unit TBXToolPals.pas

TBXToolPals contains implementation of tool palettes and color palettes.

Quick reference

classes:

- TTBXColorPalette – A toolbar item which a grid of colors
- TTBXColorSet – A component which defines 2D matrix of colors
- TTBXCustomColorSet – An internal implementation of color sets
- TTBXCustomToolPalette – A common ancestor for tool palettes
- TTBXToolPalette – A toolbar item which implements tool palettes

Class TTBXColorPalette

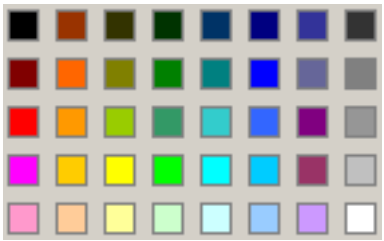
A toolbar item which a grid of colors

Hierarchy

TTBCustomItem
|
TTBXCustomItem
|
TTBXCustomToolPalette
|
TTBXColorPalette

Description

TTBXColorPalette is a tool palette descendant modified to display a color palette.



The color data displayed in the palette can be customized by setting the ColorSet property to reference an external TTBXColorSet object.

Properties

Color

published

property Color: TColor;

Specifies the current selected color.

published

ColorSet

property ColorSet: TTBXCustomColorSet;

By default, when ColorSet reference is set to nil, TTBXColorPalette uses internal color set with predefined colors. If you need to use custom colors or color names, you have to point the ColorSet property to the corresponding color set component.

See also: Class TTBXColorSet

Methods

ColorToString

function ColorToString(AColor: TColor): **string**;

Converts AColor into the string representation. If the color exists in the color matrix, its name will be retrieved from the color set, otherwise ColorToString returns the hexadecimal RGB code for the color.

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXColorPalette. Internally, creating a color palette also creates an internal hidden instance of a color set which provides default color data when there is no explicit reference by ColorSet.

FindCell

function FindCell(AColor: TColor): TPoint;

Returns the position (column, row) of the cell with specified color. If such cell does not exist, this method returns position with negative column value.

Class TTBXColorSet

A component which defines 2D matrix of colors

Hierarchy

TComponent
  |
TTBXCustomColorSet
  |
TTBXColorSet

Description

TTBXColorSet defines data for TTBXColorPalette. It provides the number of rows and columns in the color matrix as well as color values and names.

The dimensions of the matrix are stored as ColCount and RowCount properties. The values of colors and their names are requested dynamically through the OnGetColorInfo event.

Class TTBXCustomColorSet

An internal implementation of color sets

Hierarchy

TComponent
|
TTBXCustomColorSet

Description

TTBXCustomColorSet is an internal base class for color sets. For more information see TTBXColorSet.

See also: Class TTBXColorPalette

Properties

ColCount

property ColCount: Integer;

Specifies the number of columns in the color matrix.

RowCount

property RowCount: Integer;

Specifies the number of rows in the color matrix.

Methods

GetColor

function GetColor(Col, Row: Integer): TColor;

Returns the color for the specified element in color matrix. By default, this method invokes the OnGetColorInfo event to obtain the color information.

If GetColor returns clNone, the dependent TTBXColorPalette object will not be showing a color button at the specified position.

GetName

function GetName(Col, Row: Integer): **string**;

Returns the string representation for the color of specified cell. By default, this method invokes the OnGetColorInfo event to obtain the color information.

Events

OnGetColorInfo

property OnGetColorInfo: TCSGetColorInfo;
type TCSGetColorInfo = **procedure**(Sender: TTBXCustomColorSet; Col, Row: Integer; **var** Color: TColor; **var** Name: **string**) **of object**;

This event is regularly called by the color set to obtain the information about the color of the element at the position specified by Col and Row parameters.

Class TTBXCustomToolPalette

A common ancestor for tool palettes

Hierarchy

TTBCustomItem
|
TTBXCustomItem
|
TTBXCustomToolPalette

Description

TTBXCustomToolPalette is an internal base class for TTBXToolPalette. See TTBXToolPalette for more information.

Properties

ColCount

protected

property ColCount: TRowColCount;
type TRowColCount = 1..100;

A number of columns in the grid.

PaletteOptions

protected

property PaletteOptions: TTBXToolPaletteOptions;
type TTBXToolPaletteOptions = **set of** (tpoCustomImages, tpoNoAutoSelect);

A set of options which controls appearance and behavior of tool palette.

See also: OnDrawCellImage

RowCount

protected

property RowCount: TRowColCount;
type TRowColCount = 1..100;

A number of rows in the grid.

