Plasmatech Shell Control Pack - Introduction

Give your Delphi and C++Builder applications Windows Explorer functionality with the Shell Control Pack. Simply place four components on a form set two properties and you have a working Explorer! These powerful shell components include tree view, list view and combo box. Full access is available to drag and drop (into, out of and within your application), context menus including "Send To", renaming/deleting, copy/paste, namespace extensions and much more.

Add features to any application with the two-way splitter, image/indent combo box and enhanced panels. Enhanced list view and tree view controls support custom drawing, different fonts, colours and context menus for individual items.

Every application can benefit by replacing the limited and inflexible Windows common file dialogs. Microsoft did just that with Office 95 and 97. Now you too can liberate your program from the constraints of the common dialogs with 100% Delphi forms that look and act just like the system ones, only better! With extra functionality such as a resizable frame, tree view, and no three-character limit on file extensions, these components are useful in any application. And because they are Delphi forms, you can customise them directly or with visual form inheritance to add custom features like file previewing.

Internationalisation is fully supported, with translations provided for 16 languages – Czech, Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Japanese, Norwegian, Polish, Portuguese (Brazil), Russian, Spanish and Swedish.

The powerful components contained in the Shell Control Pack will give your software product distinguishing features and the standard interface your users are comfortable with. Stand out from the crowd. Order the Shell Control Pack today!

See Also

Installation | Component Summary | Unit Summary | Ordering Information | Latest Changes | Internationalisation

Installation

Delphi 2 | C++Builder 1 | All Other Environments

Users of versions **prior to v1.3h** should completely remove the earlier version, including help integration and library path entry, before installing the new version.

All Other Environments

(Delphi 3 or higher and C++Builder 3 or higher)

The Setup utility will make all the adjustments necessary to install the Shell Control Pack design-time package and online help into your chosen environments.

Source code customers should read about The problem with OLE2.

Delphi 2

Registering Components

1. From the Delphi IDE select **Component | Install**.
2. Select the **Add** button.
3. Browse for the **Reg_PTShellControls.pas** file and select OK.
4. Select **OK** again.

The component library will now be rebuilt, and should include the Shell Control Pack.

Integrating Keywords

1. Copy the PTShCtrl.hlp, PTShCtrl.cnt and PTShCtrl.kwf files into the Delphi HELP directory.
2. Start the help file installer located in the Delphi HELP\TOOLS directory called HELPINST.EXE.
3. Select **File | Open**.
4. Locate the DELPHI.HDX file located in the Delphi BIN directory.
5. Select **Keywords | Add keyword file…**
6. Locate the PTShCtrl.kwf file.
7. Select **File | Save**.

The Shell Control Pack help should now be integrated into the Delphi help system.

C++ Builder 1

Registering Components

1. From the Delphi IDE select **Component | Install**.
2. Select the **Add** button.
3. Browse for the **Reg_PTShellControls.pas** file and select OK.
4. Select **OK** again.

NOTE: C++Builder users must #include the *PTShConsts.h* file in *one* of their project *.cpp* files. If you don't do this the Shell Control Pack components will have no strings.

The component library will now be rebuilt, and should include the Shell Control Pack components.

Installing Help

Use the OpenHelp tool to install the **PTShCtrl.hlp** file into the C++ Builder help system. Read the OpenHelp.hlp file that comes with C++ Builder for more information on using OpenHelp.

Component Summary

Shell Components

 TPTShellTree	Shell tree view - the left-hand pane of Explorer.
 TPTShellList	Shell list view - the right hand pane of Explorer.
 TPTShellCombo	Shell Combo box - as seen in Explorer and common dialogs.
 TPTOpenDlg	Enhanced file open dialog - supports resizing, customisation and treeview.
 TPTSaveDlg	Enhanced file save dialog - supports resizing, customisation and treeview.
 TPTFolderBrowseDlg	Enhanced folder browse dialog - support resizing, customisation and create/delete folder buttons.

Support and Bonus Components

 TPTTreeView	Adds OnPTCustomDraw and OnPTCustomDrawEx events to TTreeView.
 TPTListView	Adds OnCustomPTDraw and OnPTCustomDrawEx events to TListView.
 TPTSplitter	General-purpose splitter panel.
 TPTImageCombo	Image combo box - associates an image with each item.
 TPTFrame	Non-windowed frame - more styles than TBevil.
 TPTGroup	Windowed container frame - more styles than TPanel.
 TPTCombobox	Adds OnSelEndOk, OnSelEndCancel, OnCloseUp and OnDeleteItem events to TCombobox.
 TPTSysFolderDlg	System folder browse dialog - accesses the system folder browser via SHBrowserForFolder.

See Also
Installation | Unit Summary

Unit Summary

Unit	Contains
FPTFolderBrowseDlg	Browse for folders dialog component and form.
FPTOpenDlg	Open/save dialog components and form.
UPTFrame	Group box and bevel replacement.
UPTImageCombo	Combo box supporting items with images and indentation.
UPTShell95	Declarations for SHELL32.DLL.
UPTShellControls	TPTShellTree, TPTShellList and TPTShellCombo components. TPTShTreeData, TPTShListData and TPTShComboData classes.
UPTShellUtils	Implements low-level utilities useful for dealing with shell interfaces and structures. Also includes utilities for creating and resolving shortcuts.
UPTSplitter	TPTSplitter, two-way splitter panel.
UPTSysFolderDlg	TPTSysFolderDlg, browse for folders using the system dialog.
UPTTreeList	TPTTreeView and TPTListView components .
Reg_PTShellControls	Registration unit for the components.

See Also
Component Summary | Installation

Latest Changes

Version	Date	Changes
v1.6	3-Jul-01	ADDED **Delphi 6** support. ADDED ShellGetSpecialFolderIconIndex function. FIXED popup menu for root-nodes. If the root node is not the Desktop, then the right-click popup menu is now correctly handled. FIXED 'access denied' error displayed on certain Windows 2000 systems due to the "System Volume Information" folder.

See Also
Revision History

Revision History

Version	Date	Changes
v1.6		See Latest Changes.
v1.5c	30-Mar-01	FIXED a memory leak in ShellFindCSIDLFromIdList.
		FIXED a problem that was causing system shutdown to not complete if a TPTShellTree or TPTShellList control was used.
		FIXED Access Violation which sometimes occurred in TPTShellTree on Windows 98 when folders were created and deleted.
		FIXED TPTShListData so extra columns can be added to TPTShellList.
v1.5b	12-Dec-00	FIXED a potential memory leak under Windows 2000.
		FIXED potential Access Violation when performing multiple drag-drop operations with the TPTShellTree.
		FIXED. Certain types of images would show with a black border on some Windows/IE/comctl.dll combinations.
		FIXED. Refreshes of TPTShellTree could cause the currently selected node to jump up one or two levels.
v1.5a	14-May-00	ADDED. Support for Windows 2000 - correct column information is now used for non-filesystem folders when available.
		CHANGED. Removed support for CF_TEXT format in drag/drop operations. This more closely matches Windows Explorer and prevents issues with some drop-target applications (such as Wordperfect).
		FIXED. Improved handling of the case where a background refresh occurs, while a popup menu or edit mode is active. Refreshes are deferred until the operation is complete.
		FIXED. Published RightClickSelect property in TPTShellTree.
v1.5	3-Mar-00	ADDED **C++Builder 5** support.
		ADDED *noUI* option to TLinkData record. Use this to prevent any error or search dialogs from displaying when resolving shortcuts.
		ADDED OnGetItemData method to TPTImageCombo control. Use of this property has improved performance of TPTShellCombo by delaying the retrieval of icon and text information for each item until it is needed.
		FIXED C++Builder header problems when using WebBrowser and Shell Controls in the same form.
		FIXED. Reversed the sort direction icon, now up=ascending, down=descending.
		FIXED. Published OnInfoTip event in TPTShellList.
		FIXED. TPTShellList.OnItemContextMenu and TPTShellTree.OnNodeContextMenu are now invoked correctly.
		FIXED WM_QUERYENDSESSION handling in hidden window used by DeviceChangeHandler.
		FIXED. TPTShellList Drag/drop now behaves more like Windows Explorer. Dropping on an item that does not accept dropped files now 'drops-through' to the owning folder. The effect is like you had dropped in empty white-space.
		FIXED. The update in v1.4a to better handle removable and dynamically mounted drives can cause excessive floppy drive access on some systems. It seems to happen only on IE5 systems, but only rarely. On those systems on which it does happen, it is reproducible. With this fix, those systems will only

		access the floppy drive after a removable disk is changed, or a drive is mounted/unmounted.
v1.4a	15-Dec-99	ADDED new property editor for TPTFrameStyle type, with custom drawing. FIXED update and refresh of drive volume labels for dynamically mounted drives such as CDROMs and ZIP drives. FIXED stack-fault (and invalid page fault) which occurred when using the TPTListView control (and consequently TPTShellList) with version 4.72.3110.1 of comctl32.dll. FIXED problem where each new CDROM mounted would give a new tree node, without deleting the previous node for that drive. FIXED case where folder column widths were sometimes not remembered between folders. FIXED OnContextPopup event in Delphi 5, now correctly published. FIXED occasional AV in Delphi 5 TVN_DELETEITEM handling. FIXED problem with sort direction icon. When switching to ViewStyle *vsReport*, the icon would not show until the header was resized. FIXED problem with the TPTShellTree control where OnInsertItem was not called for the root node. FIXED bug in Delphi 5 tree view where Delete/OnDelete were not called for all nodes in some situations. Delete/OnDelete are now correctly called in all cases. This bug has been reported to Borland.
v1.4	14-Sep-99	ADDED **Delphi 5** support. ADDED directional sort icons to TPTListView and TPTShellList. Additional properties are: • HeaderCanvas • HeaderDefaultDrawing • HeaderHandle • HeaderSortColumn • HeaderSortDirection • HeaderSortDisplayMode ADDED ShellFindCSIDLFromIdList to UPTShellUtils unit. IMPROVED support for custom sorting using OnCompare and SortColumn:=0. FIXED UPTShell95 unit declarations to avoid namespace clashes when used with C++Builder 4. FIXED editing of non-filesystem, renamable items like My Computer. It now works. FIXED OnGetImageIndex event of TPTShellList, it is now called correctly. FIXED problem where disconnected, mapped network drives could cause 100% CPU utilisation and general slowness. FIXED drag-drop error case where dropping from some data sources that don't support CF_IDLIST format would cause an access violation in the data source. FIXED handling of DefaultExt property in the case where the user enters an extension that matches DefaultExt, but that extension is not registered. The extension would be added again, now it isn't. FIXED occasional visual glitch with the size-grip control in the TPTOpenDlg, TPTSaveDlg and TPTFolderBrowseDlg controls.
v1.3h	29-Mar-99	ADDED **C++Builder 4** support.

		ADDED support for **Rename** menu item in file system context menus.
		ADDED size-grip to bottom right corner of TPTOpenDlg, TPTSaveDlg and TPTFolderBrowseDlg forms.
		ADDED "Show Desktop" button to TPTOpenDlg and TPTSaveDlg.
		IMPROVED performance of some loops in TPTShellList by using the *GetNextItem* algorithm posted in the FAQ.
		IMPROVED robustness of TPTShellLocator ReadData and WriteData methods. Now less likely to crash with corrupt form data.
		CHANGED PTShCreateNewFolder to support drives without long-filename capability.
		CHANGED behaviour when multiple items are selected in the TPTSaveDlg form. Before 1.3h, all the selected files needed to exist (*ptofFileMustExist* option was forced on). This is no longer the case.
		CHANGED CreatePanes method in TPTSplitter from private to public, enabling the run-time construction of TPTSplitter controls.
		FIXED key handling of Shift/Delete in TPTShellList and TPTShellTree controls. This key combo now deletes files immediately, bypassing the Recycle Bin in the same way as Windows Explorer.
		FIXED problem where mouse wheel was unresponsive for TPTShellList controls in *vsList* mode.
		FIXED OnDblClickOpen event in TPTShellList control. It was not being called when multiple items were opened simultaneously with a double click or Enter key.
		FIXED problem with EnsureTrailingCharDB function in UPTShellUtils. It didn't actually work for double-byte character sets.
		FIXED problem where assigning the Proportion property of TPTSplitter would not work if the Proportional property was *false*.
		FIXED problem where TPTImageCombo would always draw its background using the *clWindow* color. Now it uses its *Color* property.
		FIXED access violation on large-fonts systems using the TPTFolderBrowseDlg control.
		FIXED setting of current directory in TPTOpenDlg and TPTSaveDlg now sets the current directory on form exit to the Folder property of the list control.
v1.3g	1-Dec-98	ADDED **mouse wheel support** to TPTShellTree and TPTShellList (Delphi 4 only).
		ADDED TPTShellList.DoCommandForItem method.
		ADDED OnSelectItem event to TPTShellList (exposed Delphi 4 event).
		ADDED OnFillStart event to TPTShellList and TPTShellTree.
		FIXED bug in TPTCustomShellList.SetFileFilter method.
		FIXED DefaultExt property in TPTSaveDlg. It now works again.
		FIXED bug in TPTSplitter which could cause exceptions when reading splitter controls with 0 width or height.
		FIXED problem with TPTShellList where multiple filters would cause an empty floppy drive to timeout for each filter.
		CHANGED custom draw handling so that if the new Delphi 4 events are used, then the PTCustomDraw events are disabled.
		CHANGED OnFillComplete is now called at the end of Refresh methods as well as at the end of FillItems methods.

v1.3f	12-Jul-98	CHANGED the name of OnCustomDraw and OnCustomDrawEx events to OnPTCustomDraw and OnPTCustomDrawEx in order to prevent conflicts with Delphi 4. If you use these event handlers, you will need reattach them from the Object Inspector at design time. FIXED problem with OnAddListItem event of TPTOpenDlg and TPTSaveDlg not firing. FIXED problem with positioning of Create/Delete buttons in TPTFolderBrowseDlg. FIXED OnAddItem event of TPTFolderBrowseDlg not firing. CHANGED TPTShellTree.SortNode method from private to public. CHANGED some method parameter types to better support C++Builder 3. This should not affect existing code. CHANGED some expressions and types to better support Delphi 4. This should not affect existing code.
v1.3e	22-Apr-98	FIXED problem with TPTShellList.RefreshItems under Windows NT.
v1.3d	18-Apr-98	ADDED properties, events and methods to TPTFrmOpenDlg and TPTFrmFolderBrowseDlg to better support visual inheritance. Reference to the component object is no longer kept in the form object, making visual inheritance and customization a lot easier. ADDED OnCompare event to TPTShellList control. If SortColumn is 0, then the OnCompare event is called to provide custom sorting. ADDED OnFolderChanged event to TPTShellList. ADDED OnInsertItem event to TPTShellTree. This event is called after a node has been added to the tree. ADDED public Form property to TPTOpenDlg/TPTSaveDlg and TPTFolderBrowseDlg. ADDED new properties to TPTOpenDlg / TPTSaveDlg. OnFormClose, OnFolderChanged, OnSelectionChanged, OnFormShow and OnTypeChanged events. CHANGED algorithm used in TPTShellList.RefreshItems. It is now 10x faster (or better). FIXED Delphi drag support in TPTShellTree. If you disable OLE drag in TPTShellTree or TPTShellList, you can then activate Delphi drag by setting the DragMode property to dmAutomatic. FIXED problem with properties of TPTListView and TPTShellList controls. Checkboxes, Gridlines, HotTrack and RowSelect were not being published in C++Builder 3. FIXED problem where TPTSplitter was in non-full drag mode, when the user dropped the splitter, the OnSplitterDrop event would fire, but the OnChange event would not.
v1.3c	16-Mar-98	ADDED C++Builder 3 support. ADDED Proportion property to TPTSplitter. ADDED *ptsloShowHidden* to TPTShellList.Options, *ptstoShowHidden* to TPTShellTree.Options, *ptfbShowHidden* to TPTFolderBrowseDlg.Options and *ptofShowHidden* to TPTOpenDlg and TPTSaveDlg Options. When *true*, the control includes hidden and system files and folders, when *false* these items are omitted. If you already have forms that use TPTShellTree, TPTShellList, TPTFolderBrowseDlg, TPTOpenDlg and/or TPTSaveDlg you should check that this new option is set appropriately in your forms. CHANGED default autoscroll delay for TPTShellTree and TPTShellList from

		200ms to 100ms. FIXED memory leak in TPTShellList.RefreshItems. FIXED a number of cases where changing Options in TPTShellList and TPTShellTree controls at run-time would have no effect. FIXED a problem with sizing when a TPTSplitter control was placed inside an MDI child window.
1.3b	7-Feb-98	ADDED: You can now detect the version of COMCTL32.DLL installed by using the COMCTL32_VER variable in the UPTShellUtils unit. Also new is the GetModuleVersion function in the same unit. ADDED: Support for four new non-ANSI code page languages; Czech, Hungarian, Polish and Russian. ADDED: New examples. Check out http://plasmatech.com/scp/examples for the latest examples. ADDED: Added OnHelp event to TPTOpenDlg and TPTSaveDlg for Delphi 3 only. The OnHelp event is available in Delphi 2 and C++Builder in order to keep .dfm files compatible with Delphi 3, but the event is never invoked. CHANGED: ShellGetSystemImageList now supports link and share overlay images in Windows NT with Internet Explorer 4. Overlay images with other versions of Windows NT and Windows 95 still work. FIXED: TPTSplitter's Position could 'creep' to the left while resizing in Proportional mode. FIXED: *ptofNoValidate* now works in TPTOpenDlg/TPTSaveDlg components. FIXED: For TPTOpenDlg and TPTSaveDlg Options property, the *ptofShowHelp* option now works. FIXED: A private data member was not being correctly cleared in the TPTShListData object, causing various, minor, strange behaviours at times, especially when editing list item names. FIXED: When a shell tree and list were linked, and the tree control was recreated it would be empty. FIXED: When a tree with a non-null BaseFolder was linked to a list, the list would first show the desktop, then the BaseFolder. It no longer initially shows the desktop.
1.3a	7-Jan-98	ADDED: New examples. Check out http://plasmatech.com/scp/examples for the latest examples. ADDED: Proportional and FullDragMode properties and OnChange event to TPTSplitter. ADDED: Extra validity checking for node and item *.Data* properties. This makes it possible to add non-shell items to the TPTShellTree and TPTShellList controls. See the new *NonShellNodes* example project. ADDED: ShellGetIconIndexFromExt, ShellGetIconIndexFromPath. functions. ADDED: EnableAllChildren to TPTGroup. CHANGED: TPTShellTree and TPTShellList OnEdited event. Now gets fired after the file/folder is renamed, rather than before. FIXED: Excessive flicker during repainting of the TPTSplitter and TPTGroup controls has been minimised (Delphi 3 only). The frame of the TPTGroup control is now a true non-client area, like the frame for a standard window. Note that pre v1.3a TPTGroup controls will offset their children by the size of the non-client frame when upgraded to v1.3a. FIXED: Right-click on non-desktop base node would cause exception. FIXED: Fixed *ptofNoChangeDir* option in TPTOpenDlg and TPTSaveDlg. When *false*, the current directory is now changed. FIXED: Fixed positioning of Help button during form resizing in TPTOpenDlg

		and TPTSaveDlg.
1.3	28-Nov-97	CHANGED: Internationalisation support didn't work well with C++Builder. It's been changed so that all *.res* files are placed in the same directory as the *.pas* files - working around a C++Builder bug that prevents the use of pathed *.res* files. **C++Builder users should #include "PTShConsts.h" in ONE of their project's .cpp files.** If you don't do this then the shell controls will have no strings.
		ADDED: Internationalisation support.
		ADDED: New How To help topic for internationalisation.
		ADDED: fix for VCL's TListView.Color property. The background colour of the control, text and imagelist is now correctly set.
		ADDED: fix for VCL's TTreeView.Color and TTreeView.Font.Color properties. This fix only works with *comctl32.dll* v4.71(IE4). Other versions behave as they always did.
		ADDED: support in TPTShellList for remembering the widths of file system folder columns between folder changes.
		ADDED: support in TPTShellList for remembering the widths of file system folder columns between folders.
		ADDED: *ptfbCreateFolderIcon*, *ptfbDeleteFolderIcon* and *ptfbVirtualFolders* options to TPTFolderBrowseDlg.Options.
		ADDED: AutoSizeHeight property to TPTImageCombo.
		ADDED: Pane1 and Pane2 properties to TPTSplitter.
		FIXED: TPTShellCombo.GoUp and TPTShellTree.GoUp methods. The were ignoring the *aLevels* parameter and only going up 1 level.
		FIXED: problem where TPTFolderBrowseDlg.OnAddItem event was not being called.
		FIXED: inconsistent casing of TPTFolderBrowseDlg.SelectedPathName method. This change only affects C++Builder users.
		FIXED: TPTShellTree.OnExpanding event. Now gets invoked.
		FIXED: a problem on NT4 where setting a FileFilter in a TPTShellList would show an unnecessary error box when attempting to view a removable disk when no disk was present.
		FIXED: a problem under IE4 and Office97 where .htm, .doc, .xls files etc. could not be opened with a double-click on the shell list control.
		FIXED: a problem with TPTShellTree when BaseFolder was set to anything other than desktop. Any items dropped onto the root node would be dropped onto the desktop, rather than the actual root node folder.
		FIXED: When CreateNewFolder was called for TPTShellTree when the selected folder had no sub-folders, then the "New Folder" node was not displayed or editable.
		FIXED: non-functional *ptstoDefaultKeyHandling* and *ptstoOleDrag* options in TPTShellTree. They now work.
1.2b	12-Oct-97	ADDED: OnSplitterDrag and OnSplitterDrop events have been added to TPTSplitter.
		FIXED: a problem where changing the parent of a TPTShellCombo caused access violations. The Windows combobox control was sending multiple WM_DELETEITEM messages per item. A work-around has been applied.
		FIXED: Internet Explorer 4 (IE4) problem where floppy drives were being redundantly accessed.
		FIXED: SendTo menu did not work under IE4.
		FIXED: GDI leak due to a problem with the placement of `ShareImages:=true`.
1.2a	5-Oct-97	ADDED: OnFillComplete event to TPTShellList and TPTShellTree.

		FIXED: TPTShellList option *ptsloDontChangeFolder* now works.
		FIXED: problem with accessing node data in TPTShellList OnInsert event.
		FIXED: problem where after editing a list item's name, double clicking would use the previous name.
		FIXED: occasional slowdown due to event race condition when using dynamic update.
		FIXED: TPTShellList.DoCommandForAllSelected no longer reports an "Unspecified Error" if there are no items selected.
		FIXED: A bug in the first-release Windows 95 comctl32.dll (version 4.00.950) was causing an access violation. A work-around has been implemented.
1.2	6-Sep-97	ADDED: OLE drag and drop. *ptstoOleDrag* and *ptstoOleDrop* have been added to TPTShellTree.Options. *ptsloOleDrag* and *ptsloOleDrop* have been added to TPTShellList.Options. *ptofOleDrag* and *ptofOleDrop* have been added to Options for TPTOpenDlg and TPTSaveDlg.
		ADDED: C++Builder declarations and examples have been added to the help everywhere there is a Delphi declaration and/or example.
		ADDED: "SendTo" menu in item context menus now supports links other than .lnk and .pif. "Sending to" a shortcut to a drive now copies the files instead of opening the drive. You can now "send to" add-on accessories like the Microsoft Powertoys.
		ADDED: Significant performance improvements have been made to TPTShellList by deferring icon loading.
		ADDED: TPTShellList.RefreshItems has been added and is similar to TPTShellTree.RefreshNodes in that items are refreshed "in place". Only the changes to the list are effected - deleted items are removed, new items are added. This effectively maintains the current selection state during refreshes. Dynamic refresh is now done using this method instead of FillItems.
		ADDED: ShellGetIconIndex has been added to the UPTShellUtils unit.
		ADDED: Delphi 3's new TListView properties Checkboxes, Gridlines, HotTrack and RowSelect are now published in TPTShellList.
		ADDED: The Delphi 3 TListView property RowSelect is now published TPTListView.
		ADDED: TVN_GETDISPINFO handling of icons to TPTShellTree.
		FIXED: A problem that prevented splitters from being used in visual form inheritance was fixed. A TPTSplitter problem with SwapPanes at design time has also been fixed, and splitter bar drawing has been slightly improved.
		FIXED: DoCommand* methods now work correctly with string based verbs on all flavours of Windows.
		FIXED: TPTOpenDlg.OnInitialized was not being called.
		FIXED: The *ptofReadOnly* flag in TPTOpenDlg.Options was not being set after Execute returned.
		FIXED: The TPTOpenDlg.FilterIndex property was incorrectly zero based. It is now one based. This change will affect existing code that uses the property.
		FIXED: The problem with the OLE2 unit in Delphi 3 has been addressed with the included *fixole2.exe* utility (Developer and Client/Server versions of Delphi 3 only - there is no solution for Desktop versions).
		FIXED: The TPTShellTree component would remove the '+' symbol when trying to enumerate the folders for a removable drive when no disk was in the drive. The '+' is now removed only when a successful expand has taken place.
		FIXED: A memory leak in the protected method TPTShellList.FillList has been fixed.
		FIXED: OnClick and OnDblClick properties of TPTFrame and TPTGroup.
		FIXED: A problem with TPTShellList.OpenSelectedItems caused exceptions

		with large numbers of items.
		ADDED/FIXED: The TPTCustomDraw class is much more functional. The NoDefaultDrawing and Canvas properties now work, allowing full owner-drawing of individual items. A GDI leak when using the Font property has also been fixed. The ShellDemo program has been updated to show an example of drawing a whole item yourself.
		CHANGED: The list of items in TPTShellList is no longer refilled when a column-click is used to sort the items.
		CHANGED: All properties and methods containing the term *PathName* have been given consistent casing, capitalising the first letter of each word. Delphi users won't notice any difference. C++Builder users might need to change some of their code.
1.1a	6-Jul-97	ADDED: C++ Builder support. There is now a separate C++ Builder evaluation version and C++ Builder DCU, OBJ and HPP files have been added to the DCU version. The source version now imports directly into C++ Builder so does not include OBJ or HPP files.
		ADDED: Synchronize method to TPTShellTree, TPTShellList and TPTShellCombo. This method will force any pending updates to be fully completed before it returns.
		ADDED: *ptstoIncludeNonFolders* to TPTShellTree.Options, allowing non-folders to be shown as part of the shell tree view.
		FIXED: There was a problem where deleting the currently selected folder would cause exceptions.
		FIXED: When assigning the BaseFolder property of TPTShellTree at run time, the new root node would sometimes fail to initially enumerate its children.
1.1	26-Jun-97	ADDED: TPTTreeView and TPTListView components.
		ADDED: OnCustomDrawSh and OnCustomDrawShEx events to TPTShellTree and TPTShellList. The demo program has been updated to show how these events can be used to customise the drawing of individual shell items.
		ADDED: Improved popup menu handling on TPTShellTree and TPTShellList. You can now set the PopupMenu property of either control and have that menu used for the case when no items are selected and a popup is required. The demo program and TPTOpenDlg component have been enhanced to use this functionality.
		ADDED: OnTvCustomDrawSh and OnTvCustomDrawShEx events to TPTFolderBrowseDlg.
		ADDED: OnTvCustomDrawSh, OnTvCustomDrawShEx, OnLvCustomDrawSh and OnLvCustomDrawShEx events to TPTOpenDlg and TPTSaveDlg.
		ADDED: TPTFolderBrowseDlg.BaseFolder and TPTShellTree.BaseFolder properties.
		MOVED: The PTSH_CMDS_* constants have moved from the UPTShellControls unit to the UPTShellUtils.
		REMOVED: SortType property from TPTShellList. This property has no meaning for the shell list. Use SortColumn instead.
		FIXED: Renaming and deleting items in a TPTShellTree component would cause a linked TPTShellList to raise exceptions. This has been cleaned up significantly.
		FIXED: Keyboard handling for TPTShellList was intermittent when compiled on non-English versions of Delphi.
1c	30-May-97	OPTIMIZED: General speed improvements, especially TPTOpenDlg/TPTSaveDlg and TPTShellList - what took 40 seconds now takes from 3 to 5 seconds.
		ADDED: New option *ptsloHideFoldersWhenLinkedToTree* to TPTShellList.Options property. When this option is set *true* and the list control is linked to a shell tree control, file system folders are not shown in the list (like

		the way the open dialogs in Windows 3.1 worked). Obviously using this feature will make your interface non-standard, but some have argued why waste space showing the folders in the list when they are already visible in the tree? A similar option *ptofHideFoldersInListWhenTreeVisible* was added to TPTOpenDlg and TPTSaveDlg.
		ADDED: Delphi 3 adds new properties to TListView which have now been published in TPTShellList. They are Checkboxes, Gridlines and HotTrack.
		ADDED: TPTCustomShellTree and TPTCustomShellList components. These component follow the same pattern as Borland's "custom" components, they have all the implementation and no public or published properties. TPTShellTree and TPTShellList now derive from these "custom" components to promote properties to public or published. If you are deriving your own components from the shell controls, you should derive from the "custom" versions.
		ADDED: Editor for Filter properties of TPTOpenDlg, TPTSaveDlg and TPTShellList in both Delphi 2 and 3.
		ADDED: Extra properties to the TPTShListData class.
		CHANGED: The Columns property of TPTShellList has been promoted from protected to public. Since the columns are dynamically generated, this property is not stored and should not be written to.
		FIXED: Auto Fill options are now more rigorously applied. Especially TPTShellList.FileFilter - when auto fill is False you must call FillItems manually to apply property changes that affect a control's contents.
		FIXED: There was a problem with storing TPTShellLocator properties when using CSIDL values. Occasionally the property would be 'forgotten'. The *DefineProperties* method has been changed to fix this problem.
		REMOVED: TPTShellTree.OnDeletion and TPTShellList.OnDeletion events are no longer published. You should use TPTShellTree.OnDeleteItem and TPTShellList.OnDeleteItem events instead.
1b	17-May-97	ADDED: Support for Delphi 3. This includes some conditional compilation code and so consequently there are now separate Delphi 2 and Delphi 3 editions of the evaluation and DCU only versions. The source code version is common for both Delphi 2 and 3.
		FIXED: There is a bug in the Windows tree view control. If a folder has one child, and that child is deleted and a new child is created then the TVN_ITEMEXPANDING notification is not sent when the parent is expanded. This caused the TPTShellTree component to behave incorrectly. A work around has been implemented.
		FIXED: The Text and MaxLen properties of the TPTShellCombo control were erroneously published. It is invalid to use these properties and so they have been hidden.
		FIXED: There was undesirable design-time behaviour under Windows NT where execution of the current application would regularly stop if using the integrated debugger - with no indication of where it stopped or why.
		FIXED: There was an $C0000008 exception when running a SCP application at design-time under Windows NT. This was due to thread shutdown code. The problem did not manifest if the dynamic refresh option was not set.
1a	1-May-97	ADDED: TPTShellTree.SelectedItem property - similar to TPTShellList.SelectedItem property.
		ADDED: Help topics for TPTShellTree.GetDataFromNode and TPTShellList.GetDataFromItem.
		ADDED: IsWin95, IsWinNT and HasWin95Shell functions to UPTShellUtils.
		FIXED: TPTFolderBrowseDlg - when cursoring around the tree if you pressed Enter quickly, the most recently selected tree node was not returned as the SelectedPathName property. The problem did not exist if you the mouse was used or if there was a second or so delay after selecting the tree node with the

		keyboard and pressing Enter. FIXED: TPTShellTree - if a tree node name was edited, the SelectedFolder property did not reflect the change. FIXED: TPTShellTree - if a tree node was deleted (or the tree was automatically refreshed causing the current folder to change) then the SelectedFolder property did not reflect the change. FIXED: TPTFolderBrowseDlg - pressing F5 would sometimes cause a GP fault. FIXED: TPTOpenDlg or TPTSaveDlg would not start with a Filter other than "*.*"
1	21-Apr-97	Initial release.

See Also
Latest Changes

The Problem with OLE2.dcu in Delphi 3-5 and C++Builder 3-5

Delphi 3 introduced a new way of dealing with COM/OLE/ActiveX interfaces. However, code that needs to be portable to Delphi 2 and C++ Builder 1 cannot use the new interface features. Delphi 3+ and C++Builder 3+ still supports the older class-based interfaces declared in the OLE2 unit, but there is a serious problem.

For all packages installed in the IDE, only one instance of each DCU file is allowed. If there are two packages both requiring a single DCU then one of them cannot be installed. To get around this problem, Borland introduced the {$WEAKPACKAGEUNIT} directive. Units declared this way (such as Windows.pas) *can* be included in multiple packages - each package having its own private copy. There is no reason for OLE2 to *not* be weakly packaged, but it isn't.

There are two workarounds.

Option 1
If you have a Professional or Client/Server version of Delphi 3+ or C++Builder 3+, you can modify the source to OLE2.pas. This file can be found in the Source\Rtl\Win directory. Copy this file to the Lib\Delphi2 directory, edit the copy and add the directive {$WEAKPACKAGEUNIT} to the start of the file.

A utility *fixole2.exe* is included with the DCU and Source versions of the product (*1). This utility automatically locates the ole2.pas source code, copies it to the Lib\Delphi2 directory and patches it to include the $WEAKPACKAGEUNIT directive. *Fixole2.exe* will patch Delphi 3+ and C++Builder 3+.

Option 2
Designate a package that includes OLE2 to be the 'master' OLE2 container, then change every other package that depends on OLE2 to 'require' the master container.

(*1) *fixole2.exe* can also be downloaded from http://plasmatech.com/fixole2.exe.

Internationalisation - How To, etc…

Delphi 2/C++Builder 1 | Delphi 3-5 and C++Builder 3-5 | Non-ANSI codepages

Overview
The Shell Control Pack supports full translation of all text for twelve (12) languages using either *.res* files for Delphi 2 and C++Builder 1, or the *resourcestring* mechanism for Delphi 3-5 and C++Builder 3-5.

Currently Supported Languages
Czech*, Danish, Dutch, English, Finnish*, French, German, Hungarian*, Italian, Japanese, Norwegian, Polish, Portuguese (Brazilian), Russian*, Spanish, Swedish.

(*) Not full translation.

Future Language Support
Current plans are to add Korean, Chinese and Greek (no time frame has been set). No other languages are planned. If you would like another language supported, tell us at support@plasmatech.com.

Delphi 2 / C++Builder 1

In the Lang directory, run the *makeres.bat* file to generate the *.res* files. A bunch of *PTShConstsD2_??.res* files will be copied into the Lang directory's parent directory. A copy of each *.res* file will also remain in the individual Lang subdirectories.

NOTE: C++Builder users must add the *PTShConsts.cpp* file to their project or #include the *PTShConsts.h* file in *one* of their project *.cpp* files. Failing to do this will result in the Shell Control Pack components being without strings.

Compile Time Translation
To translate at compile-time, simply declare a compiler definition in your project options from this list.

Run Time Translation
For run-time translation, you should build the appropriate *.res* file into your resource DLLs. In your application, after loading your resource DLL you should assign its HINSTANCE to the global variable *gptshResourceInstance* (declared in the *UPTShConsts.pas* unit). Shell Control Pack components will attempt to load their strings from this module when needed.

Changing String IDs to Avoid Conflicts
You can change the sting-table ID numbers used by adjusting the *st_ptshbase* constant in the *PTShConstsD2.inc* file. C++Builder users should also adjust the *UPTShConsts.hpp* file or regenerate it if changing this value.

After changing the id base, run the *makeres.bat* file in the *Lang* directory to rebuild the ANSI resources. Japanese and Eastern European (Czech, Hungarian, Polish, Russian) version must be built manually on the appropriate language version of Windows (see Non-ANSI Codepages below).

Delphi 3-5 and C++Builder 3-5

Compile Time Translation
To use compile-time translation you simply define a compile-time symbol, which changes the *.inc* file that is included in the *UPTShConsts.pas* unit.

Run Time Translation
An example of run-time translation is include in the Shell Control Pack Examples zip file available from http://plasmatech.com/ptscp_examples.zip and also as part of the Shell Demo application that comes with the evaluation version.

The MergeDRC Application
This utility merges a CSV format file of translations with a Delphi-generated *.drc* file. The merging is done on the string name, which is relatively constant so even if Delphi re-orders the strings, the merging will still be successful.

Run *MergeDRC.exe* from the command-line with no parameters to get help.

Non-ANSI Codepages

If you are building resources on an ANSI code-page machine (most European languages), you cannot build non-ANSI code-page resources. "Building" involves translating a *.rc* to a *.res*, or compiling a source-code string (*.inc*) into an executable. The fundamental operation being performed (that is not supported by the OS in this case) is the translation of a multi-byte source file (*.rc*, *.inc*, *.pas*) into a Unicode output file (*.res*, *.exe*, *.dll*). Such support must be built into the operating system.

There are three sets of incompatible code pages supported; Japanese, Eastern European (Czech, Hungarian, Polish, Russian) and ANSI (all the rest). All Windows versions can build ANSI language resources. Any version of Windows from the Eastern European set should build all the Eastern European languages (although only Russian Windows has actually been tested for this). You must use Japanese Windows to build Japanese resources.

For example, to build the Japanese *.res* files, or include a Japanese *.inc* into your executable, you must do the building under Japanese Windows, since that is the only version of Windows capable of translating multi-byte Japanese into Unicode.

Pre-compiled *.res* files are provided for all non-ANSI code page languages; Czech, Japanese, Hungarian, Polish and Russian. Delphi 2 and C++Builder users can use these translations directly. Users of Delphi 3 or later, or C++Builder 3 or later will need to build their translated applications or translation DLLs under the appropriate version of windows.

Language Compiler Defines

Language	Symbol
Czech	LANG_CS
Danish	LANG_DA
Dutch	LANG_NL
English	LANG_EN (or none)
Finnish	LANG_FI
French	LANG_FR
German	LANG_DE
Hungarian	LANG_HU
Italian	LANG_IT
Japanese	LANG_JP
Norwegian	LANG_NO
Polish	LANG_PL
Portuguese	LANG_PT (Brazilian)
Russian	LANG_RU
Spanish	LANG_ES
Swedish	LANG_SV

Languages Not Fully Translated

All but one of the approximately 100 translated strings are the same strings used by the appropriate language version of Windows NT 4. The single string that needs manual translation is "Show Tree (F12)". Translations for this string have been generously donated by Shell Control Pack customers. The languages marked are those for which no translation has been provided for this single string. The other approx. 100 strings are properly translated in all supported languages.

If you can provide a translation for "Show Tree" in a language other than those listed and Windows NT 4 is available in that language, please send your translation to info@plasmatech.com. If you like, your name/company and homepage can be listed on the Thanks For Translation Assistance page.

Thanks for Translation Assistance Go to…

A number of people were generous enough to donate a translation for the manually translated portions of the Shell Control Pack. The are presented here listed in alphabetical order:

Contributor	Company or Personal HomePage
Manuel Onate	http://momsoft.pair.com
Cream Software - The Internet Software Developer	http://www.creamsoft.com/english/

Only those contributors who specifically requested to be added to this page are listed. If you contributed and would like to appear here, get in touch at info@plasmatech.com.

FPTOPENDLG UNIT

FPTOpenDlg unit

This unit contains the TPTOpenDlg and TPTSaveDlg components, as well as the TPTFrmOpenDlg form. The open and save components create instances of the form. You can visually inherit from TPTFrmOpenDlg to add or hide controls and change or add behaviour.

How to Visually Inherit

Components

- TPTOpenDlg
- TPTSaveDlg

TPTFrmOpenDlg Form

Both TPTOpenDlg and TPTSaveDlg use this form for their functionality.

TPTOpenDlg component

Hierarchy Properties Methods Events

Delphi Unit
FPTOpenDlg

C++Builder Header
FPTOpenDlg.hpp

Description
The TPTOpenDlg component is a replacement for the standard TOpenDialog component with many more features. The purpose of the dialog is still to let a user specify a file to open. Use the Execute method to display the dialog.

When the user clicks OK, the selected filename is stored in the FileName property.

You can let the user decide which files to make visible in the list box of the Open dialog box with the Filter property. The user can then use the List Files of Type combo box to determine which files display in the list. You set the default filter using the FilterIndex property.

You can permit the user to choose multiple filenames with the Options property so that the Files property contains a list of all the selected filenames in the list. You can customise how the Open dialog box appears and behaves with the Options property.

If you want a file extension automatically appended to the filename typed in the File Name edit box of the Open dialog box use the DefaultExt property.

Change the appearance of individual tree or list items with the OnLvCustomDrawSh and OnTvCustomDrawSh events.

You can test the component at design time from its right-click menu.

Hierarchy

TObject
|
TPersistent
|
TComponent
|
TPTDialog
|
TPTFileDlg
|
TPTCustomOpenDlg

TPTOpenDlg properties

TPTOpenDlg Legend

Properties in TPTOpenDlg

- DefaultExt
- Executing
- FileName
- Files
- Filter
- FilterIndex
- Form
- FormWidth
- FormHeight
- FormWindowState
- FormSplitterPos
- HistoryList
- InitialDir
- Options
- Title

TPTOpenDlg methods

TPTOpenDlg Legend

Methods in TPTOpenDlg

- Execute
- ReadStateFromRegistry
- ReadStateFromStream
- WriteStateToRegistry
- WriteStateToStream

TPTOpenDlg events

TPTOpenDlg Legend

Events derived from TPTCustomOpenDlg

OnAddListItem
OnAddTreeItem
OnAddComboItem
OnFolderChanged
OnFormShow
OnFormClose
OnHelp
OnInitialized
OnLvCustomDrawSh
OnLvCustomDrawShEx
OnSelectionChanged
OnTvCustomDrawSh
OnTvCustomDrawShEx
OnTypeChanged

DefaultExt property

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration
```
property DefaultExt: String;
```

C++Builder Declaration
```
__property System::AnsiString DefaultExt;
```

Description
The DefaultExt property specifies the extension that is added to the file name the user types in the File Name edit box if the user doesn't include a file-name extension in the filename. If the user specifies an extension for the filename, the value of the DefaultExt property is ignored. If the DefaultExt value remains blank, no extension is added to the filename entered in the File Name edit box.

Legal extensions can be greater than 3 characters in length. Don't include the period (.) that divides the filename and its extension.

Filter property

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration
```
property Filter: String;
```

C++Builder Declaration
```
__property System::AnsiString Filter;
```

Description
The Filter property determines the file masks available to the user for use in determining which files display in the dialog box's list box.
A file mask or file filter is a file name that usually includes wildcard characters (*.PAS, for example). Only files that match the selected file filter are displayed in the dialog box's list box, and the selected file filter appears in the File Name edit box. To specify a file filter, assign a filter string as the value of Filter. To create the string, follow these steps:

1 Type some meaningful text that indicates the type of file.
2 Type a | character (this is the "pipe" or "or" character).
3 Type the file filter.

Don't put in any spaces around the | character in the string.
Here's an example:

Delphi:
```
OpenDialog1.Filter := 'Text files|*.TXT';
```

C++Builder:
```
OpenDialog1->Filter = "Text files|*.TXT";
```

If you entered the preceding example as the Filter of an Open or Save dialog box, the string "Text files" appears in the List Files of Type drop-down list box when the dialog box appears in your application, the file filter appears in the File Name edit box, and only .TXT files appear in the list box. You can specify multiple file filters so that a list of filters appears in the List Files of Type drop-down list box or in the filter combo box. This allows the user to select from a number of file filters and determine which files are displayed in the list box.

To specify multiple file filters,

1 Create a file filter string as previously shown.
2 Type another file filter in the same way, but separate the second file filter from the first with the | character.
3 Continue adding as many file filters as you like, separating them with the | character. The string can be up to 255 characters.

Here's an example of three file filters specified as the value of the Filter property:

'Text files (*.TXT)|*.TXT|Pascal files (*.PAS)|*.PAS|Quattro Pro files (*.WB1)|*.WB1'

Now when the dialog box appears, the user can choose from three file filters that appear in the List Files of Type drop-down list box.

Note that the previous example includes the file filters in parentheses in the text parts. This isn't required, but it's a common convention that helps users understand what to expect when they select a file filter.
You can string multiple wildcard file filters together if you separate them with semicolons:

Delphi:
```
OpenDialog1.Filter := 'All files|*.TXT;*.PAS;*.WB1';
```

C++ Builder:
```
OpenDialog1->Filter = "All files|*.TXT;*.PAS;*.WBA";
```

FilterIndex property

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration
```
property FilterIndex: Integer;
```

C++Builder Declaration
```
__property int FilterIndex;
```

Description
The FilterIndex property determines which file filter specified in the Filter property appears as the default file filter in the List Files of Type drop-down list box. For example, if you set the FilterIndex value to 2, the second file filter listed in the Filter property becomes the default filter when the dialog box appears. The default FilterIndex value is 1. If you specify a value greater than the number of file filters in the Filter property, the first filter is chosen.

The default value is 1.

Form property

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration
```
property Form: TPTFrmOpenDlg;
```

C++Builder Declaration
```
__property TPTFrmOpenDlg* Form;
```

Description
Read only. Use this property to gain access to the form itself.

FormWidth property

See Also

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg components

Delphi Declaration
```
property FormWidth: Integer;
```

C++Builder Declaration
```
__property int FormWidth;
```

Description
The FormWidth property allows you to specify a width for the form. If FormWidth is -1 then a default width is used. You would typically assign this property before calling Execute. It has no effect while the dialog is executing.

When the dialog is closed (either successfully or cancelled) this property is set to the width the user adjusted the form to. You can use this feature to implement persistent form sizing. The WriteStateToRegistry, ReadStateFromRegistry and WriteStateToStream, ReadStateFromStream methods can do the persistence for you.

See Also

FormHeight property
FormWindowState property
FormSplitterPos property

ReadStateFromRegistry method
ReadStateFromStream method
WriteStateToRegistry method
WriteStateToStream method

FormHeight property

See Also

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg components

Delphi Declaration
```
property FormHeight: Integer;
```

C++Builder Declaration
```
__property int FormHeight;
```

Description
The FormHeight property allows you to specify a height for the form. If FormHeight is -1 then a default height is used. You would typically assign this property before calling Execute. It has no effect while the dialog is executing.

When the dialog is closed (either successfully or cancelled) this property is set to the height the user adjusted the form to. You can use this feature to implement persistent form sizing. The WriteStateToRegistry, ReadStateFromRegistry and WriteStateToStream, ReadStateFromStream methods can do the persistence for you.

See Also

FormWidth property
FormWindowState property
FormSplitterPos property

ReadStateFromRegistry method
ReadStateFromStream method
WriteStateToRegistry method
WriteStateToStream method

FormWindowState property

See Also

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg components

Declaration
```
property FormWindowState: TWindowState;
```

C++Builder Declaration
```
__property TWindowState FormWindowState;
```

Description
The FormWindowState property allows you to specify how the form appears on the screen. You would typically assign this property before calling Execute. It has no effect while the dialog is executing.

FormWindowState can have one of the following TWindowState values:

Value	Meaning
wsNormal	The form appears in its normal state (that is, its non-minimized, non-maximized state).
wsMinimized	The form appears in its minimized state.
wsMaximized	The form appears in its maximized state.

When the dialog is closed (either successfully or cancelled) this property is set to current form window state. You can use this feature to implement persistent form sizing. The WriteStateToRegistry, ReadStateFromRegistry and WriteStateToStream, ReadStateFromStream methods can do the persistence for you.

See Also

FormHeight property
FormWidth property
FormSplitterPos property

ReadStateFromRegistry method
ReadStateFromStream method
WriteStateToRegistry method
WriteStateToStream method

FormSplitterPos property

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration
```
property FormSplitterPos: Integer;
```

C++Builder Declaration
```
__property int FormSplitterPos;
```

Description
The FormSplitterPos property allows you to specify a starting position for the splitter bar between the tree and list views. You would typically assign this property before calling Execute. It has no effect while the dialog is executing.

When the dialog is closed (either successfully or cancelled) this property is set to the position of the splitter bar between the tree and list views. You can use this feature to implement persistent form sizing. The WriteStateToRegistry, ReadStateFromRegistry and WriteStateToStream, ReadStateFromStream methods can do the persistence for you.

See Also

FormHeight property
FormWidth property
FormWindowState property

ReadStateFromRegistry method
ReadStateFromStream method
WriteStateToRegistry method
WriteStateToStream method

HistoryList property

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration
```
property HistoryList: TStrings;
```

C++Builder Declaration
```
__property Classes::TStrings* HistoryList;
```

Description
The HistoryList property is provided for compatibility with earlier versions of TOpenDialog. This property might be implemented in a future revision.

InitialDir property

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration
```
property InitialDir: String;
```

C++Builder Declaration
```
__property System::AnsiString InitialDir;
```

Description
The InitialDir property determines the current directory when the dialog box first appears and value of the InitialDir property is shown as the current directory in the directory tree. Only files in the current directory appear in the dialog box's list of filenames. After the dialog box appears, users can then use the directory tree to change to another directory if they want.

When specifying the initial directory, include the full path name. For example,

```
C:\WINDOWS\SYSTEM
```

If no initial directory is specified, the directory that is current when the dialog box appears remains the current directory. The same is true if you specify a directory that does not exist.

Options property

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration

```
type TPTOpenOption = (ptofAllowMultiselect, ptofCreatePrompt,
                      ptofExtensionDifferent, ptofFileMustExist, ptofHideReadOnly,
                      ptofNoChangeDir, ptofNoDereferenceLinks,
                      ptofNoReadOnlyReturn, ptofNoTestFileCreate, ptofNoValidate,
                      ptofOverwritePrompt, ptofReadOnly, ptofPathMustExist,
                      ptofShareAware, ptofShowHelp, ptofAllowTree, ptofShowTree,
                      ptofShowHints, ptofHideFoldersInListWhenTreeVisible,
                      ptofOleDrag, ptofOleDrop, ptofShowHidden);

     TPTOpenOptions = set of TPTOpenOption;

property Options: TPTOpenOptions;
```

C++Builder Declaration

```
enum TPTOpenOption { ptofAllowMultiselect, ptofCreatePrompt, ptofExtensionDifferent,
ptofFileMustExist, ptofHideReadOnly, ptofNoChangeDir, ptofNoDereferenceLinks,
ptofNoReadOnlyReturn, ptofNoTestFileCreate, ptofNoValidate, ptofOverwritePrompt,
ptofReadOnly, ptofPathMustExist, ptofShareAware, ptofShowHints,
ptofHideFoldersInListWhenTreeVisiblem, ptofOleDrag, ptofOleDrop, ptofShowHidden };

typedef Set<TPTOpenOption, ptofAllowMultiselect, ptofOleDrop>  TPTOpenOptions;

__property TPTOpenOptions Options;
```

Description
These are the possible values that can be included in the Options set for the Open and Save dialog boxes:

Identifier	Meaning
ptofAllowMultiselect	When True, this option allows users to select more than one file in the File Name list view. Use the Files property to access the list of files.
ptofCreatePrompt	When True, this option displays a dialog box with a message if the user enters a filename that doesn't exist in the File Name edit box and chooses OK (Open/Save). The message tells the user the file doesn't exist and asks if the user wants to create a new file with that name.
ptofExtensionDifferent	This option is set when the filename returned from the dialog box has an extension that differs from the default file extension, the value in the DefaultExt property. Your application can then use this information. Setting a ptofExtensionDifferent value with the Object Inspector has no meaning.
ptofFileMustExist	If True, this option displays a dialog box with a message if the user enters a file that doesn't exist in the File Name edit box and chooses OK. The message informs the user the file can't be found and asks the user to make sure they entered the correct path and filename.
ptofHideReadOnly	Hides the read-only checkbox.
ptofNoChangeDir	When False, the current directory is changed to the last folder accessed by the dialog - whether the dialog was accepted or cancelled.
ptofNoDereferenceLinks	If True, directs the dialog box to return the path and filename of the selected shortcut (.LNK) file. If this value is not given, the dialog box returns the path and filename of the file referenced by the shortcut.

ptofNoReadOnlyReturn	If True, a message box appears informing the user if the selected file is read-only.
ptofNoTestFileCreate	This option applies only when the user wants to save a file on a create-no-modify network share point, which can't be opened again once it has been opened. If ptofNoTestFileCreate is True, your application won't check for write protection, a full disk, an open drive door, or network protection when saving the file because doing so creates a test file. Your application will then have to handle file operations carefully so that a file isn't closed until you really want it to be. *Not implemented 2-January-98.*
ptofNoValidate	If True, this option doesn't prevent the user from entering invalid characters in a filename. If pfofNoValidate is False and the user enters invalid characters for a filename in the File Name edit box, a message dialog box appears informing the user the filename contains invalid characters.
ptofOverwritePrompt	If True, this option displays a message dialog box if the user attempts to save a file that already exists. The message informs the user the file exists and lets the user choose to overwrite the existing file or not.
ptofReadOnly	If True, the Read Only check box is checked when the dialog box is displayed.
ptofPathMustExist	If this option is True, the user can type only existing path names as part of the filename in the File Name edit box. If the user enters a path name that doesn't exist, a message box appears informing the user that the path name is invalid. *Not implemented.*
ptofShareAware	If True, the dialog box ignores all sharing errors and returns the name of the selected file even though a sharing violation occurred. If ptofShareAware is False, a sharing violation results in a message box informing the user of the problem. *Not implemented 2-January-98.*
ptofShowHelp	If True, this option displays a Help button in the dialog box.
ptofAllowTree	If True then a "Show Tree" button is placed on the button bar.
ptofShowTree	If True then shows a tree view to the left of the list view, like a mini-explorer. This flag also reflects the visibility state of the tree after the user dismisses the dialog.
ptofHideFoldersInListWhenTreeVisible	When this option is set *true* and the list control is linked to a shell tree control, file system folders are not shown in the list (like the way the open dialogs in Windows 3.1 worked). Obviously using this feature will make your interface non-standard, but some have argued why waste space showing the folders in the list when they are already visible in the tree?
ptofShowHints	If True then popup hints are enabled on the button bar.
ptofOleDrag	When True items can be dragged out of the tree and list.
ptofOleDrop	When True items can be dropped onto the tree and list.
ptofShowHidden	When *true*, hidden and system files and folders are included in the tree and list, otherwise they are excluded.

The default value is [ptofHideReadOnly, ptofAllowTree, ptofShowHints, ptofOleDrag, ptofOleDrop, ptofShowHidden].

Title property

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg components

Delphi Declaration
```
property Title: String;
```

C++Builder Declaration
```
__property System::AnsiString Title;
```

Description
This property appears in the caption of the form. If unassigned an appropriate default title is used.

Executing property

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg components

Delphi Declaration
```
property Executing: Boolean;
```

C++Builder Declaration
```
__property bool Executing;
```

Description
Run time and read only. This property is *true* while the dialog is active.

FileName property

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration
```
property FileName: String;
```

C++Builder Declaration
```
__property System::AnsiString FileName;
```

Description
The FileName property specifies the file name that appears in the File Name edit box when the dialog box opens. The user can then select that filename or specify any other. Once the user specifies a filename and chooses OK, the value of the FileName property becomes the name of the file the user selected.
The path name can include a path. For example, to open the file README.TXT in the directory C:\TEMP, set FileName to C:\TEMP\README.TXT.

The FileName property can be set to the name of a file that doesn't exist in the current directory. In an Open dialog box, you can use this capability to let the user open a new file, and in a Save dialog box, the user can save a file that hasn't been saved before.

Files property

Applies to
TPTOpenDlg, TPTSaveDlg

Delphi Declaration
```
property Files: TStrings;
```

C++Builder Declaration
```
__property Classes::TStrings* Files;
```

Description
Run-time and read-only. The Files property value contains a list of all the file names selected in the Open or Save dialog box including the path names.

To let users select multiple file names in the dialog box, include ptofAllowMultiSelect in the Options property set (set ptofAllowMultiSelect to True).

The first item in the list of names is returned as the value of the FileName property.

OnAddListItem event

Applies to
TPTOpenDlg, TPTSaveDlg

Delphi Declaration
```
property OnAddListItem: TPTShAddItemEvent;
```

C++Builder Declaration
```
__property TPTShAddItemEvent OnAddListItem;
```

Description
This event is called every time an item is added to the file list. You have the opportunity to filter out unwanted items. You should respond to this event in the same way you would respond to the OnAddItem event in TPTShellList.

You would typically use the same event handler for all three OnAddTreeItem, OnAddListItem and OnAddComboItem events. If you use different event handlers you must ensure there are no logical inconsistencies in what you are filtering. For example, if something appears in the list, it must also appear in the tree and combo. You can safely filter anything from the list that is still present in the tree and combo. The only things you should really filter from the combo are first-level items such as My Computer or Network Neighbourhood.

Example
This example filters all folders from the list view (you can already see them on the tree view, why show them on the list as well?).

Delphi
```
procedure TMyForm.PTShellTree1AddItem(
      aSender: TObject;
      aParentIShf: IShellFolder;
      aParentAbsIdList: PItemIdList;
      aItemRelIdList: PItemIdList;
      aAttribs: Integer;
      var afAllowAdd: Bool );
begin
   afAllowAdd := ((aAttribs and SFGAO_FOLDER)=0);
end;
```

C++Builder
```
void __fastcall TMyForm::PTShellTree1AddItem(TObject *aSender,
  IShellFolder *aParentIShf, PItemIDList aParentAbsIdList,
  PItemIDList aItemRelIdList, int aAttribs, LongBool &afAllowAdd)
{
  afAllowAdd = ((aAttribs & SFGA_FOLDER)==0);
}
```

OnAddTreeItem event

See Also

Applies to
TPTOpenDlg, TPTSaveDlg

Delphi Declaration
```
property OnAddTreeItem: TPTShAddItemEvent;
```

C++Builder Declaration
```
__property TPTShAddItemEvent OnAddTreeItem;
```

Description
This event is called every time an item is added to the tree view. You have the opportunity to filter out unwanted items. You should respond to this event in the same way you would respond to the OnAddItem event in TPTShellTree.

You would typically use the same event handler for all three OnAddTreeItem, OnAddListItem and OnAddComboItem events. If you use different event handlers you must ensure there are no logical inconsistencies in what you are filtering. For example, if something appears in the list, it must also appear in the tree and combo. You can safely filter anything from the list that is still present in the tree and combo. The only things you should really filter from the combo are first-level items such as My Computer or Network Neighbourhood.

See Also

OnAddListItem event
OnAddComboItem event

TPTShellTree.OnAddItem event

OnAddComboItem event

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration
```
property OnAddComboItem: TPTShAddItemEvent;
```

C++Builder Declaration
```
__property TPTShAddItemEvent OnAddComboItem;
```

Description
This event Is called every time an item is added to the combo box. You have the opportunity to filter out unwanted items. You should respond to this event in the same way you would respond to the OnAddItem event in TPTShellCombo.

You would typically use the same event handler for all three OnAddTreeItem, OnAddListItem and OnAddComboItem events. If you use different event handlers you must ensure there are no logical inconsistencies in what you are filtering. For example, if something appears in the list, it must also appear in the tree and combo. You can safely filter anything from the list that is still present in the tree and combo. The only things you should really filter from the combo are first-level items such as My Computer or Network Neighbourhood.

OnFolderChanged event

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration

```
type TNotifyEvent = procedure(sender: TObject) of object;

property OnFolderChanged: TNotifyEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TNotifyEvent)(System::TObject* Sender);

__property TNotifyEvent OnFolderChanged;
```

Description
The OnFolderChanged event occurs when the user changes the directory that is displayed in the dialog. This can happen when the user double-clicks on a directory, clicks the *Up* button, or uses the combo box or tree view to navigate through the directory structure.

OnFormClose event

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg components

Delphi Declaration
```
type TNotifyEvent = procedure(sender: TObject) of object;

property OnFormClose: TNotifyEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TNotifyEvent)(System::TObject* Sender);

__property TNotifyEvent OnFormClose;
```

Description
Write an OnFormClose event handler to perform special processing when the dialog closes.

OnFormShow event

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg components

Delphi Declaration

```
type TNotifyEvent = procedure(sender: TObject) of object;

property OnFormShow: TNotifyEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TNotifyEvent)(System::TObject* Sender);

__property TNotifyEvent OnFormShow;
```

Description
Write an OnFormShow event handler to perform special processing when the dialog is displayed.

OnHelp event

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration
```
type THelpEvent = function (Command: Word; Data: Longint; var CallHelp: Boolean):
     Boolean of object;
property OnHelp: THelpEvent;
```

C++Builder Declaration
```
typedef bool __fastcall(__closure *THelpEvent)(unsigned short Command, int Data,
     bool& CallHelp );
__property THelpEvent OnHelp;
```

Description
Use OnHelp to write an event handler to perform special processing when help is requested. Set *CallHelp* to *false* if you don't want the standard help file invoked.

OnInitialized event

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg components

Delphi Declaration
```
type TNotifyEvent = procedure(sender: TObject) of object;

property OnInitialized: TNotifyEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TNotifyEvent)(System::TObject* Sender);

__property TNotifyEvent OnInitialized;
```

Description
This event is called after the form is created and the controls are filled, but before the form is shown.

OnLvCustomDrawSh and OnLvCustomDrawShEx events

Applies to
TPTOpenDlg, TPTSaveDlg components

Delphi Declaration

```
type TPTShLvCustomDrawEvent = procedure(
       aSender: TObject;
       aCD: TPTCustomDraw;
       aItem: TListItem;
       aData: TPTShListData ) of object;
property OnLvCustomDrawSh: TPTShLvCustomDrawEvent;
property OnLvCustomDrawShEx: TPTShLvCustomDrawEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TPTShLvCustomDrawEvent)(System::TObject* Sender,
TPTCustomDraw* aCD, TListItem* aItem, TPTShListData* aData);
__property TPTShLvCustomDrawEvent OnLvCustomDrawSh;
__property TPTShLvCustomDrawEvent OnLvCustomDrawShEx;
```

Description
You can change the appearance of individual items in the list by adjusting properties of the given *aCD* object.

Use the Font property of *aCD* to change the appearance of *aItem*.

Access shell-related data about the item with *aData*.

The OnLvCustomDrawSh event is only called for the *ptcdsItemPrePaint* draw stage, whereas OnLvCustomDrawShEx is called for every draw stage.

See TPTListView.OnPTCustomDraw for more information.

OnSelectionChanged event

Applies to
TPTOpenDlg, TPTSaveDlg

Delphi Declaration

```
type TNotifyEvent = procedure(sender: TObject) of object;

property OnSelectionChanged: TNotifyEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TNotifyEvent)(System::TObject* Sender);

__property TNotifyEvent OnSelectionChanged;
```

Description
The OnSelectionChange event occurs when the user does something to change the list displayed in the dialog. This can include opening the file-selection dialog, highlighting a file or directory, selecting a new filter, selecting a new directory, or creating a new folder.

OnTvCustomDrawSh and OnTvCustomDrawShEx events

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg components

Delphi Declaration

```
type TPTShTvCustomDrawEvent = procedure(
       aSender: TObject;
       aCD: TPTCustomDraw;
       aNode: TTreeNode;
       aData: TPTShTreeData ) of object;
property OnTvCustomDrawSh: TPTShTvCustomDrawEvent;
property OnTvCustomDrawShEx: TPTShTvCustomDrawEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TPTShTvCustomDrawEvent)(System::TObject* Sender,
TPTCustomDraw* aCD, Comctrls::TTreeNode* aNode, TPTShTreeData* aData);
__property TPTShTvCustomDrawEvent OnTvCustomDrawSh;
__property TPTShTvCustomDrawEvent OnTvCustomDrawShEx;
```

Description
You can change the appearance of individual nodes in the tree by adjusting properties of the given *aCD* object.