SelectedCell

protected

property SelectedCell: TPoint;

Indicates the current selected cell. SelectedCell.X contains the column and SelectedCell.Y – the row of the currently highlighted cell. If either of two coordinates is negative, then no cell will be highlighted.

Methods

Create

constructor Create(AOwner: TComponent);

Creates an instance of TTBXCustomToolPalette.

By default, the palette contains just a single cell. Therefore both ColCount and RowCount are set to 1.

Events

OnCalcImageSize

protected

property OnCalcImageSize: TTPCalcSize;
type TTPCalcSize = **procedure**(Sender: TTBXCustomToolPalette; Canvas: TCanvas; **var** AWidth, AHeight: Integer) **of object**;

If you provide your own images instead of the image list, this event should be handled to let the tool palette know the size of the image.

OnCellClick

protected

property OnCellClick: TTPCellClick;
type TTPCellClick = **procedure**(Sender: TTBXCustomToolPalette; **var** ACol, ARow: Integer; **var** AllowChange: Boolean) **of object**;

Handle this event to respond to changes in the selection initiated by user (Unlike the OnChange event, this event will not be fired upon the explicit change of SelectedCell in your code). In addition, you can customize the cell selectivity by altering the AllowChange parameter.

OnChange

protected

property OnChange: TNotifyEvent;

OnChange is fired each time when the SelectedCell property changes. Read SelectedCell to obtain the coordinates of the new selection.

OnDrawCellImage

protected

property OnDrawCellImage: TTPDrawCellImage;
type TTPDrawCellImage = **procedure**(Sender: TTBXCustomToolPalette; Canvas: TCanvas; ARect: TRect; ACol, ARow: Integer; Selected, Hot, Enabled: Boolean) **of object**;

When tpoCustomImages is in PaletteOptions, this event is the place where the custom image should be painted. Otherwise, you can use it to draw overlays over existing images taken from the image list.

See also: PaletteOptions

OnGetCellHint

protected

property OnGetCellHint: TTPGetCellHint;
type TTPGetCellHint = **procedure**(Sender: TTBXCustomToolPalette; ACol, ARow: Integer; **var** HintText: **string**) **of object**;

In OnGetCellHint event handler, you can customize the hint string appearing next to each item in the palette. This event will be regularly fired by the tool palette while the mouse is moved from one cell to another.

OnGetCellVisible

protected

property OnGetCellVisible: TTPGetCellVisible;
type TTPGetCellVisible = **procedure**(Sender: TTBXCustomToolPalette; ACol, ARow: Integer; **var** Visible: Boolean) **of object**;

When the tool palette is rendered on screen, it will run through all rows and columns requesting for cell visibility at each location. By responding to the OnGetCellVisible event you can customize visibility for each cell in the matrix.

Class TTBXToolPalette

A toolbar item which implements tool palettes

Hierarchy

TTBCustomItem
|
TTBXCustomItem
|
TTBXCustomToolPalette
|
TTBXToolPalette

Description

Tool palettes contain a grid of cells, each acting as a separate subitem. Notice however, that each tool palette is a single toolbar item and the cells are not the same toolbar items as the child items in TTBXSubmenuItem.

The images displayed in each cell are obtained from the TTBXCustomItem.Images property. By default, the upper left cell gets the image with index 0, the second cell in this row has index 1, etc.

TTBXToolPalette implements the generic behavior introduced in TTBXCustomToolPalette, but does not introduce any new behavior.

APPENDIX A

Changes

Version 2.1 beta 1 (29 May 2003)

- New TTBXButton control
- New Aluminum theme
- Panel objects now support transparency
- Fixed inconsistency in ShowAccelChar property in some toolbar items and controls
- Added TTBXCustomLabel.FocusControl property
- Added SimplePanel and SimpleText properties to status panels
- Added ShowAccelChar to TTBXLabelItem, TTBXLabel, TTBXLink, TTBXCheckBox
- Added the AutoScroll property to TTBXCustomPageScroller
- Added missing properties to TTBXVisibilityToggleItem (Layout, MinWidth, MinHeight, Stretch)
- Fixed resource leakage in TTBXThemeManager
- Fixed bugs with floating point and hex number parsing in spin edits
- Added UnregisterTBXTheme function
- Minor changes in theme rendering, third-party themes need to be upgraded

Version 2.0.1 (20 August 2003)

- Added a gradient to docked captions in the default theme (visible when XP visual styles are active)
- Fixed caption of TTBXToolWindow

- Fixed painting of separators in chevron submenus
- Fixed small errors in image alignment inside toolbar items