Use the Font and Brush properties of *aCD* to change the appearance of *aNode*.

Access shell-related data about the node with *aData*.

The OnTvCustomDrawSh event is only called for the *ptcdsItemPrePaint* draw stage, whereas OnTvCustomDrawShEx is called for every draw stage.

See TPTTreeView.OnPTCustomDraw for more information.

OnTypeChanged event

Applies to
TPTOpenDlg, TPTSaveDlg

Delphi Declaration
```
type TNotifyEvent = procedure(sender: TObject) of object;

property OnTypeChanged: TNotifyEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TNotifyEvent)(System::TObject* Sender);

__property TNotifyEvent OnTypeChanged;
```

Description
The OnTypeChanged event occurs when the user selects a new filter from the Files of Type combo box at the bottom of the dialog.

Execute method

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg components

Delphi Declaration
```
function Execute: Boolean;
```

C++Builder Declaration
```
bool __fastcall Execute(void);
```

Description
Creates and displays the form modally. Returns *true* if the user accepted the form, *false* if they cancelled.

ReadStateFromRegistry method

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg

Delphi Declaration
```
procedure ReadStateFromRegistry( aBaseKey: HKEY; aSubKeyName, aValueName: String );
```

C++Builder Declaration
```
void __fastcall ReadStateFromRegistry( HKEY aBaseKey, System::AnsiString
aSubKeyName, System::AnsiString aValueName );
```

Description
Reads FormWidth, FormHeight and FormWindowState properties from the given registry key item.

For TPTOpenDlg and TPTSaveDlg this method also reads FormSplitterPos and whether the tree view button was pressed or not.

ReadStateFromStream method

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg

Delphi Declaration
```
procedure ReadStateFromStream( aStream: TStream );
```

C++Builder Declaration
```
void __fastcall ReadStateFromStream( Classes::TStream* aStream );
```

Description
Reads the FormWidth, FormHeight and FormWindowState properties from the given registry key item.

For TPTOpenDlg and TPTSaveDlg this method also reads FormSplitterPos and whether the tree view button was pressed or not.

WriteStateToRegistry method

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg

Delphi Declaration
```
procedure WriteStateToRegistry( aBaseKey: HKEY; aSubKeyName, aValueName: String );
```

C++Builder Declaration
```
void __fastcall WriteStateToRegistry( HKEY aBaseKey, System::AnsiString aSubKeyName,
System::AnsiString aValueName );
```

Description
Writes FormWidth, FormHeight and FormWindowState properties to the given registry key item.

For TPTOpenDlg and TPTSaveDlg this method also stores FormSplitterPos and whether the tree view button was pressed or not.

WriteStateToStream method

Applies to
TPTOpenDlg, TPTSaveDlg, TPTFolderBrowseDlg

Delphi Declaration
```
procedure WriteStateToStream( aStream: TStream );
```

C++Builder Declaration
```
void __fastcall WriteStateToStream( Classes::TStream* aStream );
```

Description
Writes FormWidth, FormHeight and FormWindowState properties to the given stream.

For TPTOpenDlg and TPTSaveDlg this method also stores FormSplitterPos and whether the tree view button was pressed or not.

TPTSaveDlg component

Hierarchy Properties Methods Events

Delphi Unit
FPTOpenDlg

C++Builder Header
FPTOpenDlg.hpp

Description
The TPTSaveDlg component makes a Save dialog box available to your application. The purpose of the dialog box is to allow a user to specify a file to save. Use the Execute method to display the Save dialog box.

When the user chooses OK in the dialog box, the user's filename selection is stored in the dialog box's FileName property, which you can then use to process as you want.

You can let the user decide which set of files are visible in the list box of the Save dialog box with the Filter property. The user can then use the List Files of Type combo box to determine which files display in the list box. You set the default filter using the FilterIndex property.

You can permit the user to choose multiple filenames with the Options property so that the Files property contains a list of all the selected filenames in the list box. You can customise how the Save dialog box appears and behaves with the Options property.

If you want a file extension automatically appended to the filename typed in the File Name edit box of the Save dialog box, use the DefaultExt property.

Change the appearance of individual tree or list items with the OnLvCustomDrawSh and OnTvCustomDrawSh events.

You can test the component at design time from its right-click menu.

Hierarchy

TObject
|
TPersistent
|
TComponent
|
TPTDialog
|
TPTFileDlg
|
TPTCustomSaveDlg

TPTSaveDlg properties

TPTSaveDlg Legend

Properties in TPTSaveDlg


DefaultExt


Executing

FileName

Files


FilterIndex


Filter


Form


FormWidth


FormHeight


FormWindowState


FormSplitterPos


HistoryList


InitialDir


Options


Title

TPTSaveDlg methods

TPTSaveDlg Legend

Methods in TPTSaveDlg

- Execute
- ReadStateFromRegistry
- ReadStateFromStream
- WriteStateToRegistry
- WriteStateToStream

TPTSaveDlg events

TPTSaveDlg Legend

Events in TPTSaveDlg

OnAddListItem
OnAddTreeItem
OnAddComboItem
OnFolderChanged
OnFormShow
OnFormClose
OnHelp
OnInitialized
OnLvCustomDrawSh
OnLvCustomDrawShEx
OnSelectionChanged
OnTvCustomDrawSh
OnTvCustomDrawShEx
OnTypeChanged

FPTFOLDERBROWSEDLG UNIT

FPTFolderBrowseDlg unit

This unit contains the TPTFolderBrowseDlg component and the TPTFrmFolderBrowseDlg form. The component creates an instance of the form. You can visually inherit from TPTFrmFolderBrowseDlg to add or hide controls and change or add behaviour.

How to Visually Inherit

Components

- TPTFolderBrowseDlg

Forms

- TPTFrmFolderBrowseDlg

TPTFolderBrowseDlg component

Hierarchy Properties Methods Events

Delphi Unit
FPTFolderBrowseDlg

C++Builder Header
FPTFolderBrowseDlg.hpp

Description
The TPTFolderBrowseDlg makes an folder browse dialog box available to your application. The purpose of this dialog is to let the user specify a folder (use TPTOpenDlg or TPTSaveDlg to specify a file).

When the user selects OK, their selection is stored and can be retrieved with the SelectedPathName or SelectedFolder properties. These properties can also be used to set an initial selected folder before the dialog is executed.

You can filter items out of the tree by responding to the OnAddItem event.

You can change the caption of the dialog with the Title property.

You can set the initial dimensions of the dialog with the FormWidth, FormHeight and FormWindowState properties. After the user dismisses the dialog, these properties are adjusted to reflect any changes the user may have made. You can load and store these properties with the ReadStateFromRegistry, ReadStateFromStream, WriteStateToRegistry and WriteStateToStream methods.

Change the appearance of individual tree items with the OnTvCustomDrawSh events.

You can test the component at design time from its right-click menu.

Hierarchy

TObject
|
TPersistent
|
TComponent
|
TPTDialog
|
TPTFileDlg

TPTFolderBrowseDlg properties

TPTFolderBrowseDlg Legend

Properties in TPTFolderBrowseDlg

- BaseFolder
- Executing
- FormWidth
- FormHeight
- FormWindowState
- OkEnabled
- Options
- SelectedPathName
- SelectedFolder
- Status
- Title

TPTFolderBrowseDlg methods

TPTFolderBrowseDlg Legend

Methods in TPTFolderBrowseDlg

- Execute
- ReadStateFromRegistry
- ReadStateFromStream
- WriteStateToRegistry
- WriteStateToStream

TPTFolderBrowseDlg events

TPTFolderBrowseDlg Legend

Events in TPTFolderBrowseDlg

OnAddItem

OnFormShow

OnFormClose

OnInitialized

OnSelChanged

OnTvCustomDrawSh

OnTvCustomDrawShEx

BaseFolder property

Applies to
TPTFolderBrowseDlg

Declaration
```
property BaseFolder: TPTShellLocator;
```

C++Builder Declaration
```
__property TPTShellLocator* BaseFolder;
```

Description
Use this property to set the root node of the tree.

Example
This code sets the root node to be 'My Computer'.

Delphi:
```
PTFolderBrowseDlg1.BaseFolder.CSIDL := csidlDrives;
```

C++Builder:
```
PTFolderBrowseDlg1->BaseFolder->CSIDL = csidlDrives;
```

This code removes any base folder setting.

Delphi:
```
PTFolderBrowseDlg1.BaseFolder.Clear;
```

C++Builder:
```
PTFolderBrowseDlg1->BaseFolder->Clear();
```

Options property

Applies to
TPTFolderBrowseDlg

Delphi Declaration
```
type TPTFolderBrowseDlgOption = (ptfbCreateDeleteButtons, ptfbContextMenus,
        ptfbReadOnly, ptfbIncludeNonFolders, ptfbOleDrag, ptfbOleDrop,
        ptfbCreateFolderIcon, ptfbDeleteFolderIcon, ptfbVirtualFolders,
        ptfbShowHidden );
     TPTFolderBrowseDlgOptions = set of TPTFolderBrowseDlgOption;

property Options: TPTFolderBrowseDlgOptions;
```

C++Builder Declaration
```
enum TPTFolderBrowseDlgOption { ptfbCreateDeleteButtons, ptfbContextMenus,
        ptfbReadOnly, ptfbIncludeNonFolders, ptfbOleDrag, ptfbOleDrop,
        ptfbCreateFolderIcon, ptfbDeleteFolderIcon, ptfbVirtualFolders,
        ptfbShowHidden };

typedef Set<TPTFolderBrowseDlgOption, ptfbCreateDeleteButtons, ptfbVirtualFolders>
        TPTFolderBrowseDlgOptions;

__property TPTFolderBrowseDlgOptions Options;
```

Description
These are the possible values that can be included in the Options set for the Open and Save dialog boxes:

Identifier	Meaning
ptfbCreateDeleteButtons	When True, Create and Delete buttons are placed at the bottom of the form.
ptfbContextMenus	When True, right-click context menus are available in the tree control.
ptfbReadOnly	When True, you cannot rename items in the tree.
ptfbIncludeNonFolders	When True, non-folders are included in the tree. Such things as files, control panel applets, printers etc. will be available in the tree. Because of system limitations (explorer tree views weren't meant to work this way) all folders have a '+' icon, as there is no way to tell if there are any sub-items until the items are actually enumerated.
ptfbOleDrag	When True, the user can drag items out of the tree into other forms or applications.
ptfbOleDrop	When True, the user can drop items onto the tree.
ptfbCreateFolderIcon	When True, a small icon appears in the top-right of the form allowing the user to create a folder.
ptfbDeleteFolderIcon	When True, a small icon appears in the top-right of the form allowing the user to delete a folder.
ptfbVirtualFolders	When True, so-called *virtual* folders are also included in the list. Such things as the Control Panel and Printers folders appear in the list.
ptfbShowHidden	When *true*, hidden and system files and folders are included in the tree and list, otherwise they are excluded.

The default is [ptfbContextMenus, ptfbCreateFolderIcon, ptfbDeleteFolderIcon, ptfbShowHidden].

SelectedPathName property

Applies to
TPTFolderBrowseDlg

Delphi Declaration
```
property SelectedPathName: String;
```

C++Builder Declaration
```
__property System::AnsiString SelectedPathName;
```

Description
Run time only. This is an input or output property. You can set it to an initial pathname before calling Execute and the dialog will open with that pathname as the selected item. If the item is not found then a nearby folder is selected, no exception is raised in this case.

If Execute returns *true* you can read this property to see which folder the user selected. If the folder is not a filesystem folder (such as My Computer or Network Neighbourhood), then this property is an empty string.

SelectedFolder property

Applies to
TPTFolderBrowseDlg

Delphi Declaration
```
property SelectedFolder: TPTShellLocator;
```

C++Builder Declaration
```
__property TPTShellLocator SelectedFolder;
```

Description
Run time only. This is an input and output property. You can set it to an initial id list before calling Execute and the dialog will open with that folder as the selected item. If the item is not found then a nearby folder is selected, no exception is raised in this case.

If Execute returns *true* you can read this property to see which folder the user selected.

You do not need to free the returned item id list. If you need to remember the id list, you should use CopyIdList.

OkEnabled property

Applies to
TPTFolderBrowseDlg

Delphi Declaration
```
property OkEnabled: Boolean;
```

C++Builder Declaration
```
__property unsigned OkEnabled;
```

Description
Run time only. This property can be set via an event handler to enable and disable the OK button. You would typically use this property in conjunction with the OnSelChanged event to provide feedback as to which folders were acceptable.

Status property

Applies to
TPTFolderBrowseDlg

Delphi Declaration
```
property Status: String;
```

C++Builder Declaration
```
__property System::AnsiString Status;
```

Description
Run time only. This property can be set before the form is executed or via an event handler during executing to a line of text at the top of the form.

OnAddItem event

Applies to
TPTFolderBrowseDlg

Delphi Declaration
```
property OnAddItem: TPTShAddItemEvent;
```

C++Builder Declaration
```
__property TPTShAddItemEvent OnAddItem;
```

Description
This event is called every time an item is added to the tree view. You have the opportunity to filter out the unwanted items. You should respond to this event in the same way as you would respond to the OnAddItem event in TPTShellTree.

OnSelChanged event

Applies to
TPTFolderBrowseDlg

Delphi Declaration

```
type TPTFolderBrowseSelChangeEvent =
       procedure( aSender: TObject; aNewSel: PItemIdList )
       of object;

property OnSelChanged: TPTFolderBrowseSelChangeEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TPTFolderBrowseSelChangeEvent)(System::TObject*
aSender, Uptshell95::PItemIDList aNewSel);

__property TPTFolderBrowseSelChangeEvent OnSelChange;
```

Description
This event is called after the selected tree item has changed. You should not free *aNewSel*. If you need to remember the id list, you should use CopyIdList.

HOW TO VISUALLY INHERIT

How to Visually Inherit from Shell Control Pack Forms

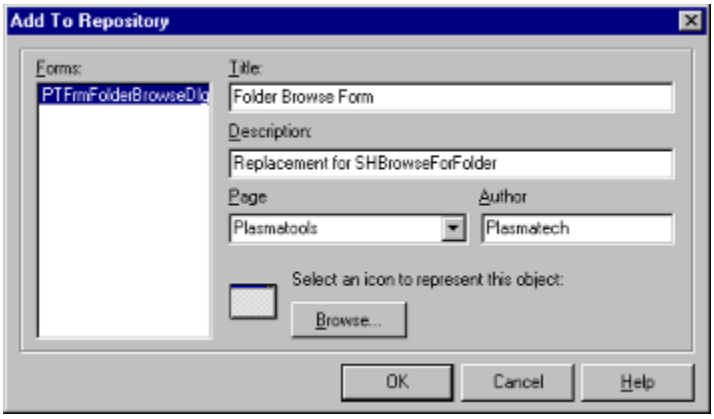
This example applies to Delphi 2, Delphi 3 and C++Builder.

Only the source code version of the Shell Control Pack supports visual inheritance.

You must first add the TPTFrmOpenDlg and TPTFrmFolderBrowseDlg forms to the repository. Once added to the repository you can Copy, Inherit or Use the forms as you would any repository form.

1. Add the TPTFrmOpenDlg and TPTFrmFolderBrowseDlg forms to the repository.

   - Ensure all projects are closed in the Delphi IDE - select File | Close All.
   - Locate the FPTFolderBrowseDlg.pas file and open it - File | Open.
   - Right-click on the form and select "Add to Repository…"



   - Fill out the dialog as above and click OK.

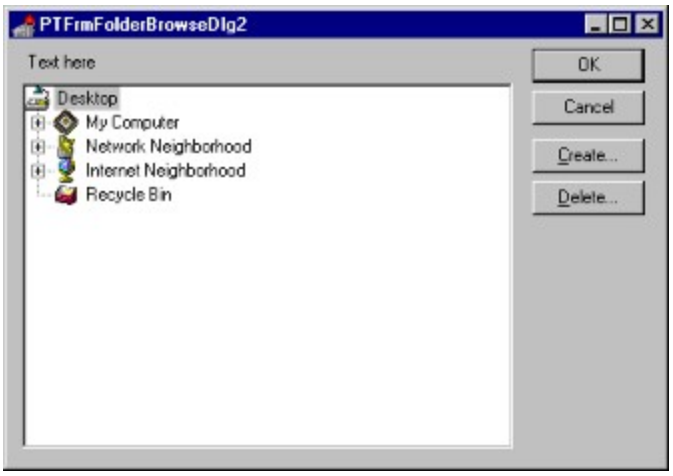
Example

A more comprehensive example can be found at the Plasmatech web site http://plasmatech.com.

Visual Form Inheritance Example

Right-aligning the buttons of the folder browse dialog.

1. Select File|New…
1. Select the "Plasmatools" tab.
1. Select the "Folder Browse Form".
1. Select the "Inherit" radio button.
1. Click OK.
1. Adjust the appearance of the new form:



Alternatively, you can edit the form as text and paste in the following:

```
inherited PTFrmFolderBrowseDlg2: TPTFrmFolderBrowseDlg2
     Left = 433
     Top = 302
     Caption = 'PTFrmFolderBrowseDlg2'
     ClientHeight = 283
     ClientWidth = 433
     PixelsPerInch = 96
     TextHeight = 13
     inherited PTShellTree1: TPTShellTree
       Width = 329
     end
     inherited ButtonPanel: TPanel
       Left = 336
       Top = 0
       Width = 97
       Height = 283
       Align = alRight
       inherited OkBtn: TButton
         Left = 10
       end
       inherited CancelBtn: TButton
         Left = 10
         Top = 34
       end
       inherited CreateBtn: TButton
         Left = 10
         Top = 66
       end
       inherited DeleteBtn: TButton
         Left = 10
         Top = 94
```

```
    end
  end
end
```

7. Add an event handler for the FormResize event and paste in the following code.

Delphi

```
procedure TPTFrmFolderBrowseDlg2.FormResize(Sender: TObject);
var r: TRect;
begin
// inherited; deliberately not called
  StatusTxt.Width := ClientWidth - ButtonPanel.Left;
  r := PTShellTree1.BoundsRect;
  PTShellTree1.BoundsRect := Rect(r.left, StatusTxt.BoundsRect.Bottom+8,
ButtonPanel.Left, ClientHeight-8);
end;
```

C++Builder

```
void TPTFrmFolderBrowseDlg2::FormResize( TObject* Sender )
{
  // inherited::FormResize(Sender);  deliberately not called
  StatusTxt->Width = ClientWidth - ButtonPanel->Left;
  Windows::TRect r = PTShellTree1->BoundsRect;
  r.Top = StatusTxt->BoundsRect.Bottom + 8;
  r.Right = ButtonPanel->Left;
  r.Bottom = ClientHeight-8;
  PTShellTree1.BoundsRect = r;
}
```

That's it!

You have now successfully used visual inheritance to create a new folder browse form.

HOW TO USE LEAK CHECKING

How to Use Leak Checking

Leak checking notifies you when your application terminates before all instances of the following classes are freed:

- TPTShTreeData
- TPTShListData
- TPTShComboData
- TPTImageComboItem

To enable leak checking for a whole project, define the symbol PTDEBUG in the conditionals section of the Project | Options dialog, Directories/Conditionals page.

Alternatively, you can enable checking in only the desired units. The following units implement leak checking:

- UPTShellControls
- UPTImageCombo

Search for the line that looks like this:

```
{ $DEFINE PTDEBUG}
```

and remove the space before the $ sign:

```
{$DEFINE PTDEBUG}
```

Note: Only the source code version of the Shell Control Pack supports leak checking.

UPTFRAME UNIT

UPTFrame unit

This unit implements two components, TPTGroup and TPTFrame. Both provide extended border styles. TPTGroup can act like a group box, containing other controls. TPTFrame is a lightweight non-windowed component that cannot contain other controls. If you don't need to contain children, then TPTFrame is faster, less bulky and provides more styles the TBevel. If you do need to contain controls, TPTGroup provides more border styles than TGroupbox.

The global procedures used to draw the frames are exposed, allowing you to add the same border styles to your own components.

Components

- TPTFrame
- TPTGroup

TPTGroup and TPTFrame components

Hierarchy Properties Methods Events

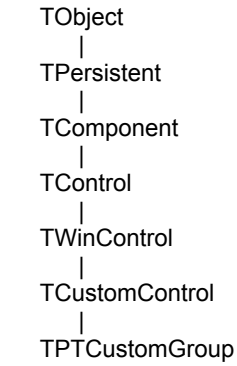
Delphi Unit
UPTFrame

C++Builder Header
UPTFrame.hpp

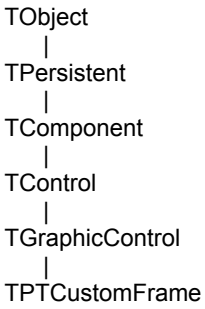
Description
TPTGroup is similar to the standard TGroupbox control, but provides many more border styles. Other features include; use of the ellipsis '…' when the caption extends outside of the bounds of the box, and Windows 95 standard disabled look..

TPTFrame has the same properties as TPTGroup but is not a windowed control and therefore uses fewer resources than TPTGroup. TPTFrame cannot own children. To see the difference, drop a component on a TPTGroup and move the TPTGroup around. The contained component moves with the group. The same operation performed with a TPTFrame and the contained control doesn't move.

TPTGroup Hierarchy



TPTFrame Hierarchy



TPTGroup, TPTFrame properties

TPTGroup, TPTFrame Legend

- Alignment
- DefaultDrawing
- FrameSpace
- FrameStyle

TPTGroup, TPTFrame methods

TPTGroup, TPTFrame Legend

EnableAllChildren

TPTGroup, TPTFrame events

TPTGroup, TPTFrame Legend

OnPaint

Alignment property

Applies to
TPTGroup, TPTFrame

Delphi Declaration
```
type TAlignment = (taLeftJustify, taCenter, taRightJustify);

property Alignment: TAlignment;
```

C++Builder Declaration
```
enum TAlignment { taLeftJustify, taRightJustify, taCenter };

__property TAlignment Alignment;
```

Description
Use this property to adjust the alignment of the caption for a group or frame control.

DefaultDrawing property

Applies to
TPTGroup, TPTFrame

Delphi Declaration
```
property DefaultDrawing: Boolean;
```

C++Builder Declaration
```
__property bool DefaultDrawing;
```

Description
Use this property to disable default drawing of the interior of the group of frame control. This also affects drawing done in the WM_ERASEBKGND message. The frame will be drawn normally regardless of the setting of this property.

This property is usually used in combination with the OnPaint event.

FrameSpace property

Applies to
TPTGroup, TPTFrame

Delphi Declaration
```
property FrameSpace: Integer;
```

C++Builder Declaration
```
__property int FrameSpace;
```

Description
Use this property to adjust the amount of space added to the border of the frame.

FrameStyle property

Applies to
TPTGroup, TPTFrame

Delphi Declaration
```
property FrameStyle: TPTFrameStyle;
```

C++Builder Declaration
```
__property TPTFrameStyle FrameStyle;
```

Description
Use this property to select one of the following frame styles.

TPTFrameStyle options

Value		Description
ptfsNone		No visible border.
ptfsSingle		Single black border.
ptfsRaised		Double raised border.
ptfsLowered		Double sunken border.
ptfsBump		Single raised border.
ptfsDint		Status line bevel.
ptfsGroup		Groupbox groove frame.
ptfsHorzLine		Horizontal groove.
ptfsHorzEdge		Double horizontal groove.
ptfsVertLine		Vertical groove.
ptfsVertEdge		Double vertical groove.

EnableAllChildren method

Applies to
TPTGroup

Delphi Declaration
```
procedure EnableAllChildren( afEnable: Boolean );
```

C++Builder Declaration
```
void __fastcall EnableAllChildren( bool afEnable );
```

Description
Use this method to enable or disable all the children of a group control. If *afEnable* is *true* the children are enabled otherwise they are disabled.

Many controls display themselves differently when disabled. It is common to require a whole group of controls to be disabled, and it is good interface design if all the controls appear disabled. With this method, you can easily enable or disable groups of related controls.

OnPaint event

Applies to
TPTGroup, TPTFrame

Delphi Declaration

```
type TPTFramePaintEvent = procedure( aSender: TObject;  aCanvas: TCanvas ) of
   object;

property OnPaint: TPTFramePaintEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TPTFramePaintEvent)(System::TObject* aSender,
Graphics::TCanvas* aCanvas);

__property TPTFramePaintEvent OnPaint;
```

Description
The OnPaint event is called after the frame is painted and after any default drawing is done. If the DefaultDrawing property is *false* then your event handler must draw then whole interior of the control.

UPTTREELIST UNIT

UPTTreeList unit

Components

- TPTTreeView
- TPTListView

Classes

- TPTCustomDraw
- TPTLvCustomDraw

Types

- TPTCustomDrawStage
- TPTLvHeaderSortDisplayMode

TPTTREEVIEW

TPTTreeView component

Hierarchy Properties Methods Events

Delphi Unit
UPTTreeList

C++Builder Header
UPTTreeList.hpp

Description
This component adds per-item custom drawing and per-item popup menu support to the standard TTreeView control.

With the OnPTCustomDraw and OnPTCustomDrawEx events you can modify the appearance of individual nodes.

Use the OnNodeContextMenu event to provide a menu for individual nodes.

Use the InvalidateNode method to repaint individual tree items with minimum flicker.

Limitations
The custom draw features provided by Microsoft with Internet Explorer 3 are not fully implemented. The following limitations have been verified using *comctl32.dll* version 4.70.

- Only the Color of the Brush property has any effect. Brush styles are ignored.
- The Rect property is uninitialised.
- If a font is selected that results in a change in the item height, view corruption and repaint problems will occur. Individual item heights are not selectable. The height of items is fixed and determined by the control's Font property.

Hierarchy

TObject
|
TPersistent
|
TComponent
|
TControl
|
TWinControl
|
TCustomTreeView

TPTTreeView properties

TPTTreeView Legend

Derived from TCustomTreeView


BorderStyle


DragMode

DropTarget


HideSelection


Images


Indent


Items


ReadOnly

Selected


ShowButtons


ShowLines


ShowRoot


SortType


StateImages

TopItem

TPTTreeView methods

TPTTreeView Legend

Derived from TPTCustomTreeView

InvalidateNode

Derived from TCustomTreeView

AlphaSort
Create
CustomSort
Destroy
FullCollapse
FullExpand
GetHitTestInfoAt
GetNodeAt
IsEditing
LoadFromFile
LoadFromStream
SaveToFile
SaveToStream

TPTTreeView events

TPTTreeView Legend

Derived from TPTCustomTreeView


OnPTCustomDraw


OnPTCustomDrawEx


OnNodeContextMenu

Derived from TCustomTreeView


OnChange


OnChanging


OnCollapsed


OnCollapsing


OnCompare


OnDeletion


OnEdited


OnEditing


OnExpanded


OnExpanding


OnGetImageIndex


OnGetSelectedIndex

InvalidateNode method

Applies to
TPTTreeView component

Delphi Declaration
```
procedure InvalidateNode( aNode: TTreeNode );
```

C++Builder Declaration
```
void __fastcall InvalidateNode( Comctrls::TTreeNode* aNode );
```

Description
This method tells Windows to repaint the portion of the control contained by *aNode* after other important Windows messages are handled. Appropriate use of this method can result in significantly reduced flicker when updating a tree where only a few nodes have changed.

Follow calls to this method with calls to Update if you want the repaint to happen immediately.

Equivalent functionality can be obtained in any tree view with the following code:

Delphi
```
var r: TRect;
...
r := aNode.DisplayRect(afTextOnly);
Windows.InvalidateRect( Handle, @r, true );
```

C++Builder
```
TRect r = aNode->DisplayRect(afTextOnly);
System::InvalidateRect( Handle, &r, true );
```

OnPTCustomDraw event

Applies to
TPTTreeView component

Delphi Declaration
```
type TPTTvCustomDrawEvent = procedure(
       aSender: TObject;
       aCD: TPTCustomDraw;
       aNode: TTreeNode ) of object;
property OnPTCustomDraw: TPTTvCustomDrawEvent;
```

C++Builder
```
typedef void __fastcall (__closure *TPTTvCustomDrawEvent)(System::TObject* aSender,
TPTCustomDraw* aCD, Comctrls::TTreeNode* aNode);

__property TPTTvCustomDrawEvent OnPTCustomDraw;
```

Description
Use the Font and Brush property of *aCD* to change the appearance of *aNode*.

This event is only called for the *ptcdsItemPrePaint* draw stage.

Example
This example sets an item bold, with red foreground and yellow background. The colours are not modified if the item is to be drawn in the selected style - doing so would just confuse the user.

Delphi
```
procedure TMyForm1.PTTreeView1PTCustomDraw( aSender: TObject; aCD: TPTCustomDraw;
aNode: TTreeNode );
begin
  if SomeFunction(aNode) then
  begin
    aCD.Font.Style := [fsBold];
    if not aNode.Selected and not aNode.DropTarget then
    begin
      aCD.Font.Color := clRed;
      aCD.Brush.Color := clYellow;
    end;
  end;
end;
```

C++Builder
```
void __fastcall TForm1::PTTreeView1PTCustomDraw(TObject *aSender, TPTCustomDraw
*aCD, Comctrls::TTreeNode *aNode)
{
  if ( SomeFunction(aNode) ) {
    aCD->Font->Style.Clear();
    aCD->Font->Style = aCD->Font->Style << fsBold;
    if ( ! aNode->Selected  &&  ! aNode.DropTarget ) {
      aCD->Font->Color = clRed;
      aCD->Brush->Color = clYellow;
    }
  }
}
```

Advanced Details
This event is only called for the *ptcdsItemPrePaint* draw stage. This is sufficient for most cases, allowing easy modification to font, foreground and background colour. If you need to process other draw stages, use the OnPTCustomDrawEx event instead.

OnPTCustomDrawEx event

Applies to
TPTTreeView component

Delphi Declaration
```
type TPTTvCustomDrawEvent = procedure(
       aSender: TObject;
       aCD: TPTCustomDraw;
       aNode: TTreeNode ) of object;
property OnPTCustomDrawEx: TPTTvCustomDrawEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTTvCustomDrawEvent)(System::TObject* aSender,
TPTCustomDraw* aCD, Comctrls::TTreeNode* aNode);

__property TPTTvCustomDrawEvent OnPTCustomDrawEx;
```

Description
Use the Font and Brush property of *aCD* to change the appearance of *aItem*.

This event is called for every draw stage.

Example
You can fairly easily emulate the OnPTCustomDraw event.

Delphi
```
procedure TForm1. PTTreeView1PTCustomDrawEx( aSender: TObject; aCD:
TPTCustomDraw; aNode: TTreeNode );
begin
  if aCD.DrawStage = ptcdsItemPrePaint then
    // processing
end;
```

C++Builder
```
void __fastcall TForm1::PTTreeView1PTCustomDrawEx(TObject *aSender,
  TPTCustomDraw *aCD, Comctrls::TTreeNode *aNode)
{
  if ( aCD->DrawStage == ptcdsItemPrePaint )
    // processing
}
```

However, there is a significant difference. The OnPTCustomDrawEx event is called for every custom-draw notification, whereas the OnPTCustomDraw event is called only once for each item. During a paint operation, Windows sends a number of custom draw notifications; 4 for the control as a whole and *4 for each item*, at least. Each of these notifications incurs some setup time when received. To paint 20 items, the OnPTCustomDrawEx event would be called (4 + 20*4)=84 times, whereas OnPTCustomDraw would be called only 20 times.

Having said that, the difference is barely noticeable on modern systems (Pentium 133 or better).