Version 2.0 (11 August 2003)

- Updated the code for compatibility with Toolbar2000 version 2.1.2
- Fixed a problem with color loading in TTBXColorItem
- Fixed buttons in spin edit items being active after disabling the item
- Fixed wrapping of items in large chevron popups
- Fixed icon rendering problems at some Win95 systems at 24bpp modes
- Fixed an issue with dock color on MDI forms
- Fixed large number handling in spin edit items
- Other minor bug fixes

Version 2.0 beta 3 (23 May 2003)

- Changed handling of toolbar and dock transparency (warning: some third-party themes might not be compatible with this feature)
- Fixed missing "dots" in the Stripes theme for vertical captions
- Fixed captions of dockable panels not showing ampersands
- Added TTBXDock which supports Windows XP themes (e.g. gradient in standard themes)
- Added TTBXMultiDock which supports docking of multiple dockable panels
- Fixed a bug in caption rotation in dockable panels
- Implemented painting of tool boxes with a gutter in popup menus
- Changed ToolboxPopup property of TTBXCustomItem—now all subitems will automatically have toolbar style
- Fixed bugs in painting of TTBXTextObject and its descendants when AutoSize=False
- Status bar panels now should respond to OnPanelClick in disabled state

Version 2.0 beta 2 (7 April 2003)

- TB2K 2.0.16 compatibility
- New documentation
- Added support for images in list box items and combo box items
- In TTBXPageScroller, the AutoScroll property has been renamed to AutoRange to make it more consistent with real behavior
- Published the forgotten TTBXToolPalette.Images property

- Fixed resource leakage in theme switching
- Added TTBXEditItem.StartEditing
- In the Default and Stripes theme, items in the menu bar are grayed when the application is inactive
- Added the TabStop and TabOrder properties to TTBXAlignmentPanel
- Fixed handling of VK_ENTER by TTBXLink and TTBXCheckBox and added
- Added OnEnter/OnExit events to TTBXLink
- Fixed the Alignment property if TTBXLink, TTBXCheckBox, and TTBXLabel. These items should support BiDiMode flags as well
- Removed the Contnrs unit dependence which would prevent compiling with D4
- Fixed a layout error in handling of tboImageAboveCaption when TTBXItem.Alignment = tbxlAuto

Version 2.0 beta 1 (21 December 2002)

- TB2K 2.0.15 and Delphi 7 compatibility
- Removed TTBXList—it is replaced with extended TTBXStringList
- Renamed TTBXComboItem -> TTBXDropDownItem and TTBXComboList -> TTBXComboBoxItem to keep class names more consistent
- Changed sizing behavior of dockable panels (see new properties DockedWidth, DockedHeight, FloatingWidth, and FloatingHeight). Now they can be docked to horizontal docks. Added the SmoothDockedResize property.
- Added the SnapDistance property to TTBXToolbar, TTBXToolWindow and TTBXDockablePanel for snapping to screen edges when floating.
- Modified TB2k design item editor to include custom hooks and added a hook for a separate toolbar with TBX items.
- Changed internal realization of TBX theme registration and allocation.
- New spin edit item with support for units, inline math, custom “constants”, etc.

Version 1.9.8 (17 May 2002)

- TB2K 2.0.14 compatibility

Version 1.9.7 (7 April 2002)

- BCB 6 compatibility (Changed DockableTo declaration in TTBXDockablePanel + fixed AutoCheck)
- Added TTBXItem.OnDrawImage event
- Some internal changes in the page scrolling to improve child alignment

- Appearance of disabled icons is now controlled with a compiler directive {$DEFINE ALTERNATIVE_DISABLED_STYLE} in TBXOfficeXPTheme.pas
- Other minor changes and bug fixes

Version 1.9.6 (27 March 2002)

- Added TTBXStatusBar component
- Some minor bug fixes

Version 1.9.5 (10 March 2002)

- TBX 2.0.12 compatibility
- Added TTBXLink, TTBXCheckBox and TTBXRadioButton controls
- Overriden a standard TColor property editor to include clHotLight system color
- Some minor bug fixes

Version 1.9.4 (23 February 2002)

- TB2K 2.0.11 compatibility
- Fixed – EInvalidTypecast exception was raised for floating tool windows when changing themes
- Fixed – wrong calculation of the edit item size when the edit item was following another item with non-standard font size
- Fixed – bugs in TTBXList.OnChange event generation
- Changes for better D4 compatibility
- Other minor bug fixes

Version 1.9.3 (14 February 2002)