OnNodeContextMenu event

Applies to
TPTTreeView component

Delphi Declaration
```
type TPTTvNodeContextMenuEvent = procedure(
       aSender: TObject;
       aNode: TTreeNode;
       var pt: TPoint;
       var aMenu: TPopupMenu ) of object;
property OnNodeContextMenu: TPTTvNodeContextMenuEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTTvOnNodeContextMenuEvent)(System::TObject*
aSender, Comctrls::TTreeNode* aNode, tagPOINT &aPos, Menus::TPopupMenu* &aMenu);

__property TPTTvNodeContextMenuEvent OnNodeContextMenu;
```

Description
This event is invoked immediately before a popup menu is shown for the tree view (in response to a user's right-click, Shift+F10 or menu-key). *aNode* is the node for which the popup should be shown. If the PopupMenu property is assigned, then *aMenu* is initialised to that value. You can change it by assigning some other menu, leave it unchanged or assign it *nil* to prevent any menu processing. When this event returns, the *aMenu* parameter will be used as the popup menu.

You can handle the menu display and processing yourself (the demo program demonstrates this) in which case you should assign *aMenu* to *nil*.

You can also reposition the menu by modifying the *pt* parameter.

TPTLISTVIEW

TPTListView component

Hierarchy Properties Methods Events

Delphi Unit
UPTTreeList

C++Builder Header
UPTTreeList.hpp

Description
This component adds sort direction icon in the report-view header (show below), per-item custom drawing and per-item popup menu support.

Change where the sort arrow is placed with the HeaderSortDisplayMode property. Change which column has the sort arrow and the direction of the arrow with the HeaderSortColumn and HeaderSortDirection properties.

With the OnPTCustomDraw and OnPTCustomDrawEx events you can modify the appearance of individual items.

Use the OnItemContextMenu event to provide a menu for individual nodes.

Limitations
The custom draw features provided by Microsoft with Internet Explorer 3 are not fully implemented. The following limitations have been verified using *comctl32.dll* version 4.70.

- The Brush property is ignored, so there is no way to change the background colour of items short of drawing the whole item yourself.
- The Rect property is uninitialised.
- The Font.Color property has no effect when an item is selected.

Hierarchy

TObject
|
TPersistent
|
TComponent
|
TControl
|
TWinControl
|
TPTCustomListView

TPTListView properties

TPTListView Legend

Derived from TPTCustomListView


HeaderCanvas


HeaderDefaultDrawing


HeaderHandle

HeaderSortColumn

HeaderSortDirection


HeaderSortDisplayMode

Derived from TCustomListView


AllocBy


BorderStyle


BoundingRect


Checkboxes


Column


ColumnClick


Columns

DropTarget


GridLines


HideSelection


HotTrack


IconOptions

ItemFocused


Items


LargeImages


MultiSelect


ReadOnly


RowSelect


SelCount

Selected


ShowColumnHeaders

SmallImages

SortType

StateImages

TopItem

ViewOrigin

ViewStyle

VisibleRowCount

TPTListView methods

TPTListView Legend

Derived from TCustomListView

TPTListView events

TPTListView Legend

Derived from TPTCustomListView

OnPTCustomDraw

OnPTCustomDrawEx

OnItemContextMenu

HeaderCanvas property

Applies to
TPTListView

Delphi Declaration
```
property HeaderCanvas: TCanvas;
```

C++Builder Declaration
```
__property Graphics::TCanvas* HeaderCanvas;
```

Description
Returns a TCanvas which can be used for drawing on the listview header. This property is only valid during processing of the OnDrawHeader event (which is not implemented).

HeaderDefaultDrawing property

Applies to
TPTListView

Delphi Declaration
```
property HeaderDefaultDrawing: Boolean;
```

C++Builder Declaration
```
__property bool HeaderDefaultDrawing;
```

Description
Not implemented.

When implemented, you will be able to disable default drawing with this property and take total control of drawing the listview header.

HeaderHandle property

Applies to
TPTListView

Delphi Declaration
```
property HeaderHandle: HWND;
```

C++Builder Declaration
```
__property HWND HeaderHandle;
```

Description
Returns the window handle of the header control. If the header has not been created, or the control is in vsReport ViewStyle, the value will be 0.

HeaderSortColumn property

Applies to
TPTListView

Delphi Declaration
```
property HeaderSortColumn: Integer;
```

C++Builder Declaration
```
__property int HeaderSortColumn;
```

Description
Use this property to set which column will show the sort icon. Columns are counted with a 0-base, i.e. the first column is column 0.

Set *HeaderSortColumn* to –1 to hide the sort indicator. Alternatively, use the HeaderSortDirection property to hide the sort indicator.

This property does not cause the execution of any sorting. It is purely a visual property.

HeaderSortDirection property

Applies to
TPTListView

Delphi Declaration
```
type TPTLvSortDirection = (ptsdAsc, ptsdDesc);
property HeaderSortDirection: TPTLvSortDirection;
```

C++Builder Declaration
```
enum TPTLvSortDirection { ptsdAsc, ptsdDesc };
__property TPTLvSortDirection HeaderSortDirection;
```

Description
Use this property to change the direction of the sort icon.

HeaderSortDisplayMode property

Applies to
TPTListView

Delphi Declaration
```
property HeaderSortDisplayMode: TPTLvHeaderSortDisplayMode;
```

C++Builder Declaration
```
__property TPTLvHeaderSortDisplayMode HeaderSortDisplayMode;
```

Description
Use this property to change where the sort icon is placed relative to the text in the header. See the TPTLvHeaderSortDisplayMode help for options.

OnPTCustomDraw event

Applies to
TPTListView

Delphi Declaration
```
type TPTLvCustomDrawEvent = procedure(
       aSender: TObject;
       aCD: TPTCustomDraw;
       aItem: TListItem ) of object;
property OnPTCustomDraw: TPTLvCustomDrawEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTLvCustomDrawEvent)(System::TObject* aSender,
TPTCustomDraw* aCD, Comctrls::TListItem* aItem);

__property TPTLvCustomDrawEvent OnPTCustomDraw;
```

Description
Use the Font property of *aCD* to change the appearance of *aItem*.

Note that the Brush property has no effect with Internet Explorer 3 list view custom controls.

This event is only called for the *ptcdsItemPrePaint* draw stage.

OnPTCustomDrawEx event

Applies to
TPTListView

Delphi Declaration
```
type TPTLvCustomDrawEvent = procedure(
       aSender: TObject;
       aCD: TPTCustomDraw;
       aItem: TListItem ) of object;
property OnPTCustomDrawEx: TPTLvCustomDrawEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTLvCustomDrawEvent)(System::TObject* aSender,
TPTCustomDraw* aCD, Comctrls::TListItem* aItem);

__property TPTLvCustomDrawEvent OnPTCustomDrawEx;
```

Description
Use the Font property during the *ptcdsItemPrePaint* draw stage to change the appearance of *aItem*.

Note that the Brush property has no effect with the Internet Explorer 3 list view custom controls.

This event is called for every draw stage.

OnItemContextMenu event

Applies to
TPTListView

Delphi Declaration
```
type TPTLvOnItemContextMenuEvent = procecure(
       aSender: TObject;
       aItem: TListItem;
       var aPos: TPoint;
       var aMenu: TPopupMenu ) of object;
property OnItemContextMenu: TPTLvOnItemContextMenuEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTLvOnItemContextMenuEvent)(System::TObject*
aSender, Comctrls::TListItem* aItem, tagPOINT &aPos, Menus::TPopupMenu* &aMenu);

__property TPTLvOnItemContextMenuEvent OnItemContextMenu;
```

Description
This event is invoked immediately before a popup menu is shown for the list view (in response to a user's right-click, Shift+F10 or menu-key). *aItem* is the item for which the popup should be shown. If the PopupMenu property is assigned, then *aMenu* is initialised to that value. You can change it by assigning some other menu, leave it unchanged or assign it *nil* to prevent any menu processing. When this event returns, the *aMenu* paramter will be used as the popup menu.

You can handle the menu display and processing yourself in which case you should assign *aMenu* to *nil*.

You can also reposition the menu by modifying the *pt* parameter.

TPTCUSTOMDRAW

TPTCustomDraw class

Hierarchy Properties

Delphi Unit
UPTTreeList

C++Builder Header
UPTTreeList.hpp

Description
An instance of the TPTCustomDraw class is passed as a parameter to the custom-draw event handlers of TPTTreeView, TPTListView, TPTShellTree and TPTShellList.

You can modify the Font property to adjust the colour and appearance of any item text.

You can modify the Brush property to adjust the background of the item.

Notes
Custom draw is only available using the common controls installed by Internet Explorer 3 or later. If you use custom draw you must ensure that your customers either install Internet Explorer 3, or have already installed it.

Hierarchy

TObject

TPTCustomDraw properties

TPTCustomDraw Legend

Implemented in TPTCustomDraw

Brush
Canvas

DrawStage

Font

Handle

IsItem

NoDefaultDrawing
RawDrawStage

Rect

WantItems

Brush property

Applies to
TPTCustomDraw class

Delphi Declaration
```
property Brush: TBrush;
```

C++Builder Declaration
```
__property Graphics::TBrush* Brush;
```

Description
Change this property to affect the appearance of the background of items in the control. Not all controls support all the features of TBrush. Non-solid brushes and dithered colours are not supported with the Internet Explorer 3 controls.

DrawStage property

Applies to
TPTCustomDraw class

Delphi Declaration
```
property DrawStage: TPTCustomDrawStage;
```

C++Builder Declaration
```
__property TPTCustomDrawStage DrawStage;
```

Description
Use this property to determine the current stage of a custom draw sequence. The order of draw stages in a custom draw sequence are:

```
ptcdsPrePaint
ptcdsPreErase
   ptcdsItemPreErase
   ptcdsItemPostErase
   ptcdsItemPrePaint
   ptcdsItemPostPaint
ptcdsPostPaint
```

The 'item' states are repeated for each item that requires painting. *ptcdsItemPrePaint* is the most useful stage. At this stage you can change the Brush and Font property to change the appearance of the item. These properties have no effect at any other stage.

Font property

Applies to
TPTCustomDraw class

Delphi Declaration
```
property Font: TFont;
```

C++Builder Declaration
```
__property Graphics::TFont* Font;
```

Description
Change this property to affect the appearance of individual items in the control. Be wary in changing the height of the font. The current edition of the custom control (supplied with Internet Explorer e) are do not support such behaviour.

Handle property

Applies to
TPTCustomDraw class

Delphi Declaration
```
property Handle: HDC;
```

C++Builder Declaration
```
__property DC Handle;
```

Description
Read only. This property is rarely used.

Returns the handled to the device context given in the original notification message. This will be a device context suitable for drawing on the control. However, the current common controls (supplied with Internet Explorer 3) do not support such drawing. This device context is instead used to select a font, text and background colours which are used by the control itself when it comes to draw. This processing is handled automatically if you use the Brush and/or Font properties.

IsItem property

Applies to
TPTCustomDraw class

Delphi Declaration
```
property IsItem: Boolean;
```

C++Builder Declaration
```
__property bool IsItem;
```

Description
Read only. The property is *true* if the current draw stage indicates and item is being processed.

RawDrawStage property

Applies to
TPTCustomDraw class

Delphi Declaration
```
property RawDrawStage: DWORD;
```

C++Builder Declaration
```
__property unsigned RawDrawStage;
```

Description
Read only. This property is rarely used. It holds the raw bit-field passed to the control in the NM_CUSTOMDRAW notification structure.

WantItems property

Applies to
TPTCustomDraw class

Delphi Declaration
```
property WantItems: Boolean;
```

C++Builder Declaration
```
__property bool WantItems;
```

Description
During the *ptcdsPrePaint* draw stage you must set this property to True to receive any 'item' draw states. If you leave WantItems *false* during *ptcdsPrePaint*, then no custom draw messages will be sent, received or processed.

This property is only of use in OnPTCustomDrawEx events. The OnPTCustomDraw events are only invoked during *ptcdsItemPrePaint* stage, so WantItems is automatically set *true* internally.

Canvas property

Applies to
TPTCustomDraw class

Delphi Declaration
```
property Canvas: TCanvas;
```

C++Builder Declaration
```
__property Graphics::TCanvas* Canvas;
```

Description
Use this property to draw to the control's canvas. Use the Rect property to determine the allowable bounds of any drawing.

NoDefaultDrawing property

Applies to
TPTCustomDraw class

Delphi Declaration
```
property NoDefaultDrawing: Boolean;
```

C++Builder Declaration
```
__property bool NoDefaultDrawing;
```

Description
Write *true* to this property during the *ptcdsItemPrePaint* draw stage to prevent any default drawing of the item.

Rect property

Applies to
TPTCustomDraw class

Delphi Declaration
```
property Rect: TRect;
```

C++Builder Declaration
```
__property Windows::TRect Rect;
```

Description
Read only. Return the bounding rectangle of the item to be drawn, in units suitable for drawing using the Canvas property.

NOTE: This property is provided for future support only. It does not work with the common controls installed by Internet Explorer 3.

TPTCustomDrawStage type

Unit
UPTTreeList

Delphi Declaration
```
type TPTCustomDrawStage = ( ptcdsUnknown, ptcdsPrePaint, ptcdsPostPaint,
        ptcdsPreErase, ptcdsItemPrePaint, ptcdsItemPostPaint, ptcdsItemPreErase,
        ptcdsItemPostErase );
```

C++Builder Declaration
```
enum TPTCustomDrawStage { ptcdsUnknown, ptcdsPrePaint, ptcdsPostPaint,
ptcdsPreErase, ptcdsItemPrePaint, ptcdsItemPostPaint, ptcdsItemPreErase,
ptcdsItemPostErase };
```

Description
This type enumerates the different stages of a custom-draw sequence. Use the DrawStage property to access this information.

TPTLvHeaderSortDisplayMode type

Unit
UPTTreeList

Delphi Declaration
```
type TPTLvHeaderSortDisplayMode = (ptlvNone, ptlvLeftAlign, ptlvRightOfText,
ptlvRightAlign);
```

C++Builer Declaration
```
enum TPTLVHeaderSortDisplayMode { ptlvNone, ptlvLeftAlign, ptlvRightOfText,
ptlvRightAlign };
```

Description
Tyis type enumerates the different placements of the TPTListView control's report-view header sort icon.

TPTLvCustomDraw class

Delphi Unit
UPTTreeList

C++Builder Header
UPTTreeList.hpp

Description
This class is instantiated internally by TPTListView. You should never create an instance of this class directly. At the public level it is exactly the same as TPTCustomDraw. You can treat instances of this class as if they were TPTCustomDraw.

UPTSHELLCONTROLS UNIT

UPTShellControls unit

The UPTShellControls unit contains the components, classes and types for the shell controls.

Components

- TPTShellTree
- TPTShellList
- TPTShellCombo

Classes

- TPTShTreeData
- TPTShListData
- TPTShComboData

Routines

- PTShCreateNewFolder

Types

- TPTShellTreeOptions
- TPTShellListOptions
- TPTShellComboOptions

Constants

- Other constants

PTShCreateNewFolder function

Unit
UPTShellControls

Delphi Declaration
```
function PTShCreateNewFolder( aPathname: String;  var aNewName: String ): Boolean;
```

C++Builder Declaration
```
extern bool __fastcall PTShCreateNewFolder( System::AnsiString aPathname,
System::AnsiString& aNewName );
```

Description
Attempts to create a new folder with the name "New Folder". If that folder already exists, an attempt is made to create "New Folder (2)". This procedure continues until a new folder is successfully created, an error other than folder exists occurs, or PTSH_MAX_FOLDER_ATTEMPTS failures have occurred.

TPTShellTreeOptions, TPTShellListOptions, TPTShellComboOptions types

Delphi Declaration

```
type TPTShellTreeOption = ( ptstoAutoFill, ptstoVirtualFolders,
       ptstoDesignInteractive, ptstoDefaultKeyHandling, ptstoContextMenus,
       ptstoDynamicRefresh, ptstoOleDrag, ptstoOleDrop, ptstoShowHidden );
     TPTShellTreeOptions = set of TPTShellTreeOption;

type TPTShellListOption = ( ptsloAutoFill, ptsloNonFilesystemAncestors,
 ptsloDesignInteractive, ptsloDefaultKeyHandling,  ptsloContextMenus,
ptsloDontChangeFolder, ptsloDontGoBelowBase, ptsloDynamicRefresh,
ptsloHideFoldersWhenLinkedToTree, ptsloOleDrag, ptsloOleDrop, ptsloFolderContextMenu,
ptsloShowHidden);
     TPTShellListOptions = set of TPTShellListOption;

type TPTShellComboOption = ( ptscoAutofill, ptscoNonFilesystemAncestors );
     TPTShellComboOptions = set of TPTShellComboOption;
```

C++Builder Declaration

```
enum TPTShellTreeOption { ptstoAutoFill, ptstoVirtualFolders,
ptstoDesignInteractive, ptstoDefaultKeyHandling, ptstoContextMenus,
ptstoDynamicRefresh, ptstoIncludeNonFolders, ptstoOleDrag, ptstoOleDrop,
ptstoShowHidden };

typedef Set<TPTShellTreeOption, ptstoAutoFill, ptstoOleDrop> TPTShellTreeOptions;

enum TPTShellListOption { ptsloAutoFill, ptsloNonFilesystemAncestors,
ptsloDesignInteractive, ptsloDefaultKeyHandling, ptsloContextMenus,
ptsloDontChangeFolder, ptsloDontGoBelowBase, ptsloDynamicRefresh,
ptsloHideFoldersWhenLinkedToTree, ptsloOldDrag, ptsloOleDrop,
ptsloFolderContextMenu, ptsloShowHidden };

typedef Set<TPTShellListOption, ptsloAutoFill, ptsloFolderContextMenu>
TPTShellListOptions;

enum TPTShellComboOption { ptscoAutofill, ptscoNonFilesystemAncestors };

typedef Set<TPTShellComboOption, ptscoAutofill, ptscoNonFilesystemAncestors>
TPTShellComboOptions;
```

Description

The shell control options type is a set of the options each control supports. See help for the particular control's Options property for more information:

- TPTShellTree.Options
- TPTShellList.Options
- TPTShellCombo.Options

Other Constants

Unit
UPTShellControls

Delphi Declaration
```
var PTSH_CHANGE_NOTIFY_DELAY: Integer = 2000;
    PTSH_CHANGE_NOTIFY_FASTDELAY: Integer = 500;
    PTSH_TREE_KEY_UPDATE_DELAY: Integer = 500;
    PTSH_MAX_FOLDER_ATTEMPTS: Integer = 50;
    PTSH_AUTOSCROLL_THRESHOLD_X: Integer = 26;
    PTSH_AUTOSCROLL_THRESHOLD_Y: Integer = 20;
    PTSH_AUTOSCROLL_MINDELAY_MS: Integer = 100;
```

C++Builder Declaration
```
extern int PTSH_CHANGE_NOTIFY_DELAY;     // Default 2000
extern int PTSH_CHANGE_NOTIFY_FASTDELAY; // Default 500
extern int PTSH_TREE_KEY_UPDATE_DELAY;   // Default 500
extern int PTSH_MAX_FOLDER_ATTEMPTS;     // Default 50
extern int PTSH_AUTOSCROLL_THRESHOLD_X;  // Default 26
extern int PTSH_AUTOSCROLL_THRESHOLD_Y;  // Default 20
extern int PTSH_AUTOSCROLL_MINDELAY_MS;  // Default 100
```

Constant	Meaning
PTSH_CHANGE_NOTIFY_DELAY	Milliseconds delay between when a change is detected and when the shell tree or list is updated.
PTSH_CHANGE_NOTIFY_FASTDELAY	Milliseconds delay between when a change is detected and when the shell tree or list is updated when the control is focused.
PTSH_TREE_KEY_UPDATE_DELAY	Milliseconds delay between a keyboard-based move of the tree selection and when any associated shell list or combo is updated.
PTSH_MAX_FOLDER_ATTEMPTS	Maximum number of times to try creating a new folder.
PTSH_AUTOSCROLL_THRESHOLD_X	Number of pixels from the left or right edge of the control before autoscrolling starts.
PTSH_AUTOSCROLL_THRESHOLD_Y	Number of pixels from the top or bottom edge of the control before autoscrolling starts.
PTSH_AUTOSCROLL_MINDELAY_MS	Milliseconds minimum delay between autoscrolls during OLE drag/drop in a shell tree or list control.

You can assign different values to these system variables.

TPTSHELLTREE COMPONENT

TPTShellTree component

Hierarchy Properties Methods Events

Delphi Unit
UPTShellControls

C++Builder Header
UPTShellControls.hpp

Description
Implements a tree-view of the shell name space.

You can control what sorts of items appear in the tree with the Options property or manually with the OnAddItem event.

The control can operate interactively at design time by setting the *ptstoDesignInteractive* option in the Options property to True.

Use the SelectedFolder or SelectedPathName properties to retrieve and set the currently selected folder.

You can limit the part of the namespace shown in the try by setting the BaseFolder property. When set, BaseFolder determines which node will appear as the root of the tree. *Do not* use the BaseFolder property when the tree is linked to a TPTShellCombo control. The TPTShellCombo control does not currently support basing.

You can link a TPTShellList control to the TPTShellTree component by setting the ShellList property. Thereafter the tree and list will automatically update each other as the user interacts with them. You can also link a TPTShellCombo with a TPTShellTree.

Hierarchy

TObject
|
TPersistent
|
TComponent
|
TControl
|
TWinControl
|
TCustomTreeView
|
TPTCustomTreeView

TPTShellTree properties

TPTShellTree Legend

Derived from TPTCustomShellTree

- BaseFolder
- Options
- SelectedItem
- SelectedPathName
- SelectedFolder
- ShellList

- ShTreeData

TPTShellTree methods

TPTShellTree Legend

Derived from TPTCustomShellTree

- DoCommandForNode
- CreateNewFolder
- FillItems
- FindNodeWithIdList
- GetDataFromNode
- GoUp
- RefreshNodes
- SortNode
- Synchronize

Derived from TPTCustomTreeView

- InvalidateNode

TPTShellTree events

TPTShellTree Legend

Derived from TPTCustomShellTree

OnAddItem

OnDeleteItem

OnFillComplete

OnFillStart

OnInsertItem

OnPopupHint

BaseFolder property

Applies to
TPTShellTree

Delphi Declaration
```
property BaseFolder: TPTShellLocator;
```

C++Builder Declaration
```
__property TPTShellLocator BaseFolder;
```

Description
Use this property to set the root node of the tree.

Options property

Applies to
TPTShellTree

Delphi Declaration
```
property Options: TPTShellTreeOptions;
```

C++Builder Declaration
```
__property TPTShellTreeOptions Options;
```

Description
These are the possible values that can be included in the Options set for the shell tree control.

Value	Meaning
ptstoAutoFill	When this option is set, the contents of the tree are automatically initialised.
ptstoVirtualFolders	When true, non-file system nodes such as Control Panel and Printers are included in the tree. Default is True.
ptstoDesignInteractive	This option only has effect at design time. When *true*, you can interact with the shell list control at design time. You can select items, navigate into folders etc. If you set this option *true*, some of Delphi's design time keyboard interface might not work.
ptstoDefaultKeyHandling	Enables default keyboard processing. This includes such keys as Ctrl+C (Copy), Ctrl+V (Paste) and F5 (Refresh).
ptstoContextMenus	Enables context menu processing. When set a right-click on a selected item or group of items will display a popup-menu.
ptstoDynamicRefresh	When *true* the tree monitors changes to all local drives (and networks drives with a mapped drive letter). When a change to a folder is detected on any drive the tree is updated to reflect the change, if that folder is visible.
ptstoIncludeNonFolders	When *true* the tree includes non-folders as nodes. Such things as files, control panel applets, printers etc. will be available in the tree. Because of system limitations (explorer tree views weren't meant to work this way) all folders have a '+' icon, as there is no way to tell if there are any sub-items until the items are actually enumerated.
ptstoOleDrag	When *true*, the user can drag items out of the tree onto other trees or lists or into other applications.
ptstoOleDrop	When *true*, the user can drop items onto tree nodes.
ptstoShowHidden	When *true*, hidden and system files and folders are included in the tree, otherwise they are excluded.

ShellList property

See Also

Applies to
TPTShellTree

Delphi Declaration
```
property ShellList: TPTShellList;
```

C++Builder Declaration
```
__property TPTShellList ShellList;
```

Description
You can set the *ShellList* property to any TPTShellList control available on the same form. When set, the list and tree are synchronised.

See Also

TPTShellList component

SelectedItem property

Applies to
TPTShellTree

Delphi Declaration
```
property SelectedItem: TPTShTreeData;
```

C++Builder Declaration
```
__property TPTShTreeData SelectedItem;
```

Description
Run-time and read only. Returns the data object describing the currently selected node in the tree. If there is no currently selected node, the property is *nil*.

This property is a convenient equivalent to either of the following expression:

Delphi:
```
PTShellTree1.ShTreeData[ PTShellTree1.Selected.Index ]
```

C++Builder:
```
PTShellTree1->ShTreeData->Item[ PTShellTree1->Selected->Index ]
```

or

Delphi:
```
PTShellTree1.GetDataFromNode( PTShellTree1.Selected )
```

C++Builder:
```
PTShellTree1->GetDataFromNode( PTShellTree1->Selected )
```

SelectedPathName property

Applies to
TPTShellTree

Delphi Declaration
```
property SelectedPathName: String;
```

C++Builder Declaration
```
__property System::AnsiString SelectedPathName;
```

Description
Run time only. This property is the pathname of the selected item. If the item is not part of the filesystem, this property will be an empty string.

When assigning to *SelectedPathName*, if the folder is not found an exception will be raised.

This property is a shortcut for the following expression:

Delphi:
```
PTShellTree1.SelectedFolder.PathName := myPathname;
```

C++Builder:
```
PTShellTree1->SelectedFolder->PathName = myPathname;
```

SelectedFolder property

Applies to
TPTShellTree

Delphi Declaration
```
property SelectedFolder: TPTShellLocator;
```

C++Builder Declaration
```
__property TPTShellLocator* SelectedFolder;
```

Description
Run time only. This property allows selection and retrieval of the selected shell folder using the convenient TPTShellLocator class.

If you don't care about item id lists, you can use the SelectedPathName property instead.

Example
To retrieve the selected folder as an item id list:

Delphi:
```
myIdList := PTShellTree1.SelectedFolder.IdList;
```

C++Builder:
```
myIdList = PTShellTree1->SelectedFolder->IdList;
```

ShTreeData property

Applies to
TPTShellTree

Delphi Declaration
```
property ShTreeData[ idx: Integer ]: TPTShTreeData;
```

C++Builder Declaration
```
__property TPTShTreeData* ShTreeData[ int idx ];
```

Description
Run time and read only.

This property is simply a convenient way of accessing the TPTShTreeData object for a given tree node. The node is specified by index.

Equivalent code is:

Delphi:
```
data := TObject(PTShellTree1.Items[ idx ].Data) as TPTShTreeData;
```

C++Builder:
```
TPTShTreeData* data = dynamic_cast<TPTShTreeData*>(PTShellTree1->Items[idx]->Data);
```

It's much more convenient to express it this way:

Delphi:
```
data := PTShellTree1.ShTreeData[ idx ];
```

C++Builder:
```
TPTShTreeData* data = PTShellTree1->ShTreeData[idx];
```

DoCommandForNode method

Applies to
TPTShellTree

Delphi Declaration
```
procedure DoCommandForNode( aNode: TTreeNode;  aCmd: PChar );
```

C++Builder Declaration
```
void __fastcall DoCommandForNode( Comctrls::TTreeNode* aNode, char* aCmd );
```

Description
Perform a string-based command for the given tree node. For more information see Folder command strings.

CreateNewFolder method

Applies to
TPTShellTree

Delphi Declaration
```
function CreateNewFolder( afEditNow: Boolean ): Boolean;
```

C++Builder Declaration
```
bool __fastcall CreateNewFolder( bool afEditNow )
```

Description
If possible, creates a new child folder of the currently selected tree node with the name "New Folder".

If *afEditNow* is *true* then the new tree node immediately enters edit mode.

Returns *true* if the new folder was created successfully, otherwise *false*.

FillItems method

Applies to
TPTShellTree

Delphi Declaration
```
procedure FillItems;
```

C++Builder Declaration
```
void __fastcall FillItems(void);
```

Description
FillItems adds nodes to the tree. You must call FillItems at least once for the tree to function. If ptstoAutoFill is in the Options property then FillItems is automatically called once before the tree becomes visible.

You can call FillItems at any time to have all the nodes in the tree cleared and re-added. All open nodes will be collapsed after a call to FillItems. It is generally preferable to call RefreshNodes to update an already filled tree to reflect changes.

FindNodeWithIdList method

Applies to
TPTShellTree

Delphi Declaration
```
function FindNodeWithIdList( baseNode: TTreeNode; idList: PItemIdList ): TTreeNode;
```

C++Builder Declaration
```
Comctrls::TTreeNode* __fastcall FindNodeWithIdList( Comctrls::TTreeNode* baseNode,
Uptshell95::PItemIDList idList );
```

Description
Searches the children of *baseNode* for an element with a relative item id list of *idList*. If found, the tree node is returned otherwise *nil* is returned.

SortNode method

Applies to
TPTShellTree

Delphi Declaration
```
procedure SortNode( aNode: TTreeNode );
```

C++Builder Declaration
```
void __fastcall SortNode( Comctrls::TTreeNode* aNode );
```

Description
Sorts the given node according to standard shell sorting order.

If *aNode* is *nil* then the whole tree is sorted.

GetDataFromNode method

Applies to
TPTShellTree

Delphi Declaration
```
function GetDataFromNode( aNode: TTreeNode ): TPTShTreeData;
```

C++Builder Declaration
```
TPTShTreeData* __fastcall GetDataFromNode( Comctrls::TTreeNode* aNode );
```

Description
Returns the TPTShTreeData object associated with the given tree node. This is simply a convenient wrapper around the following code:

Delphi:	`result := TObject(node.Data) as TPTShTreeData;`
C++Builder:	`return dynamic_cast<TPTShTreeData*>(node->Data);`

GoUp method

Applies to
TPTShellTree

Delphi Declaration
```
procedure GoUp( aLevels: Integer );
```

C++Builder Declaration
```
void __fastcall GoUp( int aLevels );
```

Description
Calling this method moves the selected item back *aLevels* in the hierarchy.

aLevels must be greater than or equal to 1.

You can navigate to the top of the namespace with the following:

Delphi:
```
PTShellTree1.SelectedItem.IdList := nil;
```

C++Builder:
```
PTShellTree1->SelectedItem->IdList = NULL;
```

RefreshNodes method

Applies to
TPTShellTree

Delphi Declaration
```
procedure RefreshNodes;
```

C++Builder Declaration
```
void __fastcall RefreshNodes(void);
```

Description
This method works on an already populated tree. Every node in the tree is checked to see if it still exists (if it doesn't it is removed from the tree) or if it has new children (added to the tree). This is the preferred method to update the tree because it does not unnecessarily collapse any tree nodes. Use this method in preference to FillItems which does collapse the tree.

Synchronize method

Applies to
TPTShellTree, TPTShellList, TPTShellCombo

Delphi Declaration
```
procedure Synchronize( afApplyToGroup: Boolean );
```

C++Builder Declaration
```
void __fastcall Synchronize( bool afApplyToGroup );
```

Description
Calling this method causes any pending updates to be immediately processed.

If *afApplyToGroup* is *false* then this method only synchronises the control for which it is called. If *afApplyToGroup* is *true* then all associated controls are synchronised.

The following situations cause pending updates:

- Using the cursor keys to move around a tree. The update of associated list and combo controls is delayed.
- Detecting file system changes with dynamic refresh options. The update is delayed to improve performance, effectively batching changes until no additional changes have been made for the allotted time.

OnAddItem event

Example

Applies to
TPTShellTree

Delphi Declaration

```
TPTShAddItemEvent = procedure(
    aSender: TObject;
    aParentIShf: IShellFolder;     // Interface to the parent folder
    aParentAbsIdList: PItemIdList; // Absolute pidl referencing the parent
    aItemRelIdList: PItemIdList;   // Relative pidl of this item
    aAttribs: DWORD;               // IShellFolder::GetAttributesOf flags
    var aAllowAdd: Bool ) of object;
property OnAddItem: TPTShAddItemEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TPTShAddItemEvent)(System::TObject* aSender,
Uptshell95::IShellFolder* aParentIShf, Uptshell95::PItemIDList aParentAbsIdList,
Uptshell95::PItemIDList aItemRelIdList, int aAttribs, unsigned &afAllowAdd);
__property TPTShAddItemEvent* OnAddItem;
```

Description
This event is called for each item added to the tree. Your handler can reject an item by setting the *aAllowAdd* parameter to *false*.

TPTShellTree.OnAddItem Example

This example uses the OnAddItem event to remove the Network Neighbourhood folder.

Delphi

```
var g_netHoodIdList: PItemIdList;  // This variable assumed previously initialised

function IsNetHood( absIdList: PItemIdList ): Boolean;
begin
  result := (UPTShellUtils.ShellCompareAbsIdLists( absIdList, g_netHoodIdList ) =
0);
end;

procedure TMyForm.OnAddItem( aSender: TObject;
                             aParentIShf: IShellFolder;
                             aParentAbsIdList: PItemIdList;
                             aItemRelIdList: PItemIdList;
                             aAttribs: DWORD;
                             var aAllowAdd: Bool );
var itemAbsIdList: PItemIdList;
begin
  itemAbsIdList := nil;
  try
    itemAbsIdList := ConcatIdLists( aParentAbsIdList, aItemRelIdList );
    afAllowAdd := not IsNetHood(itemAbsIdList);
  finally
    ShellMemFree(itemAbsIdList);
  end;
end;
```

C++Builder

```
PItemIdList g_netHoodIdList; // This variable assumed previously initialised

bool __fastcall IsNetHood( PItemIdList absIdList )
{
  return (Uptshellutils::ShellCompareAbsidList( absIdList, g_netHoodIdList )==0);
}

void __fastcall TMyForm::PTShellTree1AddItem(TObject *aSender,
 IShellFolder *aParentIShf, PItemIDList aParentAbsIdList,
 PItemIDList aItemRelIdList, int aAttribs, LongBool &afAllowAdd)
{
  PItemIdList itemAbsIdList = NULL;
  try {
    itemAbsIdList = Uptshellutils::CopyIdList( NULL, aParentAbsIdList );
    Uptshellutils::ConcatIdLists( itemAbsIdList, aItemRelIdList );
    afAllowAdd = ( !IsNetHood(itemAbsIdList) );
  }
  catch(...)
  {
    Uptshellutils::ShellMemFree( itemAbsIdList );
    throw;
  }
  Uptshellutils::ShellMemFree( itemAbsIdList );
}
```

OnDeleteItem event

Applies to
TPTShellTree

Delphi Declaration
```
type TPTShTreeDeleteItemEvent =
       procedure( aSender: TObject;
                  aNode: TTreeNode;
                  aShTreeData: TPTShTreeData ) of object;
property OnDeleteItem: TPTShTreeDeleteItemEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTShTreeDeleteItemEvent)(System::TObject*
aSender, Comctrls::TTreeNode* aNode, TPTShTreeData* aShTreeData);
__property TPTShTreeDeleteItemEvent* OnDeleteItem;
```

Description
This event is called immediately before each node of the tree is deleted. If you have stored any data with the node, this is your last chance to delete it.

Example
Assuming you stored an object of your own with tree node:

Delphi: `PTShellTree1.ShTreeData[idx].Data := TMyData.Create;`

C++Builder: `PTShellTree1->ShTreeData->Item[idx]->Data = new TMyData;`

When the *idx* node is deleted, you should respond:

Delphi
```
procedure TMyForm.PTShellTree1DeleteItem( aSender: TObject;
  aNode: TTreeNode; aShTreeData: TPTShTreeData );
begin
  TObject(aShTreeData.Data).Free;
end;
```

C++Builder
```
void __fastcall TForm1::PTShellTree1DeleteItem(TObject *aSender,
  TTreeNode *aNode, TPTShTreeData *aShTreeData)
{
  delete (TObject*)(aShTreeData->Data);
}
```

OnCustomDrawSh and OnCustomDrawShEx events

Applies to
TPTShellTree

Delphi Declaration

```
type TPTShTvCustomDrawEvent =
       procedure( aSender: TObject;
                  aCD: TPTCustomDraw;
                  aNode: TTreeNode;
                  aData: TPTShTreeData ) of object;

property OnCustomDrawSh: TPTShTvCustomDrawEvent;

property OnCustomDrawShEx: TPTShTvCustomDrawEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TPTShTvCustomDrawEvent)(System::TObject*
aSender, Upttreelist:: TPTCustomDraw* aCD, Comctrls::TTreeNode* aNode,
TPTShTreeData* aData);

__property TPTShTvCustomDrawEvent* OnCustomDrawSh;

__property TPTShTvCustomDrawEvent* OnCustomDrawShEx;
```

Description
You can change the appearance of individual nodes in the tree by adjusting properties of the given *aCD* object.

Use the Font and Brush properties of *aCD* to change the appearance of *aNode*.

Access shell-related data about the node with *aData*.

The OnTvCustomDrawSh event is only called for the *ptcdsItemPrePaint* draw stage, whereas OnTvCustomDrawShEx is called for every draw stage.

OnFillComplete event

See Also

Applies to
TPTShellTree, TPTShellList

Delphi Declaration

```
TNotifyEvent = procedure( aSender: TObject ) of object;
property OnFillComplete: TNotifyEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TNotifyEvent)(System::TObject* aSender);
__property TNotifyEvent OnFillComplete;
```

Description
This event is called after the items in the control have been added (see TPTShellList.FillItems, TPTShellTree.FillItems, TPTShellList.RefreshItems and TPTShellTree.RefreshNodes). For TPTShellTree controls, this includes when a node is expanded for the first time. For TPTShellList controls this is normally only when the control changes to view another folder.

Prior to version 1.3g, this event was *not* called after the controls were refreshed (see TPTShellTree.RefreshNodes and TPTShellList.RefreshItems).

See Also

OnFillStart
FillItems
RefreshItems
RefreshNodes

OnFillStart event

See Also

Applies to
TPTShellTree, TPTShellList

Delphi Declaration

```
TNotifyEvent = procedure( aSender: TObject ) of object;
property OnFillStart: TNotifyEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TNotifyEvent)(System::TObject* aSender);
__property TNotifyEvent OnFillStart;
```

Description
This event is called before a fill or refresh operation commences. For TPTShellTree controls, this includes when a node is expanded for the first time. For TPTShellList controls this is normally only when the control changes to view another folder.

See Also

OnFillComplete
FillItems
RefreshItems
RefreshNodes

OnInsertItem event

Applies to
TPTShellTree

Delphi Declaration
```
type TPTShTreeInsertItemEvent = procedure( aSender: TObject; aNode: TTreeNode ) of
   object;

property OnInsertItem: TPTShTreeInsertItemEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTShTreeInsertItemEvent)(System::TObject*
   aSender, Comctrls::TTreeNode* aNode);

__property TPTShTreeInsertEvent OnInsertItem;
```

Description
This event is called after each node is added to the tree.

OnPopupHint event

Applies to
TPTShellTree, TPTShellList

Delphi Declaration
```
TPTShPopupHintEvent = procedure( aSender: TObject; const hint: String ) of object;
property OnPopupHint: TPTShPopupHintEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTShPopupHintEvent)(System::TObject* aSender,
const System::AnsiString hint);
__property TPTShPopupHintEvent OnPopupHint;
```

Description
Hint text is generated as the user navigates context menus for tree and list items. As each hint is produced, this event is called.

A typical OnPopupHint event handler would simply assign the hint string to the Application.Hint property. eg.:

Delphi
```
procedure TMyForm.TreeOnPopupHint( aSender: TObject;
const hint: String );
begin
  Application.Hint := hint;
end;
```

C++Builder
```
void __fastcall TMyForm::TreeOnPopupHint( TObject* aSender, const AnsiString hint )
{
  Application->Hint = hint;
}
```

TPTShTreeData class

Unit
UPTShellControls

Description
This class handles data for each node of the TPTShellTree component. The TPTShellTree component stores one of these objects with each tree node in the TTreeNode.Data property. This means you should *never* write to the Data property of a TTreeNode in a TPTShellTree. You can however read that Data property, cast it to a TPTShTreeData class and then use the following properties to access information about the node.

You do not need to free any of the item IDs returned by these properties.

Delphi Declaration **C++Builder Declaration**

```
property AbsoluteIdList: PItemIdList;     __property PitemIdList AbsoluteIdList;
```

Read only. Returns the item id list for the tree node relative to the desktop. This particular id list is used in many other functions.

You do not need to free the returned id list. If you want to remember it (store it in a non-local variable) you must copy it — use the CopyIdList function for this purpose.

```
property Data: Pointer;                   __property void* Data;
```

This property allows the user of the tree to store extra data with each node. If you store a heap allocated structure or object with this pointer, you must ensure that it is freed in all instances by responding to the TPTShellTree.OnDeleteItem event.

Example
This example associates a new instance of a user-defined object with the currently selected node.

Delphi:
```
ShellTree1.ShTreeData[0].Data := TMyExtraData.Create;
```
C++Builder:
```
ShellTree1->ShTreeData->Item[0]->Data = new TMyExtraData;
```

```
property Editable: Boolean;               __property bool Editable;
```

Read only. This is not the same as the read-only file attribute. If the property is *true* then this node is potentially editable. Read-only files are still 'editable'. Examples of nodes that are never editable are Desktop, My Computer, Network Neighborhood etc.

```
property ParentIShf: IShellFolder;        __property Uptshell95::IShellFolder* ParentIShf;
```

Read only. The IShellFolder interface for the parent of the tree node. For the desktop node, this property is nil.

```
property ParentIdList: PItemIdList;       __property PitemIdList ParentIdList;
```

Read only. Returns the absolute item id list of the parent folder of the tree node.

You do not need to free the returned id list. If you want to remember it (store it in a non-local variable) you must copy it — use the CopyIdList function for this purpose.

```
property PathName: String;                __property System::AnsiString* PathName;
```

Read only. Returns the pathname of the associated tree node. If the node has no associated pathname (such as My Computer or Network Neighbourhood) then this property is an empty string.

```
property RelativeIdList: PItemIdList;     __property PItemIdList RelativeIdList;
```

Read only. This property is the item id list for this tree node relative to its parent.

Many useful functions can only be performed in a item via its parent. To this end, you can use a combination of *ParentIShf* and *RelativePidl* to perform IShellFolder operations on this item.

Example
This example obtains some of the attribute flags for the current item and does an action if the item is a file system folder, or an ancestor of the filesystem (such as Desktop and My Computer).
Delphi

```
var nodeData: TPTShTreeData; // assume initialised
   ...
```

```
dwAttr := SFGAO_FILESYSTEM or SFGAO_FILESYSANCESTOR;
nodeData.ParentIShf.GetAttributesOf( 1, nodeData.RelativeIdList, dwAttr );
if ((dwAttr and (SFGAO_FILESYSTEM or SFGAO_FILESYSANCESTOR))<>0 then
  DoAction;
```

C++Builder

```
TPTShTreeData* nodeData; // assume initialised
  ...
unsigned dwAttr = SFGAO_FILESYSTEM | SFGAO_FILESYSANCESTOR;
nodeData->ParentIShf->GetAttributesOf( 1, nodeData->RelativeIdList, dwAttr );
if (dwAttr & (SFGAO_FILESYSTEM | SFGAO_FILESYSANCESTOR))
  DoAction();
```

```
property ThisIShf: IShellFolder;          __property Uptshell95::IShellFolder* ThisIShf;
```

Read only. This is the IShellFolder interface for the current item.

TPTIDLISTARRAY CLASS

TPTIdListArray class

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Description
The TPTIdListArray class provides a convenient mechanism for treating an item id list as an array of individual item ids.

Delphi Declaration
```
constructor Create( p: PItemIdList );
```
C++Builder Declaration
```
__fastcall TPTIdListArray( PItemIdList p );
```
Use this constructor to create an instance of TPTIdListArray for a given id list.

Delphi Declaration
```
function GoUp( items: Integer ): PitemIdList;
```
C++Builder Declaration
```
PItemIdList GoUp( int items );
```
Removes *items* ids from then end of the list and returns the new list. You don't need to free the returned list. Subsequent calls to GoUp or Item[ ] invalidate previous results from this function.

Delphi Declaration
```
property ItemCount: Integer;
```
C++Builder Declaration
```
__property int ItemCount;
```
Read only. Returns the number of ids in the list.

Delphi Declaration
```
property Item[idx: Integer]: PitemIdList; default;
```
C++Builder Declaration
```
__property PItemIdList Item[int idx];
```
Read only. A copy of the given id is allocated from shell memory and returned. You don't have to free it. If you want to keep it you should use CopyIdList to make a copy. Each call to Item invalidates the previous return value.

Example
This example takes a given id list and calls a method for each id in the list.

Delphi
```
procedure ProcessIdlist( idlist: PItemIdList );
var idla: TPTIdListArray;
 i: Integer;
begin
  idla := TPTIdListArray.Create( idlist );
  try
    for i := 0 to idla.ItemCount-1 do
      ProcessIndiviualId( idla[i] );
  finally
    idla.Free;
  end;
end;
```

C++Builder
```
#include <memory>  // for STL auto_ptr
...
void __fastcall ProcessIdList( PItemIDList idlist )
{
  std::auto_ptr <TPTIdListArray> idla( new TPTIdListArray(idlist) );
  for( int i=0, i < idla.get()->ItemCount; i++ )
    ProcessIndividualId( idla.get()->Item[i] );
}
```

TPTSHELLLOCATOR CLASS

TPTShellLocator class

Delphi Unit
UPTShellControls

C++Builder Header
UPTShellControls.hpp

Description
The TPTShellLocator class provides an easy to use encapsulation of the different ways a shell folder can be addressed.

There are three ways a shell folder can be specified. By item id list, pathname or system folder constant (CSIDL). This class performs automatic conversion between the different types, through its properties. You can use the TPTShellLocator class in much the same way as Variant variables are used.

For example, if you write a pathname, you can read an id list. If you write a CSIDL you can read a pathname. The only caveat is if you write an id list or pathname, you cannot meaningfully read CSIDL (it will return *csidlNone*).

Since TPTShellLocator class is derived from TPersistent it can be used as a published property of components.

Properties
- IdList
- PathName
- CSIDL

Methods
- Clear
- IsEqual

Events
- OnChange

IdList property

Applies to
TPTShellLocator

Delphi Declaration
```
property IdList: PItemIdList;
```

C++Builder Declaration
```
__property PitemIdList IdList;
```

Description
Returns the item id list of the folder addressed by the TPTShellLocator object.

PathName property

Applies to
TPTShellLocator

Delphi Declaration
```
property PathName: String;
```

C++Builder Declaration
```
__property System::AnsiString* PathName;
```

Description
Returns the pathname of the folder addressed by the TPTShellLocator object. If the folder is not part of the file system an empty string is returned.

CSIDL property

Applies to
TPTShellLocator

Delphi Declaration
```
property CSIDL: TCSIDL;
```

C++Builder Declaration
```
__property TCSIDL CSIDL;
```

Description
When assigned sets the TPTShellLocator object to the specified system folder constant.

When read, the value depends on whether the CSIDL property was used to assign a system folder constant, or the IdList or PathName properties were used. If CSIDL was used then the returned value is the same as the assigned value. If IdList or PathName was used then *csidlNone* is returned.

Clear method

Applies to
TPTShellLocator

Delphi Declaration
```
procedure Clear;
```

C++Builder Declaration
```
void Clear(void);
```

Description
Clears the folder addressed by the TPTShellLocator class. When cleared, the IdList property returns *nil*, the PathName property returns an empty string and the CSIDL property returns *csidlNone*.

IsEqual method

Applies to
TPTShellLocator

Delphi Declaration
```
function IsEqual( aToThis: TPTShellLocator ): Boolean;
```

C++Builder Declaration
```
bool IsEqual( TPTShellLocator* aToThis );
```

Description
Returns True if the passed in object refers to the same namespace location as this object.

OnChange property

Applies to
TPTShellLocator

Delphi Declaration
```
property OnChange: TNotifyEvent;
```

C++Builder Declaration
```
__property TnotifyEvent OnChange;
```

Description
The OnChange event specifies which event handler should execute when the contents of the TPTShellLocator class changes.

TPTSHELLLIST COMPONENT

TPTShellList component

Hierarchy Properties Methods Events

Delphi Unit
UPTShellControls

C++Builder Header
UPTShellControls.hpp

Description
Implements the list-view of the contents of a shell folder.

Use the Folder property to get and set the current folder. For single selection lists, use the SelectedItem property to retrieve the selected item data.

Use the FileFilter property and / or the OnAddItem event to specify which files should be included in the list.

Use the OnCustomDrawSh or OnCustomDrawShEx events to change the appearance of individual list items.

The control can act independently or automatically together with TPTShellTree or TPTShellCombo controls. To link to a TPTShellTree component set the ShellList property of the tree to the desired TPTShellList component. The components will then automatically work together and present a consistent interface. To link to a TPTShellCombo set the ShellList property of the combo.

Hierarchy

TObject
|
TPersistent
|
TComponent
|
TControl
|
TWinControl
|
TCustomListView
|
TPTCustomListView
|
TPTCustomShellList

TPTShellList properties

TPTShellList Legend

Derived from TPTCustomShellList


FileFilter


Folder


Options

SelectedItem

SortColumn


ShListData

Derived from TPTCustomListView


HeaderCanvas

HeaderDefaultDrawing


HeaderHandle

HeaderSortColumn

HeaderSortDirection


HeaderSortDisplayMode

TPTShellList methods

TPTShellList Legend

Derived from TPTCustomShellList

- CreateNewFolder
- DoCommandForAllSelected
- DoCommandForFolder
- DoCommandForItem
- FillItems
- GetDataFromItem
- GoUp
- OpenItem
- OpenSelectedItems
- ProcessMenu
- ProcessMenuForAllSelected
- RefreshItems
- SelectAll
- Synchronize

TPTShellList events

TPTShellList Legend

Derived from TPTCustomShellList


OnAddItem


OnDblClickOpen


OnDeleteItem


OnCustomDrawSh


OnCustomDrawShEx


OnPopupHint


OnFillComplete


OnFolderChanged

FileFilter property

See Also

Applies to
TPTShellList

Delphi Declaration
```
property FileFilter: String;
```

C++Builder Declaration
```
__property System::AnsiString* FileFilter;
```

Description
The Filter property specifies the file masks used in determining which files are displayed in the shell list control. A file mask or file filter is a file name that usually includes wildcard characters (*.PAS, for example). Only files that match the selected file filter are displayed in the list. To specify a file filter, assign a filter string as the value of FileFilter.

Multiple filters can be used by applying semicolon (;) delimiters.

Example
To show only image files, you might use the following filter:

Delphi:
```
PTShellList1.FileFilter := '*.bmp;*.gif;*.jpg';
```

C++Builder:
```
PTShellList1->FileFilter = "*.bmp;*.gif;*.jpg";
```

See Also

OnAddItem event

Folder property

See Also

Applies to
TPTShellList

Delphi Declaration
```
property Folder: TPTShellLocator;
```

C++Builder Declaration
```
__property TPTShellLocator Folder;
```

Description
This property identifies the current folder for the shell list control. Set this property to change the current folder, or read it to determine the current folder.

Examples

1. Set the current folder to a pathname.

   Delphi:
   ```
   PTShellList1.Folder.PathName := 'C:\';
   ```

   C++Builder:
   ```
   PTShellList1->Folder->PathName = "C:\";
   ```

2.. Set the current folder to the control panel folder.

   Delphi:
   ```
   PTShellList1.Folder.CSIDL := csidlControls;
   ```

   C++Builder:
   ```
   PTShellList1->Folder->CSIDL = csidlControls;
   ```

3. Display the current folder.

   Delphi:
   ```
   ShowMessage( 'The current folder is '+PTShellList1.Folder.PathName );
   ```

   C++Builder:
   ```
   ShowMessage( AnsiString("The current folder is") +
                PTShellList1->Folder->PathName );
   ```

See Also

TPTShellLocator class
System folder constants

Options property

Applies to
TPTShellList

Delphi Declaration
```
property Options: TPTShellListOptions;
```

C++Builder Declaration
```
__property TPTShellListOptions Options;
```

Description
These are the possible values that can be included in the Options set for the shell list control.

Value	Meaning
ptsloAutoFill	When this option is set, the contents of the list are automatically initialised.
ptsloNonFilesystemAncestors	When *false*, items which are not part of the filesystem, and not parents of filesystem folders are automatically excluded from the list. Candidates for exclusion are Control Panel, Printers and Recycle bin for example. My Computer will not be excluded since it is the ancestor of filesystem items.
ptsloDesignInteractive	This option only has effect at design time. When *true*, you can interact with the shell list control at design time. You can select items, navigate into folders etc. If you set this option *true*, some of Delphi's design time keyboard interface might not work.
ptsloDefaultKeyHandling	Enables default keyboard processing. This includes such keys as Backspace, Enter, Ctrl+A (Select All), Ctrl+C (Copy) and Ctrl+X (Cut).
ptsloContextMenus	Enables context menu processing. When set a right-click on a selected item or group of items will display a popup-menu.
ptsloDontChangeFolder	When set, navigation is disabled. You cannot go into a folder, or go back to a parent folder.
ptsloDontGoBelowBase	Only valid when the shell list control is used by itself - not linked to a shell combo and/or shell tree. Prevents the user navigating up past the folder specified in the BaseFolder property.
ptsloDynamicRefresh	When *true* the list monitors changes to items in it's folder. When a change is detected the list is refreshed (a deliberate delay of 2 seconds is imposed between the detection of a change and the update of the list).
ptsloHideFoldersWhenLinkedToTree	When this option is set *true* and the list control is linked to a shell tree control, file system folders are not shown in the list (like the way the open dialogs in Windows 3.1 worked). Obviously using this feature will make your interface non-standard, but some have argued why waste space showing the folders in the list when they are already visible in the tree?
ptsloOleDrag	When *true*, the user can drag items out of the list onto other trees or lists or into other applications.
ptsloOleDrop	When *true*, the user can drop items onto list item.
ptsloFolderContextMenu	When this option is true, and the PopupMenu property is unassigned a default menu similar to Explorer's "no items are selected" menu is shown. This gives access to *New Document* functionality. *Not implemented yet 7 Mar 98.*
ptsloShowHidden	When *true*, hidden and system files and folders are included in the

list, otherwise they are excluded.

SelectedItem property

Applies to
TPTShellList

Delphi Declaration
```
property SelectedItem: TPTShListData;
```

C++Builder Declaration
```
__property TPTShListData SelectedItem;
```

Description
Run-time and read only. Returns the data object describing the currently selected item in a single selection list. If there is no currently selected item, the property is *nil*.

This property is a convenient equivalent to either of the following to expressions:

Delphi: `PTShellList1.ShListData[ PTShellList1.Selected.Index ]`

C++Builder: `PTShellList1->ShListData[ PTShellList1->Selected->Index ]`

or

Delphi: `PTShellList1.GetDataFromItem( PTShellList1.Selected )`

C++Builder: `PTShellList1->GetDataFromItem( PTShellList1->Selected )`

SortColumn property

Applies to
TPTShellList

Delphi Declaration
```
property SortColumn: Integer;
```

C++Builder Declaration
```
__property int SortColumn;
```

Description
Run-time only. One based index of the column in details view currently used to sort the items.

Negative values indicate a reverse sort order, so a value of -1 would mean the first column in reverse order for example.

Zero (0) is used to indicate a custom sort order. When SortColumn = 0, the OnCompare event is called to determine the sort order. You can use the GetDataFromItem method to convert the TListItem parameters into TPTShListData objects.

ShListData property

Applies to
TPTShellList

Delphi Declaration
```
property ShListData[ idx: Integer ]: TPTShListData;
```

C++Builder Declaration
```
__property TPTShListData ShListData[ int idx ];
```

Description
Run time and read only.

This property is simply a convenient way of accessing the TPTShListData object for a given list item. The item is specified by index.

Equivalent code is:

Delphi:
```
data := TObject(PTShellList1.Items[ idx ].Data) as TPTShListData;
```

C++Builder:
```
TPTShListData* data = dynamic_cast<TPTShListData*>(PTShellList1->Items[idx]->Data);
```

It is much more convenient this way:

Delphi:
```
data := PTShellList1.ShListData[ idx ];
```

C++Builder:
```
TPTShListData* data = PTShellList1->ShListData[ idx ];
```

Example

1. Get the path name for the currently focused item.

Delphi:
```
mystr := PTShellList1.ShListData[ PTShellList1.ItemFocused.Index ].PathName;
```

C++Builder:
```
AnsiString mystr = PTShellList->ShListData[ PTShellList1->ItemFocused->Index ]->Pa
```

2. Get the file names of all the selected items.

Delphi
```
var i: Integer;
    sl: TStringList;
...
for i := 0 to PTShellList1.Items.Count-1 do
begin
  if PTShellList1.Items[i].Selected then
    with PTShellList1.ShListData[i] do
      if FileName <> '' then sl.Add( FileName  );
end;
```

C++Builder
```
#include <memory>  // For STL auto_ptr
...
std::auto_ptr<TStringList> sl( new TStringList );
for( int i=0; i < PTShellList1->Items->Count; i++ ) {
  if (PTShellList1->Items->Item[i]->Selected) {
    TPTShListData* data = PTShellList1->ShListData[i];
    if ( data->FileName.IsEmpty() )
      sl.get()->Add( data->FileName );
  }
}
```

CreateNewFolder method

Applies to
TPTShellList

Delphi Declaration
```
function CreateNewFolder( afEditNow: Boolean ): Boolean;
```

C++Builder Declaration
```
bool __fastcall CreateNewFolder( bool afEditNow );
```

Description
If possible, creates a new child folder of the currently selected tree node with the name "New Folder".

If *afEditNow* is *true* then the new tree node immediately enters edit mode.

Returns *true* if the new folder was created successfully otherwise *false*.

DoCommandForAllSelected method

Applies to
TPTShellList

Delphi Declaration
```
procedure DoCommandForAllSelected( aCmd: PChar );
```

C++Builder Declaration
```
void __fastcall DoCommandForAllSelected( char* aCmd );
```

Description
Perform a string-based command for all the selected items.

For more information see Folder command strings.

DoCommandForFolder method

Applies to
TPTShellList

Delphi Declaration
```
procedure DoCommandForFolder( cmd: PChar );
```

C++Builder Declaration
```
void __fastcall DoCommandForFolder( char* cmd );
```

Description
Perform a string-based command for the current folder. The current folder is the parent of the items in the list control.

For more information see Folder command strings.

DoCommandForItem method

Applies to
TPTShellList

Delphi Declaration
```
procedure DoCommandForItem( aItem: TListItem;  cmd: PChar );
```

C++Builder Declaration
```
void __fastcall DoCommandForItem( TListItem* aItem, char* cmd );
```

Description
Perform a string-based command for the given list item.

For more information see Folder command strings.

FillItems method

Applies to
TPTShellList

Delphi Declaration
```
procedure FillItems;
```

C++Builder Declaration
```
void __fastcall FillItems(void);
```

Description
FillItems clears the list and then adds items. You must call FillItems at least once for the list to function. If *ptsloAutoFill* is in the Options property then FillItems is automatically called once before the list becomes visible.

You can call FillItems at any time to have all the items in the list cleared and re-added. It is generally preferable to call RefreshNodes to update an already filled list to reflect changes.

GoUp method

Applies to
TPTShellList

Delphi Declaration
```
procedure GoUp( aLevels: Integer );
```

C++Builder Declaration
```
void __fastcall GoUp( int aLevels );
```

Description
Calling this method moves the list's Folder property up *aLevels* in the hierarchy.

aLevels must be greater than or equal to 1.

You can navigate to the top of the namespace with the following:

Delphi:
```
PTShellList1.Folder.IdList := nil;
```

C++Builder:
```
PTShellList1->Folder->IdList = NULL;
```

GetDataFromItem method

Applies to
TPTShellList

Delphi Declaration
```
function GetDataFromItem( aItem: TListItem ): TPTShListData;
```

C++Builder Declaration
```
TPTShListData* __fastcall GetDataFromItem( TListItem* aItem );
```

Description
Returns the TPTShListData object associated with the given list item. This is simply a convenient wrapper around the following code:

Delphi:
```
result := TObject(item.Data) as TPTShListData;
```

C++Builder:
```
return dynamic_cast<TPTShListData*>((TObject*)(item.Data));
```

OpenItem method

Applies to
TPTShellList

Delphi Declaration
```
procedure OpenItem( aNode: TListItem );
```

C++Builder Declaration
```
void __fastcall OpenItem( TListItem* aNode );
```

Description
If aNode is a folder then it is assigned to the list's Folder property and the list refilled. If aNode is not a folder then the default menu item action is performed on that item.

OpenSelectedItems method

Applies to
TPTShellList

Delphi Declaration
```
procedure OpenSelectedItems;
```

C++Builder Declaration
```
void __fastcall OpenSelectedItems(void);
```

Description
Performs the default menu item action on all the selected items. If no items are selected, nothing happens.

ProcessMenu method

Applies to
TPTShellList

Delphi Declaration
```
procedure ProcessMenu( aItem: TListItem;  at: TPoint );
```

C++Builder Declaration
```
void __fastcall ProcessMenu( TListItem* aItem,  const TPoint& at );
```

Description
Creates, displays and processes hints and selection for the context menu of the given item. The menu is placed left aligned at the screen coordinate specified by the *at* parameter.

ProcessMenuForAllSelected method

See Also

Applies to
TPTShellList

Delphi Declaration
```
procedure ProcessMenuForAllSelected( at: TPoint );
```

C++Builder Declaration
```
void __fastcall ProcessMenuForAllSelected( const TPoint& at );
```

Description
Creates, displays and processes the context menu for all the currently selected items. *at* is the top left pixel coordinate of the popup menu in screen coordinates.

See Also

ProcessMenu method

RefreshItems method

Applies to
TPTShellList

Delphi Declaration
```
procedure RefreshItems;
```

C++Builder Declaration
```
void __fastcall RefreshItems(void);
```

Description
This method works on an already populated list. Every item in the list is checked to see if it still exists (if it doesn't it is removed from the list) or if it new items are present (added to the list). This is the preferred method to update the list because it does not unnecessarily refresh any items or erase the user's selection state. Use this method in preference to FillItems which does erase the selection state.

SelectAll method

Applies to
TPTShellList

Delphi Declaration
```
procedure SelectAll;
```

C++Builder Declaration
```
void __fastcall SelectAll(void);
```

Description
Selects all the list items.

OnAddItem event

Applies to
TPTShellList

Delphi Declaration
```
TPTShAddItemEvent = procedure(
    aSender: TObject;
    aParentIShf: IShellFolder;     // Interface to the parent folder
    aParentAbsIdList: PItemIdList; // Absolute pidl referencing the parent
    aItemRelIdList: PItemIdList;   // Relative pidl of this item
    aAttribs: DWORD;               // IShellFolder::GetAttributesOf flags
    var aAllowAdd: Bool ) of object;

property OnAddItem: TPTShAddItemEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTShAddItemEvent)(System::TObject* aSender,
    Uptshell95::IShellFolder* aParentIShf, Uptshell95::PItemIDList aParentAbsIdList,
    Uptshell95::PItemIDList aItemRelIdList, int aAttribs, unsigned &afAllowAdd);

__property TPTShAddItemEvent OnAddItem;
```

Description
This event is called for each item added to the tree. Your handler can reject an item by setting the *aAllowAdd* parameter to *false*.