- TB2K 2.0.10 compatibility
- Fixed painting of glyphs in MDI buttons and floating toolbars (before, they did not appear when WindowsXP manifests were enabled)
- Added TTBXToolbar.ItemTransparency to allow customizing the appearance of embedded toolbars
- Added simulation for transparent shadows in OfficeXP theme for Windows98/ME
- Fixed the TTBXCustomItem.Stretch property
- Fixed TTBXLabel.Underline
- Removed popup sounds from combo items

Version 1.9.2 (8 February 2002)

- Attention: See documentation or the web site for information on new license agreement, registration and donations
- New implementation of color mixing for OfficeXP theme
- Added dockable panels (a.k.a. task panes)
- Added tool palettes. If you are using the TTBXColorArray, please change it to TTBXColorPalette as TTBXColorArray will be removed in future versions
- Added support for transparent toolbars and toolbars with custom colors
- Fixed tboUseEditWhenVertical for edit items (it will work for combo boxes as well)
- The structure of the installation directory is changed
- Other minor changes and bug fixes

Version 1.9.1 (12 January 2002)

- Continuing internal reorganization
- Added TTBXCustomItem.Stretch property
- Added TTBXSwitcher.EnableXPStyles and TTBXSwitcher.FlatMenuStyle properties
- Removed TTBXEditItem.CharCase, since now it is implemented in tb2k
- Added TTBXEditItem.ReadOnly; TTBXCustomComboItem.AlwayseSelectFirst and TTBXCustomComboItem.DropDownList properties
- Some changes are made to color blending algorithms in OfficeXP theme, although keep in mind, that this is a work in progress
- Other minor changes and bug fixes

Version 1.9 (29 December 2001)

- Very serious internal changes, which will make the library easier to maintain. Some code cleaning is still required...
- Almost full support for WindowsXP visual styles
- Bugs causing access violation problems in windows 98/ME should be fixed
- The license agreement is modified. The new version is effective starting from this version of the package

Version 1.8.2 (23 December 2001)

- Better support for shadows in OfficeXP theme and better support for flat style menus in WindowsXP

Version 1.8.1 (22 December 2001)

- New: Better appearance of the Default theme in Windows XP

- Bug fixes

Version 1.8 (22 December 2001)

- New: tb2k 2.0.7 compatibility
- Added: Native shadows in OfficeXP theme (Windows 2000 or XP is required for transparent shadows); other themes support shadows in Windows XP, which are implemented in tb2k
- Added: OwnerDraw methods to string lists and combos
- Fixed: Windows shut down bug
- Changed: appearance of edit and combo items in Stripes theme
- Other minor changed and bug fixes

Version 1.7 (22 November 2001)

- New: tb2k 2.0.6 compatibility
- Fixed: Updated color mixing for popup edges (OfficeXP theme) and a few other minor painting issues
- Added: CharCase, PasswordChar, and OnChange to TTBXEditItem
- Added: AlwaysSelectFirst property to submenu items and combo boxes
- Added: Better handling of combo boxes... it is still not perfect though
- Updated: most painting routines. OfficeXP theme should now consume much less CPU cycles

Version 1.6 (5 November 2001)

- New: Lists and combo boxes
- Added: support for EditItems
- Added: C++ Builder Compatibility
- Added: Design time editor for multi-line captions
- Bugfix: Sounds should work now
- Bugfix: Fixed sometimes incorrect arrangement of elements within the toolbar item
- Bugfix: In OfficeXP theme, popups are now joined to their parent items, as they are supposed to be

Version 1.5 (15 October 2001)

- Added a new theme—Stripes
- Catching up with tb2k 2.0.5

Version 1.4 (7 October 2001)

- tb2k 2.0.4 compatibility
- Bugfix: Resource leakage in MDI menu items
- Added: Limited support for low color palette-based desktop modes
- Added: A theme switching component

Version 1.3 (17 September 2001)

- Bugfix: Wrong handling of mouse up events
- Bugfix: Updating of floating windows when theme is changed
- Bugfix: Erroneous painting of close and MDI buttons with some drivers

Version 1.2 (27 August 2001)

- Added support for chevrons, MDI buttons, MDI window items, floating toolbars and tool windows

Version 1.1 (15 August 2001)

- Bugfix: Disabled items clicking
- Bugfix: OfficeXP theme appearance now should be more consistent in 16bpp and 24bpp desktop color depths
- Bugfix: Wrong painting of disabled items highlighted with the mouse (W98 only)
- Added: Color items
- Other minor changes

Version 1.0 (7 August 2001)

- First Release

Index