See the example for TPTShellTree.OnAddItem for more information.

OnDblClickOpen event

Applies to
TPTShellList

Delphi Declaration
```
type TPTShDblClickOpenEvent = procedure( aSender: TObject;
        var afHandled: Boolean ) of object;

procedure OnDblClickOpen: TPTShDblClickOpenEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTShDblClickOpenEvent)(System::TObject*
    aSender, bool &afHandled );

__property TPTShDblClickOpenEvent OnDblClickOpen;
```

Description
This event is called just before double-click (or Enter key) processing is performed on a list item or items. It is not called when the user double-clicks a folder or a link to a folder.

The default behaviour when a non-folder, non-link to folder item is double-clicked is to perform the default menu item action on that item (edit for .txt files, execute for .exe files etc.). By responding to this event you can change that behaviour.

For example, the TPTOpenDlg and TPTSaveDlg components use this event to make double-click select the item and end the dialog.

To prevent the default action, set *afHandled* to *true*. To allow the default action, leave *afHandled* unchanged or assign it to *false*.

OnDeleteItem event

Applies to
TPTShellList

Delphi Declaration
```
type TPTShListDeleteItemEvent =
       procedure( aSender: TObject;
                  aItem: TListItem;
                  aData: TPTShListData ) of object;
property OnDeleteItem: TPTShListDeleteItemEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTShListDeleteItemEvent)(System::TObject*
    aSender, Comctrls::TListItem* aNode, TPTShListData* aShListData);
__property TPTShListDeleteItemEvent OnDeleteItem;
```

Description
This event is called immediately before each item of the list is deleted. If you have stored any data with the item, this is your last chance to delete it.

Example
Assuming you stored an object of your own with a list item:

Delphi:
```
PTShellList1.ShListData[idx].Data := TMyData.Create;
```

C++Builder:
```
PTShellList1->ShListData[idx]->Data = new TMyData;
```

When the *idx* node is deleted, you should respond:

Delphi
```
procedure TMyForm.OnListDeleteItem( aSender: TObject;
                                    aItem: TListItem;
                                    aData: TPTShListData );
begin
  TObject(aData.Data).Free;
end;
```

C++Builder
```
void __fastcall TMyForm::OnListDeleteItem( System::TObject* aSender,
Comctrls::TListItem* aNode, TPTShListData* aShListData )
{
  delete (TObject*)(aShListData->Data);
}
```

OnCustomDrawSh and OnCustomDrawShEx events

Applies to
TPTShellList

Delphi Declaration
```
type TPTShLvCustomDrawEvent =
       procedure( aSender: TObject;
                  aCD: TPTCustomDraw;
                  aItem: TListItem;
                  aData: TPTShListData ) of object;

property OnCustomDrawSh: TPTShLvCustomDrawEvent;

property OnCustomDrawShEx: TPTShLvCustomDrawEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTShLvCustomDrawEvent)(System::TObject*
     aSender, Upttreelist::TPTCustomDraw* aCD, Comctrls::TListItem* aItem,
     TPTShListData* aData);

__property TPTShLvCustomDrawEvent OnCustomDrawSh;

__property TPTShLvCustomDrawEvent OnCustomDrawShEx;
```

Description
You can change the appearance of individual items in the list by adjusting properties of the given *aCD* object.

Use the Font property of *aCD* to change the appearance of *aItem.*

Access shell-related data about the item with *aData*.

The OnLvCustomDrawSh event is only called for the *ptcdsItemPrePaint* draw stage, whereas OnLvCustomDrawShEx is called for every draw stage.

See TPTListView.OnPTCustomDraw for more information.

OnFolderChanged event

Applies to
TPTShellList component

Delphi Declaration
```
type TNotifyEvent = procedure(sender: TObject) of object;

property OnFolderChanged: TNotifyEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TNotifyEvent)(System::TObject* Sender);

__property TNotifyEvent OnFolderChanged;
```

Description
The OnFolderChanged event occurs when the user changes the directory that is displayed in the list view. This can happen when the user double-clicks on a directory, clicks the *Up* button, or uses the combo box or tree view to navigate through the directory structure.

This event also fires if the folder is changed via code.

TPTShListData class

Unit
UPTShellControls

Description
This class handles data for each node of the TPTShellList component.

Delphi Declaration **C++Builder Declaration**
```
property AbsoluteIdList: PItemIdList;     __property PitemIdList AbsoluteIdList;
```
Read only. Returns the item id list for the tree node relative to the desktop. This particular id list is used in many other functions.

You do not need to free the returned id list. If you want to remember it (store it in a non-local variable) you must copy it — use the CopyIdList function for this purpose.

Delphi Declaration **C++Builder Declaration**
```
property Attributes: DWORD;               __property unsigned Attributes;
```
Read only. Returns the raw SFGAO_ attributes for the folder. These attributes are cached by the TPTShListData object.

```
property ColText[col: Integer]: String;     __property System::AnsiString* ColText[int col];
```
Read only. Returns the item's text for a particular column of the list control. If the column does not apply, it returns an empty string.

```
property Data: Pointer;                   __property void* Data;
```
This property allows the user of the list to store extra data with each item. If you store a heap allocated structure or object with this pointer, you must ensure that it is freed in all instances by responding to the TPTShellList.OnDeleteItem event.

Example
This example associates a new instance of a user-defined object with the first list item.

Delphi:
```
ShellList1.ShListData[0].Data := TMyExtraData.Create;
```
C++Builder:
```
ShellList1->ShListData->Item[0]->Data = new TMyExtraData;
```

```
property DisplayName: String;             __property System::AnsiString* DisplayName;
```
Read only. Returns the name of the item, suitable for displaying to the user. If the user has "hide extensions" on then the display name will not include extensions. This function can take some time to execute, especially if you call it for every item in a list of hundreds.

```
property Editable: Boolean;               __property bool Editable;
```
Read only. This is not the same as the read-only file attribute. If property is *true* then this node is potentially editable. Read-only files are still 'editable'. Examples of nodes that are never editable are Desktop, My Computer, Network Neighborhood etc.

```
property FileName: String;                __property System::AnsiString* FileName;
```
Read only. Returns the file name of the selected item suitable for use with system functions. To present a name to the user you should not use the property, use DisplayName instead. Returns an empty string for non-filesystem items.

```
property FileType: String;                __property System::AnsiString* FileType;
```
Read only. Returns the description of the item.

```
function IsFileSystem: Boolean;           bool __fastcall IsFileSystem(void);
```
Returns *true* if the item is a file system object (such as a file or directory, not including virtual folders), *false* if otherwise.

```
function IsFolder: Boolean;               bool __fastcall IsFolder(void);
```
Returns *true* if the item is a folder (including virtual folders), *false* if otherwise.

```
function IsLnkShortcut: Boolean;          bool __fastcall IsLnkShortcut(void);
```

Returns *true* if the items is a '.lnk' file shortcut, *false* if otherwise.

```
property Modified: String;                __property System::AnsiString* Modified;
```
Read only. Returns an appropriately formatted date/time string of the last modified date/time of the item. If this property does not apply, it returns an empty string.

```
property Owner: TPTShellList;             __property TPTShellList* Owner;
```
Read only. Returns the owner of the item.

```
property PathName: String;                __property System::AnsiString* PathName;
```
Read only. Returns the fully qualified path name of the item. Returns an empty string for non-filesystem items.

```
property RelativeIdList: PItemIdList;     __property PItemIdList RelativeIdList;
```
Read only. This property returns the item id list of the item, relative to the item's parent folder. You do not need to free the returned item. If you want to keep it, you should copy it with CopyIdList.

```
property Size: String;                    __property System::AnsiString* Size;
```
Read only. If the item has a size in bytes, this property returns the string as seen in Explorer's list view - such as "1KB". If this property does not apply, it returns an empty string. Note that Explorer's list view always shows amounts in KB, whereas other parts of the shell use bytes, KB, MB, GB and TB. This property only shows the KB amount.

TPTSHELLCOMBO COMPONENT

TPTShellCombo component

Hierarchy Properties Methods Events

Delphi Unit
UPTShellControls

C++Builder Header
UPTShellControls.hpp

Description
TPTShellCombo is derived from TPTImageCombo and implements an explorer-like combobox.

You can modify the behaviour of the control with the Options property.

Use the SelectedFolder property to get and set the currently selected folder.

Use the GoUp method to navigate back up the namespace.

Use the FillItems method to initialise or refresh the contents of the control.

The control can act independently or automatically together with TPTShellTree or TPTShellList controls. To link to a TPTShellList control set the ShellList property to the desired list control. To link to a TPTShellTree control set the ShellTree property to the desired tree control. Only one of ShellList or ShellTree properties can be set at a time.

Hierarchy

TObject
|
TPersistent
|
TComponent
|
TControl
|
TWinControl
|
TPTCustomCombobox
|
TPTCustomImageCombo

TPTShellCombo properties

TPTShellCombo Legend

Derived from TPTCustomShellCombo


IndentPixels


Options


SelectedFolder

ShComboData


ShellList


ShellTree

TPTShellCombo methods

TPTShellCombo Legend

Derived from TPTCustomShellCombo

FillItems

GoUp

Synchronize

IndentPixels property

Applies to
TPTShellCombo

Delphi Declaration
```
property IndentPixels: Integer;
```

C++Builder Declaration
```
__property int IndentPixels;
```

Description
Number of pixels per level of indentation. Applies to all items.

Options property

Applies to
TPTShellCombo

Delphi Declaration
```
property Options: TPTShellComboOptions;
```

C++Builder Declaration
```
__property TPTShellComboOptions Options;
```

Description
These are the possible values that can be included in the Options set for the shell combo control.

Identifier	Meaning
ptscoAutofill	When this option is set, the contents of the combo box are automatically initialised.
ptscoNonFilesystemAncestors	When *true* items which are not part of the file system, and not parents of file system folders are automatically excluded from the list. Candidates for exclusion are Control Panel, Printers and Recycle bin for example. My Computer will not be excluded since it is the ancestor of file system items.

ShellList property

See Also

Applies to
TPTShellCombo

Delphi Declaration
```
property ShellList: TPTShellList;
```

C++Builder Declaration
```
__property TPTShellList* ShellList;
```

Description
You can set the *ShellList* property to any TPTShellList control available on the same form. When set, the list and combo are synchronised.

You can set only one of ShellList and ShellTree. After setting ShellList, subsequent attempts to set ShellTree will be ignored. To clear the setting, set ShellList to *nil*.

If ShellList is set, the combo and list work together. To make all three controls (combo, tree and list) work together, you must set the ShellTree property of the combo and the ShellList property of the tree.

See Also

TPTShellList component
TPTShellTree component

ShellTree property

See Also

Applies to
TPTShellCombo

Delphi Declaration
```
property ShellTree: TPTShellTree;
```

C++Builder Declaration
```
__property TPTShellTree* ShellTree;
```

Description
You can set the *ShellTree* property to any TPTShellTree control available on the same form. When set, the tree and combo are synchronised.

You can set only one of ShellList and ShellTree. After setting ShellTree, subsequent attempts to set ShellList will be ignored. To clear the setting, set ShellTree to *nil*.

If ShellTree is set, the combo and tree work together. To make all three controls (combo, tree and list) work together, you must set the ShellTree property of the combo and the ShellList property of the tree.

See Also

TPTShellList component
TPTShellTree component

SelectedFolder property

Applies to
TPTShellCombo

Delphi Declaration
```
property SelectedFolder: TPTShellLocator;
```

C++Builder Declaration
```
__property TPTShellLocator* SelectedFolder;
```

Description
Run time only. You can set this property to change the currently selected folder, or read it to retrieve the current selection information.

ShComboData property

Applies to
TPTShellCombo

Delphi Declaration
```
property ShComboData[ idx: Integer ]: TPTShComboData;
```

C++Builder Declaration
```
__property TPTShComboData* ShComboData[ int idx ];
```

Description
Run time and read only.

This property is simply a convenient way of accessing the TPTShComboData object for a given tree node. The node is specified by index.

Equivalent code is:

Delphi:
```
data := TObject(PTShellCombo1.Items[ idx ].Data) as TPTShComboData;
```

C++Builder:
```
TPTShComboData* data = dynamic_cast<TPTShComboData*>(PTShellCombo1->Items[idx]->Data)
```

It's much more convenient to express it this way:

Delphi:
```
data := PTShellCombo1.ShComboData[ idx ];
```

C++Builder:
```
TPTShComboData* data = PTShellCombo1->ShComboData[idx];
```

FillItems method

Applies to
TPTShellCombo

Delphi Declaration
```
procedure FillItems;
```

C++Builder Declaration
```
void __fastcall FillItems(void);
```

Description
FillItems adds items to the combobox. You must call FillItems at least once for the combo to function. If ptscoAutoFill is in the Options property then FillItems is automatically called once before the combo becomes visible.

GoUp method

Applies to
TPTShellCombo

Delphi Declaration
```
procedure GoUp( aLevels: Integer );
```

C++Builder Declaration
```
void __fastcall GoUp( int aLevels );
```

Description
Calling this method moves the selected item back *aLevels* in the hierarchy.

aLevels must be greater than or equal to 1.

You can navigate to the top of the namespace with the following:

Delphi:
```
PTShellCombo1.SelectedFolder.IdList := nil;
```

C++Builder:
```
PTShellCombo1->SelectedFolder->IdList = NULL;
```

OnAddItem event

Applies to
TPTShellCombo

Delphi Declaration
```
property OnAddItem: TPTShAddItemEvent;
```

C++Builder Declaration
```
__property TPTShAddItemEvent OnAddItem;
```

Description
This event is called every time an item is added to the combo box. You have the opportunity to filter out the unwanted items. You should respond to this event in the same way as you would respond to the OnAddItem event in TPTShellTree.

TPTShComboData class

Delphi Unit
UPTShellControls

C++Builder Header
UPTShellControls.hpp

Description
This class handles data for each item of the TPTShellCombo component.

Delphi Declaration
```
property AbsoluteIdList: PItemIdList;
```
C++Builder Declaration
```
__property PitemIdList AbsoluteIdList;
```
Read only. Returns the item id list for the item relative to the desktop. This particular id list is used in many other functions.

You do not need to free the returned id list. If you want to remember it (store it in a non-local variable) you must copy it — use the CopyIdList function for this purpose.

```
property ParentIdList: PItemIdList;
```
```
__property PitemIdList ParentIdList;
```
Read only. Returns the absolute item id list of the parent folder of the combo item.

You do not need to free the returned id list. If you want to remember it (store it in a non-local variable) you must copy it — use the CopyIdList function for this purpose.

```
property ParentIShf: IShellFolder;
```
```
__property IshellFolder ParentIShf;
```
Read only. The IShellFolder interface for the parent of this combo item. For the desktop node, this property is **nil**.

```
property RelativeIdList: PitemIdList;
```
```
__property PitemIdList RelativeIdList;
```
Read only. This property returns the item id list of the item, relative to the item's parent folder. You do not need to free the returned item. If you want to keep it, you should copy it with CopyIdList.

UPTIMAGECOMBO UNIT

UPTImageCombo unit

The UPTImageCombo unit contains the TPTImageCombo component and related classes.

Components

- TPTCustomImageCombo
- TPTCustomCombobox
- TPTImageCombo
- TPTCombobox

Classes

- TPTImageComboItem

TPTCustomImageCombo component

Delphi Unit	C++Builder Header
UPTImageCombo	UPTImageCombo.hpp

Description
The TPTCustomImageCombo component is the abstract base type for the TPTImageCombo and TPTShellCombo components.

All the properties declared in TPTCustomImageCombo are declared protected. The derived components publish those properties that are relevant in their case.

See the references to the derived components for details.

TPTCustomCombobox component

Delphi Unit	C++Builder Unit
UPTImageCombo	UPTImageCombo

Description
The TPTCustomCombobox component is the abstract base type for the TPTCombobox and TPTImageCombo components.

The events declared in TPTCustomCombobox are declared protected. The derived components publish those properties that are relevant in their case.

See the references to the derived components for details.

TPTImageCombo component

Hierarchy Properties Methods Events

Delphi Unit
UPTImageCombo

C++Builder Header
UPTImageCombo.hpp

Description
TPTImageCombo implements a combobox with an associated image and indentation level for each item. The images are taken from an associated image list.

Hierarchy

TObject
|
TPersistent
|
TComponent
|
TControl
|
TWinControl
|
TPTCustomCombobox

TPTImageCombo properties

TPTImageCombo Legend

Derived from TPTCustomImageCombo


AutoSizeHeight


ImageList


IndentPixels


ImageComboItem

TPTImageCombo methods

TPTImageCombo Legend

Derived from TPTCustomImageCombo

AddItem

TPTImageCombo events

TPTImageCombo Legend

Derived from TPTCustomImageCombo


OnDeleteItem


OnGetItemData

AutoSizeHeight property

Applies to
TPTImageCombo

Delphi Declaration
```
property AutoSizeHeight: Boolean;
```

C++Builder Declaration
```
__property bool AutoSizeHeight;
```

Description
When this property is *true*, any change to the Font or ImageList of the image combo will cause the control to resize vertically to fit the new Font or ImageList height by automatically setting the ItemHeight. The greater of the Font and ImageList heights are used for the new item height.

Default value is *true*.

ImageList property

Applies to
TPTImageCombo

Delphi Declaration
```
property ImageList: TImageList;
```

C++Builder Declaration
```
__property Controls::TImageList* ImageList;
```

Description
Run-time only. The Images property for an image combo box determines which image list (TImageList) is associated with the tree view. An image list contains a list of bitmaps that can be displayed to the left of an item's label.

IndentPixels property

Applies to
TPTImageCombo

Delphi Declaration
```
property IndentPixels: Integer;
```

C++Builder Declaration
```
__property int IndentPixels;
```

Description
Number of pixels per level of indentation. Applies to all items.

ImageComboItem property

Applies to
TPTImageCombo

Delphi Declaration
```
property ImageComboItem[ idx: Integer ]: TPTImageComboItem;
```

C++Builder Declaration
```
__property TPTImageComboItem* ImageComboItem[ int idx ];
```

Description
Run time and read only.

This property is a convenient way of accessing the TPTImageComboItem object for a given list item. The item is specified by index.

Equivalent code is:

Delphi:
```
data := PTShellCombo1.Items.Objects[idx] as TPTImageComboItem;
```

C++Builder:
```
TPTImageComboItem* data = dynamic_cast<TPTImageComboItem*)(PTShellCombo1->Items->Obje
```

It is much more convenient this way:

Delphi:
```
data := PTShellCombo1.ImageComboItem[ idx ];
```

C++Builder:
```
TPTImageComboItem* data = PTShellCombo1->ImageComboItem[idx];
```

AddItem method

Applies to
TPTImageCombo

Delphi Declaration
```
function AddItem( aCaption: String;  aIcon: Integer;  aIndent: Integer ):
TPTImageComboItem;
```

C++Builder Declaration
```
TPTImageComboItem* __fastcall AddItem( System::AnsiString* aCaption,  int aIcon,
int aIndent );
```

Description
Use this method to add items in preference to calling members of the Items property.

OnDeleteItem event

See Also

Applies to
TPTImageCombo

Delphi Declaration

```
TPTDeleteComboItemEvent = procedure( aSender: TObject;
    aItem: TPTImageComboItem ) of object;
property OnDeleteItem: TPTDeleteComboItemEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TPTDeleteComboItemEvent)(System::TObject*
     aSender, TPTImageComboItem* aItem);
__property TPTDeleteComboItemEvent OnDeleteItem;
```

Description
If you associate extra data with each combobox item, you should respond to this event to delete that extra data. If you do not respond to this event, your extra data will leak unless freed elsewhere.

Example
This example assumes that a user-defined object of type TMyDataItem has been assigned to the Data property of the TPTImageComboItem for each combobox item. In this situation, you might respond to this OnDeleteItem event by freeing your object.

Delphi

```
procedure TMyForm.OnImageCombo1DeleteItem( aSender: TObject;  aItem:
TPTImageComboItem );
begin
  TMyDataItem(aItem.Data).Free;
end;
```

C++Builder

```
void __TMyForm::OnImageCombo1DeleteItem( System::TObject* aSender,
TPTImageComboItem* aItem )
{
  delete (TMyDataItem*)(aItem->Data);
}
```

See Also

TPTImageComboItem class
TPTImageCombo.AddItem method

OnGetItemData event

See Also

Applies to
TPTImageCombo

Delphi Declaration

```
TPTImageComboGetItemDataEvent = procedure( aSender: TObject;
    aItem: TPTImageComboItem ) of object;
property OnGetItemData: TPTImageComboGetItemDataEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure * TPTImageComboGetItemDataEvent)(System::TObject*
     aSender, TPTImageComboItem* aItem);
__property TPTImageComboGetItemDataEvent OnGetItemData;
```

Description
This event is called every time a combo item is drawn. You can use this event to defer supplying, for example, *Caption* and *mageIndex* values for *aItem*. The TPTShellCombo control uses this feature to improve performance.

See Also

TPTImageComboItem class
TPTImageCombo.AddItem method

TPTImageComboItem class

Unit
UPTImageCombo

Description
This class handles data for each item of the TPTImageCombo component. The TPTImageCombo component stores one of these objects with each item in the Items.Objects[ ] property. This means you should *never* write to the Items.Objects array of a TPTImageCombo component. You should instead use the ImageComboItem array to access the TPTImageComboItem objects for each combo item.

Delphi Declaration **C++Builder Declaration**

```
property Caption: String;                    __property System::AnsiString* Caption;
```

The text part of the combo item. The text is drawn to the right of the image.

```
property Data: Pointer;                      __property void* Data;
```

This property allows the user of the image combo to store extra data with each item. If you store a heap allocated structure or object with this pointer, you must ensure that it is freed in all cases by responding to the TPTImageCombo.OnDeleteItem event.

Example
This example associates a new instance of a user-defined object with the currently selected node.

Delphi:
```
ImageCombo1.ImageComboItem[0].Data := TMyExtraData.Create;
```
C++Builder:
```
ImageCombo1->ImageComboItem[0]->Data = new TMyExtraData;
```

```
property Indent: Integer;                    __property int Indent;
```

Number of levels to indent this item. The number of pixels per level is set by the owning TPTImageCombo component.

```
property ImageIndex: Integer;                __property int ImageIndex;
```

Index into TPTImageCombo image list to use for this item.

```
property OverlayIndex: Integer;              __property int OverlayIndex;
```

Index into the TPTImageCombo image list of the image to use as an overlay. The ImageIndex image is drawn first and the OverlayIndex is transparently drawn on top.

```
property Tag: Integer;                       __property int Tag;
```

Arbitrary data field available for the storage of information with each combo item.

TPTCombobox component

Hierarchy Properties Methods Events

Delphi Unit
UPTImageCombo

C++Builder Header
UPTImageCombo.hpp

Description
TPTCombobox enhances the VCL TCombobox by adding four new events.

Hierarchy

TObject
|
TPersistent
|
TComponent
|
TControl
|
TWinControl

TPTCombobox events

TPTCombobox Legend

Derived from TPTCustomCombobox

OnDeleteItem

OnCloseUp

OnSelEndCancel

OnSelEndOk

OnDeleteItem event

Applies to
TPTCombobox

Delphi Declaration
```
type TPTDeleteComboItemEvent =
       procedure( aSender: TObject;  aItem: TObject ) of object;

property OnDeleteItem: TPTDeleteComboItemEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTDeleteComboItemEvent)(System::TObject*
      aSender, void * aItem);

__property TPTDeleteComboItemEvent OnDeleteItem;
```

Description
This event gives you an opportunity to delete any data you have stored as an Object associated with an Item.

You can store data with an item by using the Objects property of the Items property. e.g.

Delphi: `PTCombobox.Items.Objects[3] := TMyObject.Create;`

C++Builder: `PTCombobox->Items->Objects[3] = new TMyObject;`

When that item is deleted, this OnDeleteItem event will be fired giving you an opportunity to delete the object.

Delphi
```
procedure TMyForm.OnCombo1DeleteItem( aSender: TObject;  aItem: TObject );
begin
  aItem.Free;
end;
```

C++Builder
```
void __fastcall TMyForm::OnCombo1DeleteItem( TObject* aSender, TObject* aItem )
{
  delete aItem;
}
```

This event is fired whenever the combobox receives a CN_DELETEITEM notification message.

OnCloseUp event

Applies to
TPTCombobox

Delphi Declaration
```
property OnCloseUp: TNotifyEvent;
```

C++Builder Declaration
```
__property TNotifyEvent OnCloseUp;
```

Description
This event is fired when the drop down list box of the combo box has been closed. In the case when both OnChange and OnCloseUp events are sent at the same time, you cannot predict their order.

OnSelEndCancel event

Applies to
TPTCombobox

Delphi Declaration
```
property OnSelEndCancel: TNotifyEvent;
```

C++Builder Declaration
```
__property TNotifyEvent OnSelEndCancel;
```

Description
This event is fired when the user selects an item, but then selects another control or closes the form. It indicates the user's initial selection is to be ignored.

OnSelEndOk event

Applies to
TPTCombobox

Delphi Declaration
```
property OnSelEndOk: TNotifyEvent;
```

C++Builder Declaration
```
__property TNotifyEvent OnSelEndOk;
```

Description
This event is fired when the user selects a list item, or selects an item and then closes the list. It indicates that the user's selection is to be processed.

It is generally preferable to process this event instead of OnChange, especially if the action to be performed when the selection is processed is time consuming.

Consider the user who selects an item from the combo box with the keyboard. They tab to the control, drop down the list and press the up/down keys until they reach the desired item. Every time they move up or down, an OnChange event is fired. If you wish to perform processing based on the user selection, it is preferable to wait until they have finished their selection before doing the processing (in most cases). This is the purpose of this event.

UPTSPLITTER UNIT

UPTSplitter unit

The UPTSpliiter unit contains the TPTSplitter control.

Classes

TPTSplitter

TPTSplitter component

Hierarchy Properties Methods Events

Delphi Unit
UPTSplitter

C++Builder Header
UPTSplitter.hpp

Description
TPTSplitter implements a two-pane container onto which other components can be placed. At run-time, the user can adjust the splitter bar by dragging with the left button.

The splitter bar can be dragged at design-time with the right mouse button.

Note: You can construct pane views with more than two panes by placing one TPTSplitter component inside one of the panes of another TPTSplitter.

The following table shows which is Pane 1 and which is Pane 2 for both splitter orientations.

	ptstVertical	**ptstHorizontal**
Pane 1	Left	Top
Pane 2	Right	Bottom

Use the OnChange event to detect when the Position property changes.

The OnResize event is fired when the bounds of the splitter itself are resized. It is not called when the splitter bar is positioned – use OnChange for that.

Use OnSplitterDrag and OnSplitterDrop to monitor user interaction with the splitter bar.

Hierarchy

TObject
|
TPersistent
|
TComponent
|
TControl
|
TWinControl
|
TCustomControl

TPTSplitter properties

TPTSplitter Legend

Implemented in TPTSplitter


FullDragMode


Pane1


Pane2


Pane1Color


Pane2Color


Pane1FrameStyle


Pane2FrameStyle


Pane1MinSize


Pane2MinSize


Position

Proportion


Proportional


SplitterWidth


Style

TPTSplitter methods

TPTSplitter Legend

Implemented in TPTSplitter

SwapPanes

TPTSplitter methods

TPTSplitter Legend

Implemented in TPTSplitter

OnChange

OnResize

OnSplitterDrag

OnSplitterDrop

FullDragMode property

Applies to
TPTSplitter

Delphi Declaration
```
type TPTFullDragMode = (ptfdOff, ptfdOn, ptfdUser);

property FullDragMode: TPTFullDragMode;
```

C++Builder Declaration
```
enum TPTFullDragMode { ptfdOff, ptfdOn, ptfdUser };

__property TPTFullDragMode FullDragMode;
```

Description
When being dragged, the splitter bar can appear as a hatched region with the positioning taking place after the mouse is released, or as a continuous movement of the bar and the panes with the mouse. This is similar to the *Full Window Drag* feature found in Windows.

The FullDragMode property can be one of three values:

Value	Meaning
ptfdOff	Full drag mode is disabled. The hatched mode of dragging is always used.
ptfdOn	Full drag mode is enabled. The splitter bar continuously resizes during the drag operation.
ptfdUser	Full drag mode is either enabled or disabled, depending on the user's *Full Window Drag* preference as specified in the Windows Display Properties control panel applet.

In most cases, you should leave this property set to *ptfdUser* to respect the user's wishes.

Pane1 and Pane2 properties

Applies to
TPTSplitter

Delphi Declaration
```
property Pane1: TPTPane;
property Pane2: TPTPane;
```

C++Builder Declaration
```
__property TPTPane* Pane1;
__property TPTPane* Pane2;
```

Description
Run time and read only. Use these properties to access the components representing the panes of the splitter.

Example
This example shows how to retrieve the count of components in each pane.

Delphi
```
var leftcount, rightcount: Integer;
begin
  leftcount := PTSplitter1.Pane1.ControlCount;
  rightcount := PTSplitter1.Pane2.ControlCount;
end;
```

C++Builder
```
int leftcount = PTSplitter->Pane1->ControlCount;
int rightcount = PTSplitter->Pane2->ControlCount;
```

Pane1Color and Pane2Color properties

Applies to
TPTSplitter

Delphi Declaration
```
property Pane1Color: TColor;
property Pane2Color: TColor;
```

C++Builder Declaration
```
__property Graphics::TColor Pane1Color;
__property Graphics::TColor Pane2Color;
```

Description
Selects the color of each of the panes.

Example
This example sets the colour of the second pane to *clRed*.

Delphi:
```
PTSplitter1.Pane2Color := clRed;
```

C++Builder:
```
PTSplitter1->Pane2Color = clRed;
```

Pane1FrameStyle and Pane2FrameStyle properties

Applies to
TPTSplitter

Delphi Declaration
```
property Pane1FrameStyle: TPTFrameStyle;
property Pane2FrameStyle: TPTFrameStyle;
```

C++Builder Declaration
```
__property TPTFrameStyle Pane1FrameStyle;
__property TPTFrameStyle Pane2FrameStyle;
```

Description
Selects the style of the border of each of the panes.

If a pane will contain a single control aligned to the client area you should set the pane frame style to *ptfsNone* and use the border of the contained control.

Pane1MinSize and Pane2MinSize properties

Applies to
TPTSplitter

Delphi Declaration
```
property Pane1MinSize: Integer;
property Pane2MinSize: Integer;
```

C++Builder Declaration
```
__property int Pane1MinSize;
__property int Pane2MinSize;
```

Description
Determines the minimium size of the first and second panes. This prevents the user dragging the splitter inside these minimum ranges, however you can still programatically set the Position property to within these minimums.

Position property

Applies to
TPTSplitter

Delphi Declaration
```
property Position: Integer;
```

C++Builder Declaration
```
__property int Position;
```

Description
This property determines the position of the splitter bar, measured in pixel units.

To change the position at design time, click and drag with the right mouse button.

Proportion property

Applies to
TPTSplitter

Delphi Declaration
```
property Position: Single;
```

C++Builder Declaration
```
__property float Proportion;
```

Description
Run time only. This property returns or sets the position of the splitter bar as a proportion of the width or height of the control (depending on horizontal or vertical orientation).

The value is a factor between 0.0 and 1.0 representing the distance from Position 0 where 0.0 = pixel 0 and 1.0 = the width/height of the control.

A typical use of this value is to position the splitter bar fully left, right or centered. To center the bar, simply set the *Proportion* property to 0.5, fully left (or top) is 0.0, fully right (or bottom) is 1.0.

Proportional property

Applies to
TPTSplitter

Delphi Declaration
```
property Proportional: Boolean;
```

C++Builder Declaration
```
__property bool Proportional;
```

Description
This property only has effect when the splitter control is being resized. When Proportional is *true* and the splitter control is resized, the splitter bar is repositioned to maintain the relative percentage of space available to the left and right panes.

When *false*, the splitter bar remains in the same place regardless of how big or small the splitter control gets.

SplitterWidth property

Applies to
TPTSplitter

Delphi Declaration
```
property SplitterWidth: Integer;
```

C++Builder Declaration
```
__property int SplitterWidth;
```

Description
This property determines the width in pixels of the splitter bar.

Style property

Applies to
TPTSplitter

Delphi Declaration
```
type TPTSplitterStyle = (ptstVertical, ptstHorizontal);

property Style: TPTSplitterStyle;
```

C++Builder Declaration
```
enum TPTSplitterStyle { ptstVertical, ptstHorizontal };

__property TPTSplitterStyle Style;
```

Description
This property determines in which direction the splitter bar is drawn.

ptstVertical	Bar is drawn vertically and measured from left to right, left being 0.
ptstHorizontal	Bar is drawn horizontally and measured from top to bottom, top begin 0.

SwapPanes method

Applies to
TPTSplitter

Delphi Declaration
```
procedure SwapPanes( afSwapSplit: Boolean );
```

C++Builder Declaration
```
void SwapPanes( bool afSwapSplit );
```

Description
Swaps the left and right panes. If *afSwapSplit* is *true* then the splitter line is also swapped. For example, if the splitter bar was 100 pixels from the left edge and SwapPanes was called with *afSwapSplit true*, the bar would be moved to 100 pixels from the right edge. If *afSwapSplit* were *false*, the bar would remain 100 pixels from the left.

You can swap the panes at design-time from the right-click menu for the component.

OnChange event

Applies to
TPTSplitter

Delphi Declaration
```
property OnChange: TNotifyEvent;
```

C++Builder Declaration
```
__property TNotifyEvent OnChange;
```

Description
The OnChange event occurs after the splitter bar is positioned.

OnResize event

Applies to
TPTSplitter

Delphi Declaration
```
property OnResize: TNotifyEvent;
```

C++Builder Declaration
```
__property TNotifyEvent OnResize;
```

Description
The OnResize event occurs when the splitter control itself is resized. It is not called when the splitter bar is positioned.

OnSplitterDrag event

Applies to
TPTSplitter

Delphi Declaration
```
TPTSplitterDragEvent = procedure( aSender: TObject;  aPosition: Integer ) of object;
property OnSplitterDrag: TPTSplitterDragEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTSplitterDragEvent)(System::TObject* aSender,
     int aPosition);
__property TPTSplitterDragEvent OnSplitterDrag;
```

Description
This event is called while the splitter bar is being dragged. *aPosition* is the current drag position in pixels.

OnSplitterDrop event

Applies to
TPTSplitter

Delphi Declaration
```
TPTSplitterDragEvent = procedure( aSender: TObject;  aPosition: Integer ) of object;
property OnSplitterDrop: TPTSplitterDropEvent;
```

C++Builder Declaration
```
typedef void __fastcall (__closure *TPTSplitterDragEvent)(System::TObject* aSender,
     int aPosition);
__property TPTSplitterDragEvent OnSplitterDrop;
```

Description
This event is called after the splitter bar has been successfully repositioned due to mouse action. *aPosition* is the new drag position.

Note this event is not called if the splitter bar position is set with code.

UPTSYSFOLDERDLG UNIT

UPTSysFolderDlg unit

The UPTSysFolderDlg unit contains Delphi-friendly component and function wrappers around the SHBrowseForFolder function.

Classes

TPTSysFolderDlg

Types

TPTSysFolderDlgOption

TPTSysFolderDlgOption type

Delphi Declaration

```
type TPTSysFolderDlgOption = ( fdoComputers, fdoPrinters, fdoDontGoBelowDomain,
     fdoReturnFSAncestors, fdoReturnOnlyFSDirs, fdoStatusText );

TPTSysFolderDlgOptions = set of TPTSysFolderDlgOption;
```

C++Builder Declaration

```
enum TPTSysFolderDlgOption { fdoComputers, fdoPrinters, fdoDontGoBelowDomain,
     fdoReturnFSAncestors,     fdoReturnOnlyFSDirs, fdoStatusText };

typedef Set<TPTSysFolderDlgOption, fdoComputers, fdoStatusText>
     TPTSysFolderDlgOptions;
```

Description

Value	Description
fdoComputers	When set, this flag leaves the OK button disabled until a computer is selected. This flag is obviously most useful when combined with a domain of **csidlNetwork**. See System Folder Constants (TCSIDL type).
fdoPrinters	When set, this flag limits what appears in the tree to printers, or intermediate nodes required to reach a printer. In some cases it causes everything (including files) to be shown instead. The only case where this flag does anything useful and works as apparently intended is when combined with the **csidlNetwork** domain. In this case, it lets you browse the network and disables the OK button unless a network printer is selected. See System Folder Constants (TCSIDL type).
fdoDontGoBelowDomain	Prevents the Network Neighbourhood from showing users in your workgroup at the first level. You still see the "Entire Network" item, from which you can navigate to your workgroup.
fdoReturnFSAncestors	Seems to behave the same as fdoComputers.
fdoReturnOnlyFSDirs	Disables the OK button unless the selected item is a file-system folder, or has an associated file system folder. UNC pathnames are considered part of the file-system.
fdoStatusText	Adds a line of text to the dialog. You can change the text by assigning the StatusText property during dialog execution.

TPTSysFolderDlg component

TPTSysFolderDlg component

Properties Methods Events

Delphi Unit
UPTSysFolderDlg

C++Builder Header
UPTSysFolderDlg.hpp

Description
This class wraps the functionality provided by the SHBrowseForFolder function. This functionality is built in to Windows 95 and Windows NT 4.0, but unfortunately you have very little control over the behaviour of the dialog. It is certainly not as flexible as the common dialogs - there is no provision to add or position controls. If you wanted to place a "Create New Folder" button on the SHBrowseForFolder dialog, you're out of luck. Also, the system dialog does not detect file-system changes.

Fortunately, the TPTShellTree class makes implementing a replacement SHBrowseForFolder dialog easy. A flexible replacement is provided in the FPTFolderBrowseDlg unit. If you would rather use the system dialog, you can access it with this class.

You can test the browse dialog at design time by right-clicking on the component and selecting the "Test" menu item. This will execute the browse dialog using the current properties. Events will not be called of course.

Properties

- Domain
- Options
- Title

Run time only Properties

- Executing
- FolderPath
- FolderIdList
- OkEnabled
- StatusText

Methods

Execute

Events

- OnInitialized
- OnSelChanged

Executing property

Applies to
TPTSysFolderDlg

Delphi Declaration
```
property Executing: Boolean;
```

C++Builder Declaration
```
__property bool Executing;
```

Description
Read only. This property returns *true* during a call to the Execute method. When Execute returns, this property again returns *false*.

FolderPath property

See Also

Applies to
TPTSysFolderDlg

Delphi Declaration
```
property FolderPath: String;
```

C++Builder Declaration
```
__property System::AnsiString FolderPath;
```

Description
Run time only. This property can be used as both an input and an output property. Before calling Execute, you can assign this property to the name of a file-system folder. When Execute is called, the named folder will start selected. If the folder doesn't exist, then a folder somewhere along the path in FolderPath will start highlighted. If no FolderPath Is assigned, selection begins at the Desktop folder.

Note that not all namespace elements can be represented with FolderPath. Only file-system elements - drives and directories. If the selected item was not a file-system element (like My Computer or Control Panel) FolderPath will be empty. In this case, you can use FolderIdList to obtain the item id list.

See Also

Execute property

FolderIdList property

Applies to
TPTSysFolderDlg

Delphi Declaration
```
property FolderIdList: PItemIdList;
```

C++Builder Declaration
```
__property PItemIdList FolderIdList;
```

Description
Run time only. This property can be used as both an input and an output property. Before calling Execute, you can assign this property to the item id list of a namespace element. When Execute is called, the named folder will start selected. If the folder doesn't exist, then a folder somewhere along the path in FolderIdList will start highlighted. If no FolderIdList Is assigned, selection begins at the Desktop folder.

OkEnabled property

See Also

Applies to
TPTSysFolderDlg

Delphi Declaration
```
property OkEnabled: Boolean;
```

C++Builder Declaration
```
__property bool OkEnabled;
```

Description
Run time and **write** only. Use this property from within event callbacks to change the enabled state of the OK button.

See Also

StatusText property
OnInitialized event
OnSelChanged event

StatusText property

See Also

Applies to
TPTSysFolderDlg

Delphi Declaration
```
property StatusText: String;
```

C++Builder Declaration
```
__property System::AnsiString StatusText;
```

Description
Run time and **write** only. Use this property from within event callbacks to change the line of status text. You must have turned on the status line first with the fdoStatusText option.

See Also

OkEnabled property
OnInitialized event
OnSelChanged event

Domain property

Applies to
TPTSysFolderDlg

Delphi Declaration
```
property Domain: TCSIDL;
```

C++Builder Declaration
```
__property TCSIDL Domain;
```

Description
This property determines where the browse tree starts from.

Options property

Applies to
TPTSysFolderDlg

Delphi Declaration
```
property Options: TPTSysFolderDlgOptions;
```

C++Builder Declaration
```
__property TPTSysFolderDlgOptions Options;
```

Description
This property modifies the default behaviour of the browse dialog.

Title property

Applies to
TPTSysFolderDlg

Delphi Declaration
```
property Title: String;
```

C++Builder Declaration
```
__property System::AnsiString Title;
```

Description
This property sets the title text in the browse dialog. The title is not the dialog caption, it is a label just below the caption. You can only change the dialog caption by responding to the OnInitialized event and calling Windows.SetWindowText.

OnInitialized event

See Also

Applies to
TPTSysFolderDlg

Delphi Declaration

```
TFDOnInitializedEvent = procedure( aSender: TPTSysFolderDlg;  hwnd: HWND ) of
object;
property OnInitialized: TFDOnInitializedEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TFDOnInitializedEvent)(TPTSysFolderDlg* aSender,
int hwnd);
__property TFDOnInitializedEvent OnInitialized;
```

Description
This event is called after the browse dialog is created, but before it is shown. You have an opportunity to adjust the initial selection, set the status text or change the enabled state of the Ok button.

See Also

OnSelChanged event
OkEnabled property
StatusText property

OnSelChanged event

See Also

Applies to
TPTSysFolderDlg

Delphi Declaration

```
TFDOnSelChangedEvent = procedure( aSender: TPTSysFolderDlg;  hwnd: HWND;  pidl:
PItemIdList ) of object;

property OnSelChanged: TPTOnSelChangedEvent;
```

C++Builder Declaration

```
typedef void __fastcall (__closure *TFDOnSelChangedEvent)(TPTSysFolderDlg* aSender,
int hwnd, Uptshell95::PItemIDList pidl);

__property TFDOnSelChangedEvent OnSelChanged;
```

Description
This event is called when a selection is made in the browse tree view. You can adjust the selection, set the status text or change the enabled state of the OK button. If you adjust the selection, then this event will be called again - recursively - so you must set some sort of flag to avoid infinite recursion problems.

See Also

OnInitialized event
OkEnabled property
StatusText property

Execute method

Applies to
TPTSysFolderDlg

Delphi Declaration
```
function Execute: Boolean;
```

C++Builder Declaration
```
bool __fastcall Execute(void);
```

Description
The Execute method displays the dialog box in the application and returns *true* when the user closes the dialog box by choosing the OK button. If the user chooses Cancel or closes the dialog box by using the system menu, Execute returns *false*.

UPTShell95 Unit

UPTShell95 unit

Delphi 2 provides access to some of the functionality of SHELL32.DLL via the ShellApi unit, but crucial functionality is not supported. The unit UPTShell95 rectifies this and also provides a set of utilities to simplify access to many of the features of SHELL32.DLL.

Delphi 2.01 and 3 do provide a much extended ShellApi unit, but to retain compatibility with Delphi 2.0 the Shell Control Pack units all use the declarations found in UPTShell95 in preference to ShellApi.

The functions implemented in SHELL32.DLL which are declared in UPTShell95 are:

- SHAddToRecentDocs
- SHBrowseForFolder
- SHChangeNotify
- SHGetDataFromIDList
- SHGetDesktopFolder
- SHGetMalloc
- SHGetPathFromIDList
- SHGetInstanceExplorer
- SHGetSpecialFolderLocation
- SHLoadInProc

SHAddToRecentDocs function

Unit
UPTShell95

Delphi Declaration
```
procedure SHAddToRecentDocs( flags: UINT;  p: Pointer );
```

C++Builder Declaration
```
extern "C" void __stdcall SHAddToRecentDocs( int flags,  void* p );
```

Description
Adds a document to the shell's list of recently used documents or clears all documents from the list. The user gains access to the list through the Start menu of the Windows taskbar.

Flags can be one of the following:

SHARD_PATH	*p* is the address of a path string.
SHARD_PIDL	*p* is the address of an item identifier list.

If *p* is **nil** then all documents are cleared from the list.

SHGetDataFromIDList function

Unit
UPTShell95

Delphi Declaration

```
function SHGetDataFromIDList( pshf: IShellFolder;   // Parent
                              pidl: PItemIdList;    // Item of Parent to process
                              nFormat: Integer;     // Specifies a format
                              p: Pointer;           // Points to output buffer
                              cb: Integer           // Size of buffer passed in
                            ): HResult;
```

C++Builder Declaration
Not declared in C++Builder version.

Description

pshf	The IshellFolder interface of the parent folder.
pidl	The subfolder relative to *pshf*.
nFormat	Can be SHGDFIL_FINDDATA (used for file system objects) or SHGDFIL_NETRESOURCE (used for network resources).
pv	Pointer to a TWin32FindData structured (for SHGDFIL_FINDDATA) or a TNetResource structure (for SHGDFIL_NETRESOURCE)
cb	Should be SizeOf(TWin32FindData) for SHGDFIL_FINDDATA or SizeOf(TNetResource)+1024 for SHGDFIL_NETRESOURCE.

The function succeeds with a return value of NOERROR. If the arguments are not compatible, E_INVALIDARG is returned.

If *nFormat* Is SHGDFIL_NETRESOURCE the function will have sufficient buffer space or not. If the buffer is large enough the string entries for network name, local name, provider and comments are placed into the buffer. If the buffer is not large enough, only the non-string entries of TNetResource will be valid.

Example

```
function GetRemoteName( parent: IShellFolder;  pidl: PItemIdList ): String;
var tmpnres: packed record
      nres: TNetResource;
      buffer: array[0..1023] of Byte;
    end;
begin
  result := '';
  if Succeeded(SHGetDataFromIDList( parent, pidl, SHGDFIL_NETRESOURCE,
                                    @tmpnres, Sizeof(tmpnres) ))
  then
    if Assigned(tmpnres.nres.lpRemoteName) then
      result := String(tmpnres.nres.lpRemoteName);
end;
```

SHGetDesktopFolder function

Unit
UPTShell95

Delphi Declaration
```
function SHGetDesktopFolder( var i: IShellFolder ): HResult;
```

C++Builder Declaration
```
extern "C" long __stdcall SHGetDesktopFolder(IShellFolder* &i);
```

Description
Retrieves the IShellFolder interface for desktop. The desktop is the root of the namespace.

Returns S_OK if successful, otherwise returns an OLE error code. You can convert the code to a string with the SysErrorMessage function.

SHGetMalloc function

Unit
UPTShell95

Delphi Declaration
```
function SHGetMalloc( i: IMalloc ): HResult;
```

C++Builder Declaration
```
extern "C" long __stdcall SHGetMalloc(Ole2::IMalloc* &p0);
```

Description
Retrieves a pointer to the shell's IMalloc interface. You must use this interface to allocate and free memory that will be used by the shell. Typically this interface is used to free item id lists and POLESTR structures returned from many shell interface methods.

Returns S_OK if successful, otherwise returns an OLE error code. You can convert the code to a string with the SysErrorMessage function.

SHGetPathFromIDList function

Unit
UPTShell95

Delphi Declaration
```
function SHGetPathFromIDList( pidl: PItemIdList;  pszPath: PChar ): Bool;
```

C++Builder Declaration
```
extern "C" unsigned __stdcall SHGetPathFromIDList(PItemIDList pidl, char* pszPath);
```

Description
Converts an item identifier list to a file system path.

pidl is an item identifier list to a file-system element, relative to the root of the shell namespace - the desktop.

Returns *true* if successful, *false* otherwise - typically when *pidl* is invalid or doesn't reference a namespace object.

You will normally call ShellGetPathFromIdList instead of this function, as it simply returns a Delphi string instead of PChar.

UPTShellUtils Unit

UPTShellUtils unit

Description
Implements low-level utilities useful for dealing with shell interfaces and structures. Also includes utilities for creating and resolving shortcuts.

Routines

ShellMemAlloc
ShellMemFree
ShellMemRealloc

GetModuleVersion

ShellFindCSIDLFromIdList
ShellGetDisplayPathName
ShellGetFolderFromIdList
ShellGetFriendlyNameFromIdList
ShellGetIconIndex
ShellGetIconIndexFromExt
ShellGetIconIndexFromPath
ShellGetIdListFromPath
ShellGetPathFromIdList
ShellGetSpecialFolderIconIndex
ShellGetSpecialFolderPath
ShellGetSystemImageList

CreateShortcut
CreateQuickShortcut
ResolveShortcut

CopyIdList
CompareAbsIdLists
ConcatIdLists
IdListLen

IsWin95
IsWinNT
HasWin95Shell

Types

TPTModuleVersion

Constants

Folder command strings
COMCTL32.DLL versions

See Also
UPTShell95

ShellMemAlloc function

See Also

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function ShellMemAlloc( size: Cardinal ): Pointer;
```

C++Builder Declaration
```
extern void* __fastcall ShellMemAlloc(Cardinal size);
```

Description
Allocates a block of memory using the shell's allocator.

See Also

ShellMemFree function
ShellMemRealloc function
SHGetMalloc function

ShellMemFree procedure

See Also

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration

```
procedure ShellMemFree( p: Pointer );
```

C++Builder Declaration

```
extern void* __fastcall ShellMemFree( void* p );
```

Description
Frees a block of memory previously allocated using the shell's allocator.

See Also

ShellMemAlloc function
ShellMemRealloc function
SHGetMalloc function

ShellMemRealloc function

See Also

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function ShellMemRealloc( p: Pointer;  newsize: Cardinal ): Pointer;
```

C++Builder Declaration
```
extern void* __fastcall ShellMemRealloc( void* p, Cardinal newsize );
```

Description
Reallocates a block of memory previously allocated using the shell's allocator.

See Also

ShellMemFree function
ShellMemAlloc function
SHGetMalloc function

GetModuleVersion function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function GetModuleVersion( const aModuleName: String;  var {out} aVersion:
     TPTModuleVersion ): Boolean;
```

C++Builder Declaration
```
extern bool __fastcall GetVersionNumber( const System::AnsiString aModuleName,
     TPTModuleVersion& aVersion );
```

Description
This function retrieves the module version information for the given module. *aModuleName* can specify a module on the normal module search path, or a fully qualified pathname.

If the function returns *true* then *aVersion* contains the 64-bit version number for the given module. If it returns *false* then *aVersion* is undefined. Use *GetLastError* to retrieve extended error information.

Example
This example retrieves and displays the version information for COMCTL32.DLL. See the comtctl32.dll versions topic for more comctl32.dll information.

Delphi
```
procedure TestGetModule;
var v: TPTModuleVersion;
begin
  if GetModuleVersion('comctl32.dll', v) then
    ShowMessage( Format('%d.%d.%d.%d',[v.w4, v.w3, v.w2, v.w1]) )
  else
    ShowMessage( 'GetModuleVersion failed!' );
end;
```

C++Builder
```
void __fastcall TestGetModule()
{
  TPTModuleVersion v;
  if (GetModuleVersion("comctl32.dll", v)) {
    char s[80];
    sprintf( "%d.%d.%d.%d", v.w4, v.w3, v.w2, v.w1 );
    ShowMessage( AnsiString(s) );
  }
  else
    ShowMessage( "GetModuleVersion failed!" );
}
```

TPTModuleVersion type

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration

```
TPTModuleVersion = packed record
    case Integer of
      0: (w1, w2, w3, w4: Word);  // Higher number means more significant -
w4=major, w3=minor etc.
      1: (dw1, dw2: Integer);
      2: (asComp: Comp);          // Treat as a single 64-bit integer
      3: (_1, _2, minor, major: Word);
      4: (_3, version: Integer);
  end;
```

C++Builder Declaration

```
union TPTModuleVersion {
  struct { WORD w1; WORD w2; WORD w3; WORD w4; };
  struct { DWORD dw1; DWORD dw2 };
  struct { WORD _1; WORD _2; WORD minor; WORD major; };
  struct { DWORD _3;  DWORD version );
};
```

Description
This type is used by the GetModuleVersion function to retrieve the 64-bit version number for a module in such a way as the important information like *version*, *minor* and *major* are easily accessible.

COMCTL32.DLL versions

Delphi Declaration

```
const
  COMCTL32_VER580 = (5 shl 16) or 80;         // IE5 version
  COMCTL32_VER472 = (4 shl 16) or 72;         // IE4.01 version
  COMCTL32_VER471 = (4 shl 16) or 71;         // IE4 version
  COMCTL32_VER470 = (4 shl 16) or 70;         // IE3 version
  COMCTL32_VER400 = (4 shl 16) or 00;         // Win95 first release version

var
  COMCTL32_VER: TPTModuleVersion;  // Current version of comctl32
```

C++Builder Declaration

```
#define COMCTL32_VER580 (327760)  // IE5 version
#define COMCTL32_VER472 (262216)  // IE4.01 version
#define COMCTL32_VER471 (262215)  // IE4 version
#define COMCTL32_VER470 (262214)  // IE3 version
#define COMCTL32_VER400 (262144)  // Win95 first release version

extern TPTModuleVersion COMCTL32_VER; // Current ver of comctl32
```

Description

The COMCTL32_VER variable is assigned in the initialisation part of the UPTShellUtils unit to the version number of the system comctl32.dll module.

Constants such as COMCTL32_VER471 and COMCTL32_VER470 can be compared against *COMCTL32_VER.version* to check the capabilities of the custom controls contained in comctl32.dll. It is also a good indicator of the installed version of Internet Explorer.

Example

This example shows how to use the TPTModuleVersion type to easily compare comctl32.dll versions.

Delphi

```
if COMCTL32_VER.version <= COMCTL32_VER470 then
  ShowMessage( 'Internet Explorer 3 or earlier' )
else
  ShowMessage( 'Internet Explorer 4 or later' );
```

C++Builder

```
if (COMCTL32_VER.version <= COMCTL32_VER470)
  ShowMessage( "Internet Explorer 3 or earlier" );
else
  ShowMessage( "Internet Explorer 4 or later" );
```

ShellFindCSIDLFromIdList function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function ShellFindCSIDLFromIdList( aIdList: PItemIdList ): TCSIDL;
```

C++Builder Declaration
```
extern TCSIDL __fastcall ShellFindCSIDLFromIdList( Uptshell95::PItemIDList aPidl );
```

Description
Given an item id list, this function will return the TCSIDL constant that matches. If not match is found, then *csidlNone* is returned.

Note that this function is very slow. It simply checks *aIdList* against each TCSIDL until it finds a match, or all TCSIDL values are processed.

ShellGetDisplayPathName function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function ShellGetDisplayPathName( const pathname: String ): String;
```

C++Builder Declaration
```
extern System::AnsiString __fastcall ShellGetDisplayPathname( System::AnsiString
     pathname);
```

Description
Files and directory names are case-insensitive, case-retained in Windows 95 and NT. This presents a programming problem when the user is allowed to key-in the name of a file system object. As long as that name is used internally, there is no problem - since case is functionally irrelevant. If the name is displayed to the user however, they expect to see the proper casing of the pathname.

For example:

- A file exists named "C:\My Documents\Sales Report 1996.xls"
- The user keys in this name somewhere - maybe in an edit field or as a command line parameter.
- They can't remember the casing, of course, so they type:
  "C:\MY DOCUMENTS\SALES REPORT 1996.XLS"
- If we want to display that name back to the user (say in the form caption) it is not obvious how we should obtain the properly cased name.
- A call to ShellGetDisplayPathName will return the properly cased name.

If the file doesn't exist or any other error occurs, the result is an empty string.

ShellGetFolderFromIdList function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function ShellGetFolderFromIdList( p: PItemIdList;  var ish: IShellFolder ):
HResult;
```

C++Builder Declaration
```
extern long __fastcall ShellGetFolderFromIdList(Uptshell95::PItemIDList p,
     Uptshell95::IShellFolder* &ish);
```

Description
Given the item id list *p* (relative to the desktop) returns the associated IShellFolder interface.

Returns S_OK if successful, otherwise returns an OLE error code. You can convert the code to a string with the SysErrorMessage function.

ShellGetFriendlyNameFromIdList

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration

```
type TPTFriendlyNameFlags = (ptfnNormal, ptfnInFolder, ptfnForParsing);

function ShellGetFriendlyNameFromIdList( ishf: IShellFolder;
                                         pidl: PItemIdList;
                                         flags: TPTFriendlyNameFlags ): String;
```

C++Builder Declaration

```
enum TPTFriendlyNameFlags { ptfnNormal, ptfnInFolder, ptfnForParsing };

extern System::AnsiString __fastcall
     ShellGetFriendlyNameFromIdList(Uptshell95::IShellFolder* ishf,
     Uptshell95::PItemIDList pidl, TPTFriendlyNameFlags flags);
```

Description
Retrieves the display name for the specified file object or subfolder. The *pidl* parameter specifies the item for which the name will be retrieved. If *ishf* is *nil* then *pidl* must be an absolute id list, otherwise it must be relative to *ishf*. The name can be retrieved in a number of formats using different values in the *flags* parameter.

Code	Usage
ptfnNormal	Default display name that is suitable for a file object displayed by itself, as shown in the following examples.
ptfnInFolder	Display name that is suitable for a file object displayed within its respective folder, as shown in the following examples.
ptfnForParsing	Display name that can be passed to the ParseDisplayName method of the parent folder's IshellFolder object.

Code	File system path	Display name	Notes
ptfnNormal	C:\Windows\file.txt	file	If not showing extensions.
ptfnInFolder	C:\Windows\file.txt	file	Where C has the volume name My Drive.
	\\Computer\Share	Share	
	C:\	My Drive (C)	
ptfnForParsing	C:\Windows\file.txt	C:\Windows\file.txt	
	\\Computer\Share	\\Computer\Share	
	C:\	C:\	

ShellGetIconIndex function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function ShellGetIconIndex( absIdList: PItemIdList;  uFlags: DWORD ): Integer;
```

C++Builder Declaration
```
extern int __fastcall ShellGetIconIndex( Uptshell95::PItemIDList,  unsigned
     uFlags );
```

Description
Returns the index into the system image list of the icon associated with the *absIdList* namespace item.

Use this function in conjunction with ShellGetSystemImageList to get the index of the icon for individual shell items.

Flags applicable to *uFlags*

SHGFI_SMALLICON	Use this flag to get the index of icons from the small image list - there is no difference between the index of icons in the small and large image lists.
SHGFI_LARGEICON	Use this flag to get the index of icons from the large image list - there is no difference between the index of icons in the small and large image lists.
SHGFI_OPENICON	Many shell items have the concept of open and closed states (folders for example). Specify this flag to retrieve the icon for the open state. Leave this flag out to get the icon for the closed state.

Example
This example retrieves the open and closed icon states for the given item id list.

Delphi
```
procedure GetOpenCloseIcons( p: PItemIDList;  var openicon, closedicon:
Integer );
begin
  openicon := ShellGetIconIndex( p, SHGFI_SMALLICON or SHGFI_OPENICON );
  closedicon := ShellGetIconIndex( p, SHGFI_SMALLICON );
end;
```

C++Builder
```
void __fastcall GetOpenCloseIcons( PItemIDList p, int& openicon, int& closedicon )
{
  openicon = Uptshellutils::ShellGetIconIndex( p, SHGFI_SMALLICON | SHGFI_OPENICON );
  closedicon = Uptshellutils::ShellGetIconIndex( p, SHGFI_SMALLICON );
}
```

ShellGetIconIndexFromExt function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function ShellGetIconIndexFromExt( const ext: String;  uFlags: DWORD ): Integer;
```

C++Builder Declaration
```
extern int __fastcall ShellGetIconIndexFromExt( const System::AnsiString ext,
     unsigned uFlags );
```

Description
Returns the index into the system image list of the icon associated with the given file extension. Include the leading period, as in '.txt'.

Use this function in conjunction with ShellGetSystemImageList to get the index of the icon for particular file extensions.

See ShellGetIconIndex for a list of allowable values for the *uFlags* parameter.

Example
This example retrieves the icon for text files.

Delphi
```
function GetTextIcon: Integer;
begin
  result := ShellGetIconIndexFromExt( '.txt', 0 );
end;
```

C++Builder
```
int __fastcall GetTextIcon()
{
  return ShellGetIconIndexFromExt( ".txt", 0 );
}
```

ShellGetIconIndexFromPath function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function ShellGetIconIndexFromPath(
      const path: String;  uFlags: DWORD ): Integer;
```

C++Builder Declaration
```
extern int __fastcall ShellGetIconIndexFromPath(
      const System::AnsiString path,  unsigned uFlags );
```

Description
Returns the index into the system image list of the icon associated with the given file or folder path.

Use this function in conjunction with ShellGetSystemImageList to get the index of the icon for individual folders and files without first obtaining their item id list.

See ShellGetIconIndex for a list of allowable values for the *uFlags* parameter.

Example
This example retrieves the open and closed icon states for folders.

Delphi
```
procedure GetFolderIcons( var openicon, closedicon: Integer );
begin
  openicon := ShellGetIconIndexFromPath( 'c:\', SHGFI_OPENICON );
  closedicon := ShellGetIconIndexFromPath( 'c:\', 0 );
end;
```

C++Builder
```
void __fastcall GetOpenCloseIcons( PitemIDList p, int& openicon, int& closedicon )
{
  openicon = Uptshellutils::ShellGetIconIndexFromPath( "c:\\", SHGFI_OPENICON );
  closedicon = Uptshellutils::ShellGetIconIndexFromPath( "c:\\", 0 );
}
```

ShellGetIdListFromPath function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function ShellGetIdListFromPath( const path: String;  var p: PItemIdList ): HResult;
```

C++Builder Declaration
```
extern long __fastcall ShellGetIdListFromPath(const System::AnsiString path,
     Uptshell95::PItemIDList & p);
```

Description
Given the pathname of a filesystem folder or file, returns the item id list *p* (relative to the desktop).

To convert an item id list to a pathname use ShellGetPathFromIdList.

You must free the returned item id list with ShellMemFree or the shell's IMalloc interface.

Returns S_OK if successful, otherwise returns an OLE error code. You can convert the code to a string with the SysErrorMessage function.

ShellGetPathFromIdList function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function ShellGetPathFromIdList( pidl: PItemIdList ): String;
```

C++Builder Declaration
```
extern System::AnsiString __fastcall ShellGetPathFromIdList(Uptshell95::PItemIDList
pidl);
```

Description
Converts an item id list to a pathname. Returns an empty string on error.

To convert a pathname to an item id list use ShellGetIdListFromPath.

ShellGetSpecialFolderIconIndex function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function ShellGetSpecialFolderIconIndex(csidl: TCSIDL;  uFlags: DWORD): String;
```

C++Builder Declaration
```
extern int __fastcall ShellGetSpecialFolderIconIndex(TCSIDL csidl,  unsigned
uFlags);
```

Description
Returns the index into the system image list of the icon associated with the special folder specified by *csid*.

Use this function in conjunction with ShellGetSystemImageList to get the index of the icon for special shell folders.

Flags applicable to *uFlags*

SHGFI_SMALLICON	Use this flag to get the index of icons from the small image list - there is no difference between the index of icons in the small and large image lists.
SHGFI_LARGEICON	Use this flag to get the index of icons from the large image list - there is no difference between the index of icons in the small and large image lists.
SHGFI_OPENICON	Many shell items have the concept of open and closed states (folders for example). Specify this flag to retrieve the icon for the open state. Leave this flag out to get the icon for the closed state.

ShellGetSpecialFolderPath function

See Also

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration

```
function ShellGetSpecialFolderPath( ahwnd: HWND;  csidl: TCSIDL ): String;
```

C++Builder Declaration

```
extern System::AnsiString __fastcall ShellGetSpecialFolderPath(int ahwnd, TCSIDL
csidl);
```

Description
Returns the pathname of the special folder. *csidl* must be one of the system folder constants.

If acquiring the pathname requires a user interface to be presented (such as a password request dialog) then *ahwnd* is used as the parent of the dialog. If *ahwnd* is 0 then if an interface is required the function will simply fail.

If an error occurs the function returns an empty string.

See Also

SHGetSpecialFolderLocation function

ShellGetSystemImageList function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration

```
type TPTShellIconSize = (ptsizSmall, ptsizLarge);

function ShellGetSystemImageList( aSize: TPTShellIconSize ): THandle;
```

C++Builder Declaration

```
enum TPTShellIconSize { ptsizSmall, ptsizLarge };

extern int __fastcall ShellGetSystemImageList(TPTShellIconSize aSize);
```

Description
Returns the handle of the selected system image list, or 0 on error.

The system maintains two image list caches of every icon used in the shell — one for large icons, one for small. Many shell oriented functions and interfaces return an index into this list for shell objects. This removes any need for you to load and cache shell icons yourself, just use an index into the system list and let the system handle it. The same index can be used for both small and large image lists.

Example
Assume *SmallList* and *LargeList* are of members of type TImageList.

Delphi

```
procedure TMyForm.InitImageLists;
begin
  SmallList.ShareImages := true;  // This is ESSENTIAL!
  SmallList.Handle := ShellGetSystemImageList( ptsizSmall );

  LargeList.ShareImages := true;
  LargeList.Handle := ShellGetSystemImageList( ptsizLarge );
end;
```

C++Builder

```
void __fastcall TMyForm::InitImageLists()
{
  SmallList->ShareImages = true;  // This is ESSENTIAL!
  SmallList->Handle = Uptshellutils::ShellGetSystemImageList( ptsizSmall );
  LargeList->ShareImages = true;
  LargeList->Handle = Uptshellutils::ShellGetSystemImageList( ptsizLarge );
}
```

You must set your image list component to `ShareImages=true`, otherwise when the imagelist is deleted it will take the system image list with it. Explorer looks kinda funny after you have deleted its icon image list!

CreateShortcut function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration

```
function CreateShortcut( const linkPathName: String;  const linkData: TLinkData ):
HResult;
```

C++Builder Declaration

```
extern long __fastcall CreateShortcut(const System::AnsiString linkPathName, const
TLinkData &linkData);
```

Description
This function provides access to all shortcut features via the TLinkData record. You typically don't need all this functionality. Most of the time, CreateQuickShortcut is a better alternative.

LinkPathName	is the name of the shortcut file to be created.
LinkData	is an input record used to access function options.

Returns S_OK if successful, otherwise returns an OLE error code. You can convert the code to a string with the SysErrorMessage function.

TLinkData record

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration

```
type TLinkDataOption = (ldoUseDesc, ldoUseArgs, ldoUseIcon,
                        ldoUseWorkDir, ldoUseHotKey, ldoUseShowCmd);
     TLinkDataOptions = set of TLinkDataOption;

     TLinkData = record
       // Mandatory members for creating shortcuts
        pathName: String;     // Pathname of original object
        options: TLinkDataOptions; // Set of flags indicating optional member usage

       // Optional members
        desc: String;         // Description of link file (its filename for example)
        args: String;         // Command-line arguments
        iconPath: String;     // Pathname of file containing the icon
        iconIndex: Integer;   // Index of icon in 'iconPath'.  -ve values are
resource ids.
        workingDir: String;   // Working directory when process starts
        showCmd: Integer;     // How to show the initial window
        hotkey: Word;         // Hot key for the link
        noIU: Boolean;        // Prevent any error or search dialogs from displaying

       // Members used by ResolveShortcut
        idList: PItemIdList;
        w32fd: TWin32FindData;
     end;
```

C++Builder Declaration

```
enum TlinkDataOption { ldoUseDesc, ldoUseArgs, ldoUseIcon, ldoUseWorkDir, ldoUseHotKey,
ldoUseShowCmd };

typedef Set<TlinkDataOption, ldoUseDesc, ldoUseShowCmd>  TLinkDataOptions;

struct TlinkData
{
// Mandatory members for creating shortcuts
 System::AnsiString pathName; // Pathname of original object
 TlinkDataOptions options;    // Set of flags indicating optional member usage

// Optional members
 System::AnsiString desc;     // Description of link file (its filename for example)
 System::AnsiString args;     // Command-line arguments
 System::AnsiString iconPath; // Pathname of file containing the icon
 int iconIndex; // Index of icon in 'iconPath'.  -ve values are resource ids.
 System::AnsiString workingDir; // Working directory when process starts
 int showCmd;                 // How to show the initial window
 Word hotkey;                 // Hot key for the link
 bool noUI;                   // Prevent any error or search dialogs from displaying

// Members used by ResolveShortcut
 Uptshell95::TitemIDList *idList;
 _WIN32_FIND_DATAA w32fd;
};
```

Description
To create a shortcut, you must fill in the mandatory members and call CreateShortcut. Use the *options* member to specify which optional members you have filled in.

Valid values for *showCmd* are SW_SHOWNORMAL, SW_SHOWMAXIMIZED and SW_SHOWMINNOACTIVE.

To resolve a shortcut, you can simply call ResolveShortcut and pass an uninitialised TLinkData structure.

CreateQuickShortcut function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function CreateQuickShortcut( const linkPathName, targetPathName: String ): HResult;
```

C++Builder Declaration
```
extern long __fastcall CreateQuickShortcut( const System::AnsiString linkPathName,
const System::AnsiString targetPathName );
```

Description
This is a version of CreateShortcut with reduced flexibility, but easier to use.

LinkPathName is the name of the shortcut file to be created.
TargetPathName is the object to which the new shortcut will point.

Returns S_OK if successful, otherwise returns an OLE error code. You can convert the code to a string with the SysErrorMessage function.

Example
This example creates a shortcut on the desktop.

Delphi 2
```
var hr: HResult;
. . .
hr := CreateQuickShortcut(ShellGetSpecialFolderPath(0, csidlDesktop) + '\My
Shortcut.lnk', 'myapp.exe');
if Ole2.Failed(hr) then raise Exception.Create( SysErrorMessage(hr) );
```

Delphi 3
```
OleCheck( CreateQuickShortcut(ShellGetSpecialFolderPath(0, csidlDesktop) + '\My
Shortcut.lnk', 'myapp.exe') );
```

C++Builder
```
HRESULT hr = Uptshellutils::CreateQuickShortcut( ShellGetSpecialFolderPath(0,
     csidlDesktop) + "\My Shorcut.lnk", "myapp.exe");
if (Ole2::Failed(hr))
  throw new Exception( SysErrorMessage(hr) );
```

ResolveShortcut function

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function ResolveShortcut( const linkPathName: String;  var linkData:
TLinkDataTop_TLinkData;  fWantIdList: Boolean ): HResult;
```

C++Builder Declaration
```
extern long __fastcall ResolveShortcut(const System::AnsiString linkPathName,
TLinkData &linkData, bool fWantIdList);
```

Description
This functions fills in the passed *linkData* structure with given shortcut's details.

If the *fWantIdList* parameter is *true* then the idList member of the linkData structure is set to the id list of the item referenced by the shortcut. You must remember to free this item with the ShellMemFree procedure.

Returns S_OK if successful, otherwise returns an OLE error code. You can convert the code to a string with the SysErrorMessage function.

Example
This example resolves a shortcut and displays a few of its parameters in a message box.

Delphi
```
procedure ResolveIt;
var dwres: DWORD;
    linkData: TLinkData;
begin
  dwres := ResolveShortcut('c:\mytest.lnk', linkData, true);
  if Failed(dwres) then raise Exception.Create(SysErrorMessage(dwres));
  ShowMessage( Format('Points to: %s'#13'Icon path: %s'#13'Working Dir: %s',
                      [linkData.pathName, linkData.iconPath,
linkData.workingDir]) );
  ShellMemFree( linkData.idList );
end;
```

C++Builder
```
void __fastcall ResolveIt()
{
  TLinkData linkData;
  HRESULT hr = Uptshellutils::ResolveShortcut( "C:\mytest.lnk", linkData, true );
  if (Ole2::Failed(hr)) throw new Exception( SysErrorMessage(hr) );
  char s[256];
  sprintf( s, "Points to: %s\nIcon path: %s\nWorking Dir:%s", linkData.pathName,
     linkData.iconPath, linkData.workingDir );
  ShowMessage( s );
  ShellMemFree( linkData.idList );
}
```

CopyIdList function

See Also

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function CopyIdList( ishm: IMalloc;  pidl: PItemIdList ): PItemIdList;
```

C++Builder Declaration
```
extern PItemIdList __fastcall CopyIdList( IMalloc* ishm, PItemIdList pidl );
```

Description
Copies an item id list using the given allocator. The *ishm* parameter must be the shell allocator interface returned with the SHGetMalloc function or nil. If nil, then the SHGetMalloc is used internally to aquire the shell's IMalloc interface. Performance is reduced if you pass nil.

You must free the returned item id list with the shell allocator, or ShellMemFree.

See Also

ConcatIdLists function
IdListLen function
SHGetMalloc function
ShellMemFree function

Item ID Lists

CompareAbsIdLists function

See Also

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function CompareAbsIdLists( pidl1, pidl2: PItemIdList ): Integer;
```

C++Builder Declaration
```
extern int __fastcall CompareAbsIdLists( PItemIdList pidl1, PItemIdList pidl2 );
```

Description
Compares to absolute item id lists. Absolute id lists are relative to the desktop folder.

Returns one of four values:

Code	Meaning
MAXINT	An error occurred.
less than zero	The first item should precede the second (pidl1 < pidl2).
greater than zero (not MAXINT)	The first item should follow the second (pidl1 > pidl2).
zero	The two items are the same (pidl1 = pidl2).

See Also

SHGetDesktopFolder function

Item ID Lists

ConcatIdLists function

See Also

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration

```
function ConcatIdLists( ishm: IMalloc;  aFirst, aSecond: PItemIdList ): PItemIdList;
```

C++Builder Declaration

```
extern PItemIdList __fastcall ConcatIdLists( IMalloc* ishm, PItemIdList aFirst,
PItemIdList aSecond );
```

Description
Concatenates the two item id lists. A new block of memory is allocated using the given *ishm* allocator. This should be the shell allocator acquired with SHGetMalloc or nil. If nil, then SHGetMalloc is used internally to aquire the shell's IMalloc interface. Performance is reduced if you pass nil.

You must free the returned item id list with the shell allocator, or ShellMemFree.

See Also

CopyIdList function
IdListLen function
SHGetMalloc function
ShellMemFree function

Item ID Lists

IdListLen function

See Also

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function IdListLen( pidl: PItemIdList ): Integer;
```

C++Builder Declaration
```
extern int __fastcall IdListLen( PItemIdList pidl );
```

Description
Returns the length of the given item id list in bytes.

pidl can be nil in which case 0 is returned.

See Also

ConcatIdLists function
CopyIdList function

IsWin95 function

See Also

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function IsWin95: Boolean;
```

C++Builder Declaration
```
extern bool __fastcall IsWin95(void);
```

Description
Returns *true* if running under Windows 95.

See Also

IsWinNT function
HasWin95Shell function

IsWinNT function

See Also

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function IsWinNT: Boolean;
```

C++Builder Declaration
```
extern bool __fastcall IsWinNT(void);
```

Description
Returns *true* if running under Windows NT.

See Also

IsWin95 function
HasWin95Shell function

HasWin95Shell function

See Also

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Delphi Declaration
```
function HasWin95Shell: Boolean;
```

C+Builder Declaration
```
extern bool __fastcall HasWin95Shell(void);
```

Description
Returns *true* if running under the 'new' shell introduced in Windows 95. Currently Windows 95 and Windows NT 4.0+ cause this function to return *true*. If this function returns *false*, you should not use any of the Shell Control Pack components or functions.

See Also

IsWin95 function
IsWinNT function

Folder Command Strings

Delphi Unit
UPTShellUtils

C++Builder Header
UPTShellUtils.hpp

Description
Most system folders support string-based command execution. If you tell any system folder to execute the "Properties" command, the folder will display its properties dialog. The UTPShellUtils unit contains some of the most common command strings. The list is not exhaustive.

To execute one of these commands for a folder in the TPTShellTree component use the DoCommandForNode method.

To execute a command for an item in the TPTShellList component use the DoCommandForItem, DoCommandForFolder or DoCommandForAllSelected methods.

Delphi Declarations
```
// -- These commands are available to most folders --
const PTSH_CMDS_DELETE = 'delete';
      PTSH_CMDS_PASTE  = 'paste';
      PTSH_CMDS_CUT    = 'cut';
      PTSH_CMDS_COPY   = 'copy';
      PTSH_CMDS_PROPERTIES = 'properties';
      PTSH_CMDS_EXPLORE = 'explore';  // Opens a Windows explorer
      PTSH_CMDS_OPEN   = 'open';      // Opens a Windows explorer folder-view
      PTSH_CMDS_FIND   = 'find';      // Open the find dialog
      PTSH_CMDS_LINK   = 'link';      // Same as 'Create Shortcut' menu item

// -- Commands used by Dialup Networking
const PTSH_CMDS_DUN_CREATE = 'create';    // Create new connection wizard
      PTSH_CMDS_DUN_CONNECT = 'connect';  // Make a connection
```

C++Builder Declarations
```
#define PTSH_CMDS_DELETE "delete"
#define PTSH_CMDS_PASTE "paste"
#define PTSH_CMDS_CUT "cut"
#define PTSH_CMDS_COPY "copy"
#define PTSH_CMDS_PROPERTIES "properties"
#define PTSH_CMDS_EXPLORE "explore"
#define PTSH_CMDS_OPEN "open"
#define PTSH_CMDS_FIND "find"
#define PTSH_CMDS_LINK "link"
#define PTSH_CMDS_DUN_CREATE "create"
#define PTSH_CMDS_DUN_CONNECT "connect"
```

Types

License

Copyright
This product in whole or in part, including all files, data, and documentation, from here on referred to as "Product" is Copyright © 1996-2001 Plasmatech Software Design, all rights reserved, and is protected by the copyright laws of the State of Victoria and the Commonwealth of Australia, international treaties and all other applicable national or international laws.

Disclaimer
This product and the accompanying files and documentation are sole "as is" and without warranties as to performance or merchantability or any other warranties whether expressed or implied. The user and/or licensee assume the entire risk as to the use of this product. Plasmatech Software Design does not assume liability for the use of this product beyond the original purchase price. In no event will Plasmatech Software Design be liable for additional direct or indirect damages including any lost profits, lost savings, or other incidental or consequential damages arising from any defects, or the use or inability to use this product, even if Plasmatech Software Design has been advised of the possibility of such damages.

Restrictions
You may not use, copy, modify, translate, or transfer the product or any copy except as expressly defined in this agreement. You may not remove or modify any copyright notice. This license agreement may not be modified or supplemented without the written consent of Plasmatech Software Design.

Operating license
You have the non-exclusive right to use the product only by a single person, on a single computer at a time. In group situations, where multiple persons will use the Product, you must purchase an individual license for each member of the group. **A person is considered using the Product if he or she has any components of the Product installed on his or her component palette.** Use over a network is permitted provided that only individuals possessing a license use the Product. This means many individuals can use a single copy of the Product, residing on a server only if each and every user has a license. It is a breach of this agreement to use a networked copy of the Product if you do not possess a valid license.

Distribution license
You must be registered with Plasmatech Software Design to distribute executable or compiled code in accordance with this license. You may not reproduce or distribute copies of any part of the Product, or the Product as a whole, including all source code, help files or code units.

You may not use the Product in any other product that directly or indirectly competes with the Product. Specifically, you may not include this Product as part of any code library, as source code or in compiled form. You may not provide any means by which your users could create, modify, or incorporate any part of this Product into their own products.

Back-up and transfer
You may make one copy of the Product solely for back-up purposes. You may transfer the Product to someone else only of that person agrees with the terms and conditions detailed in this agreement. If you transfer the product you must destroy your copy of it, including all documentation and any back-up copy you may have.

Terms
This license is effective until terminated. You may terminate it by destroying the complete product and any backup copy. This license will also terminate if you fail to comply with any terms or conditions of this agreement. You agree upon such termination to destroy all copies of the Product.

Rights and restrictions
All other rights and restrictions not specifically granted in this license are reserved worldwide by Plasmatech Software Design.

Ordering Information

Pricing

Product	Price in US dollars	
Shell Control Pack with source	$149	per user.
Shell Control Pack DCU only	$79	per user.
Shell Control Pack DCU to source upgrade	$70	per user.
Shell Control Pack Site License (with source)	$1500	unlimited users within a single company.

Every developer in your organisation that has any Shell Control Pack components installed on their component palette must have a license. All licenses in an organisation must be equivalent, i.e. they must be source code **or** DCU only.

Purchase of the "site license" permits any number of users employed by the registered organisation to install and use the shell control pack within 160 kilometres of the registered address. For worldwide licenses contact Plasmatech directly.

Payment Options

Secure Online Ordering via Credit Card
This is by far the most popular form of payment, as it avoids all the inconvenience of currency conversion and snail mail. You can pay with MasterCard, Visa, Amex, Discover or Diners via secure server. Visit the ordering page on our website at http://plasmatech.com/ordering.htm for details.

Telephone Orders
Phone orders should be made during Californian business hours (GMT -8:00), Monday to Saturday. There is a $5 handling charge for telephone orders. Have your credit card and e-mail address details ready, and ensure you have the product name correct.

Telephone Order Number +1 (510) 658-5244

Where '+' is your international dialing prefix. Leave the '+1' out if you are ordering from within the USA or Canada.

Offline payments
Visit the ordering page on our website at http://plasmatech.com/ordering.htm for details.

Support
Purchasing entitles you to unlimited support via e-mail, minor upgrades and bug fix releases indefinitely and major version upgrades for one year.

See Also
Contact Information

Contact Information

E-mail is the preferred way of contacting us - remember that your registration entitles you to unlimited support via e-mail.

Web: http://plasmatech.com

E-mail: General inquiries: info@plasmatech.com

Product support: support@plasmatech.com

Fax: +61 3 9545 1486
Where + is your international dialing prefix (011in the US)

Post: Plasmatech Software Design
69 Remington Drive
Glen Waverley VIC 3150
AUSTRALIA

Glossary Terms

Plasmatech Software Design 2048-bit PGP Public Key

Right-click in this popup and select copy. Paste into a file, such as `Plamatk.asc` and add that file to your PGP keyring.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: 2.6.3i
mQENAzJIkyoAAAEIAMPCwhFrdtKBBpwUf/Tun079j52kWX2fWRlpMpEXkZ1WtX4X
NPhZpzRRBMM5KN/9ln4Rck2wI3uTF3ynG80b0GSmu6Bthcyx9qxPtbilayw/BBAP
C3BMkid4fp/7Dbll3tjqAvEc6KLwMNKKOBqvn2m31buB7TvKj+Xlx3sRd7PQwTsU
mTWUNjrrOeldTPyFHqp5zXHBHAYMaruzgDCJpjeXhVJfBNeUHZCibcwzrmq9JBo0
J4Abibg6dgeBzmPwvd2Fek/6ZVawpD0+XKU4zHDsu6FkkBtuTTI5dGoRSIUCkdJ8
eVUB6DwIW0qfzsst7gWA2xPq543G0Wk5riOPQlUABRG0NlBsYXNtYXRlY2ggU29m
dHdhcmUgRGVzaWduIDxQbGFzbWF0ZWNoQG5lbWVzaXMuY29tLmF1PokBFQMFEDJI
kyrRaTmuI49CVQEBgAsH/ioy+eet7Z0Ko838JhpoRO+8WzD8mKUWiCf65v+yQmXC
nBXmqQQ4EHUNkrpoUcMVztj0pJ3PpvYF7SwFvuShbytB4uQUvKf7sRscru6QkaC0
LlmdyuI6mpTpDhe9PKJx9CPtLNfYPqGDEfAWHiV3g7OVkFOZJW/UVP1t7kuvaYn7
nbkgrowgc9TiSiSpKsi9K7+YjYuiTjNhAFZGljNBTz1PfQSWSX0agtxhROZgkKdg
Gqjncjlmaqyi9GjERMMUa6HG6VTe9gTj2dLczvFNQrblr6KMWaA1vNbyt+q9mP3r
X11xeGss+NMWTMc7UFsxiCfDwrVEJ6wP7b/doy5SB74=
=UQvT
-----END PGP PUBLIC KEY BLOCK-----
```

Item ID Lists

An item identifier (item ID) is a binary data structure that uniquely identifies an object in the object's parent folder. When you need to trace a path to an item from the desktop, you use an item ID list. The user never sees item IDs; display names are shown instead.

The concept of a binary list of items allows the introduction of non-filesystem name space items. Items such as *My Computer*, and *Network Neighbourhood* have no pathname.

Think of your system. From a filesystem perspective you have an A: drive, maybe a B: drive, a number of hard-disk partitions C:, D: etc. maybe some mapped network drives and a CD-ROM. If you were using a command line, where could you go to get a directory of all these things? The answer is you can't. The pathname based filesystem has no concept of *My Computer* (a folder that contains all the drives on your computer). Within a drive, directories are stored hierarchically but the hierarchy stops at the root directory of each drive. You cannot go "up one level" from the root directory of a drive.

Windows 95 extended the filesystem hierarchy "up" two levels by adding non-filesystem folders, to produce an abstract *name space*. The root of the hierarchy is the *Desktop* folder, which is the only folder that has no parent.

If we have extra levels in our name space hierarchy, how do we uniquely identify them? The well-known pathname is no longer sufficient.

Consider the following pathname.

```
C:\Dir1\Dir2\file1.txt
```

If we represent each item ID brace-delimited, the item ID list would be:

```
[My Computer] [C:] [Dir1] [Dir2] [file1.txt]
```

Win95 Disabled Look

When standard controls such as text, group box and buttons are disabled, the caption appears grey on 16 colour WIn95 displays and bevelled grey at higher colour depths on Win95 or any colour depth on WinNT 4.0. While many Delphi 2.0 controls don't follow this standard, all Plasmatech controls do.

Delphi Label	Win95 / Plasmatech Label
Enabled Disabled	Enabled Disabled

System Folder Constants (TCSIDL type)

Delphi Unit	C++Builder Header
UPTShellUtils	UPTShellUtils.hpp

The following constants can be used to refer to system folders in various functions and components of the Shell Control Pack. The system folder constants map to the location of the particular system folder on the current machine.

csidlBitBucket		Recycle bin — file system directory containing file objects in the user's recycle bin. The location of this directory is not in the registry; it is marked with the hidden and system attributes to prevent the user from moving or deleting it.
csidlControls		Control Panel — virtual folder containing icons for the control panel applications.
csidlDesktop		Windows desktop — virtual folder at the root of the name space.
csidlDesktopDirectory		File system directory used to physically store file objects on the desktop (not to be confused with the desktop folder itself).
csidlDrives		My Computer — virtual folder containing everything on the local computer: storage devices, printers, and Control Panel. The folder may also contain mapped network drives.
csidlFonts		Virtual folder containing fonts.
csidlFavorites		File system directory that contains the user's favourites.
csidlNethood		File system directory containing objects that appear in the network neighbourhood.
csidlNetwork		Network Neighbourhood — virtual folder representing the top level of the network hierarchy.
csidlPersonal		File system directory that serves as a common repository for documents. Typically the My Documents folder.
csidlPrinters		Printers folder — virtual folder containing installed printers.
csidlPrograms		File system directory that contains the user's program groups (which are also file system directories).
csidlRecent		File system directory that contains the user's most recently used documents.
csidlSendTo		File system directory that contains Send To menu items.
csidlStartMenu		File system directory containing **Start** menu items.
csidlStartup		File system directory that corresponds to the user's Startup program group.
csidlTemplates		File system directory that serves as a common repository for document templates.
csidlCommonStartMenu		File system directory that contains the programs and folders that appear on the **Start** menu for all users.
csidlCommonPrograms		File system directory that contains the directories for the common

	program groups that appear on the **Start** menu for all users.
csidlCommonStartup	File system directory that contains the programs that appear in the Startup folder for all users. The system starts these programs whenever any user logs on to Windows NT or starts up Windows 95.
csidlCommonDesktopDirectory	File system directory that contains files and folders that appear on the desktop for all users.
csidlAppData	The meaning of this item is not documented.
csidlPrintHood	The meaning of this item is not documented.

Shell Locator Editor

There are three methods of identifying namespace entities, which the Shell Locator Editor allows you to set.

1. CSIDL - Constant System identifiers
Using the CSIDL method, you choose from one of the predefined system constants.

These constants map to commonly referenced system folders. They are the most useful way of referencing shell folders since they are machine independent. The constant *csidlPrograms* will reference the user's Programs folder regardless of where it is located. If you had used one of the other two mechanism of locating namespace entities, the folder might not exist on another machine.

For example, Machine 1 has Windows installed at C:\WINDOWS so the *csidlPrograms* identifier will resolve to the path "C:\WINDOWS\Start Menu\Programs". Machine 2 has Windows installed at E:\WIN95 so the *csidlPrograms* identifier will resolve to the path "E:\WIN95\Start Menu\Programs". Non-English languages are also handled correctly.

2. Pathname
You can specify a folder by name in the provided edit field. Only filesystem directories can be specified. This is useful if you know the pathname will not change and can therefore be fixed at design time.

3. Item id list
You can choose a namespace entity using a popup tree-view. Any namespace entities, such as "Network Neighbourhood" or "My Computer" are acceptable, as well as file system directories. "Network Neighbourhood" or "My Computer" are portable across different Windows installations, but file system directories might not be.

Legend

 Read only

 Published

D2 Delphi 2 specific

 Delphi 3 specific

C1 C++ Builder 1 specific

 C++ Builder 3 specific

Using shell control automatic interaction

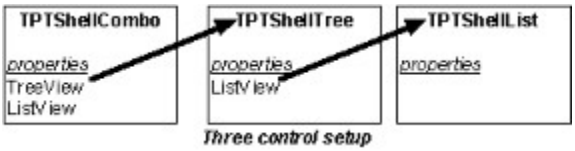
The interactions between TPTShellTree, TPTShellList and TPTShellCombo can be complex. They can work together to provide a no-code explorer-like interface, or they can be used singly for more specialised requirements.

A common combination is to combine TPTShellCombo and TPTShellList to provide an interface similar to the system File Open/Save dialog.

The linkage properties are:

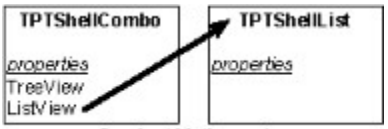
TPTShellCombo	**TPTShellTree**	**TPTShellList**
ShellTree	ShellList	(none)
ShellList		

When all three components are used together, you should set the ShellTree property of TPTShellCombo to your Tree and the ShellList property of TPTShellTree to your List. Leave the ShellList property of TPTShellCombo blank in this case.



Three control setup

When using TPTShellCombo together with TPTShellList you should set the ShellList property of TPTShellCombo to your list. Leave the ShellTree property of TPTShellCombo blank in this case.



Combo / Listbox setup

TPTShellTree and TPTShellList can be used together in a similar way.

You need not rely on the automatic interactions. You can leave the ShellList and ShellTree properties *nil* and process the interactions yourself.

All three controls can stand alone by leaving the above properties blank. The TPTShellCombo is of little use on its own.

The Shell Control Pack now includes an abstract OLE Data Object infrastructure that makes dealing with drag sources, drop targets and clipboard transfer much easier.

UPTOleDataTransfer unit

Classes

- TPTOleDataObject
- TPTOleDataSource
- TPTOleDropTarget

Routines

- Create_CFTEXT_HGlobal
- Create_CFHDROP_HGlobal
- Create_CFIDLIST_HGlobal

- Convert_CFTEXT_String
- DuplicateHGlobal

Constants

- CF_IDLIST

TPTOleDataObject class

Delphi Unit
UPTOleDataTransfer

Properties

Methods

AttachClipboard	Attaches the data object that is on the Clipboard.
IsDataAvailable	Checks whether data is available in a specified format.
GetData	Copies data from the attached OLE data object in a specified format.
GetStreamData	Copies data from the attached OLE data object into a CFile pointer in the specified format.
GetGlobalData	Copies data from the attached OLE data object into an HGLOBAL in the specified format.
BeginEnumFormats	Prepares for one or more subsequent GetNextFormat calls.
GetNextFormat	Returns the next data format available.
AttachIDataObject	Attaches the specified OLE data object to the COleDataObject.
ReleaseIDataObject	Detaches and releases the associated IDataObject object.
DetachIDataObject	Detaches the associated IDataObject object.

Events

Description
The COleDataObject class is used in data transfers for retrieving data in various formats from the Clipboard, through drag and drop, or from an embedded OLE item. These kinds of data transfers include a source and a destination. The data source is implemented as an object of the COleDataSource class. Whenever a destination application has data dropped in it or is asked to perform a paste operation from the Clipboard, an object of the COleDataObject class must be created.

This class enables you to determine whether the data exists in a specified format. You can also enumerate the available data formats or check whether a given format is available and then retrieve the data in the preferred format. Object retrieval can be accomplished in several different ways, including the use of a CFile, an HGLOBAL, or an STGMEDIUM structure.

TPTOleDataSource class

Delphi Unit
UPTOleDataTransfer

Properties

Methods

CacheData	Offers data in a specified format using a STGMEDIUM structure.
CacheGlobalData	Offers data in a specified format using an HGLOBAL.
DoDragDrop	Performs drag-and-drop operations with a data source.
SetClipboard	Places a COleDataSource object on the Clipboard.
Clear	Clears the COleDataSource object of data.
FlushClipboard	Renders all data to the Clipboard.
GetClipboardOwner	Verifies that the data placed on the Clipboard is still there.
DelayRenderData	Offers data in a specified format using delayed rendering.
DelayRenderStreamData	Offers data in a specified format in a CFile pointer.
DelaySetData	Called for every format that is supported in OnSetData

Events

OnRenderData	Retrieves data as part of delayed rendering.
OnRenderStreamData	Retrieves data into a CFile as part of delayed rendering.
OnRenderGlobalData	Retrieves data into an HGLOBAL as part of delayed rendering.
OnSetData	Called to replace the data in the COleDataSource object.

Description
The TPTOleDataSource class acts as a cache into which an application places the data that it will offer during data transfer operations, such as Clipboard or drag-and-drop operations.

The object supports delayed rendering. (*not implemented 30-Jul-97*)

Whenever you want to prepare data for a transfer, you should create an object of this class and fill it with your data using the most appropriate method for your data. The way it is inserted into a data source is directly affected by whether the data is supplied immediately (immediate rendering) or on demand (delayed rendering). For every Clipboard format in which you are providing data by passing the Clipboard format to be used (and an optional FORMATETC structure), call DelayRenderData.