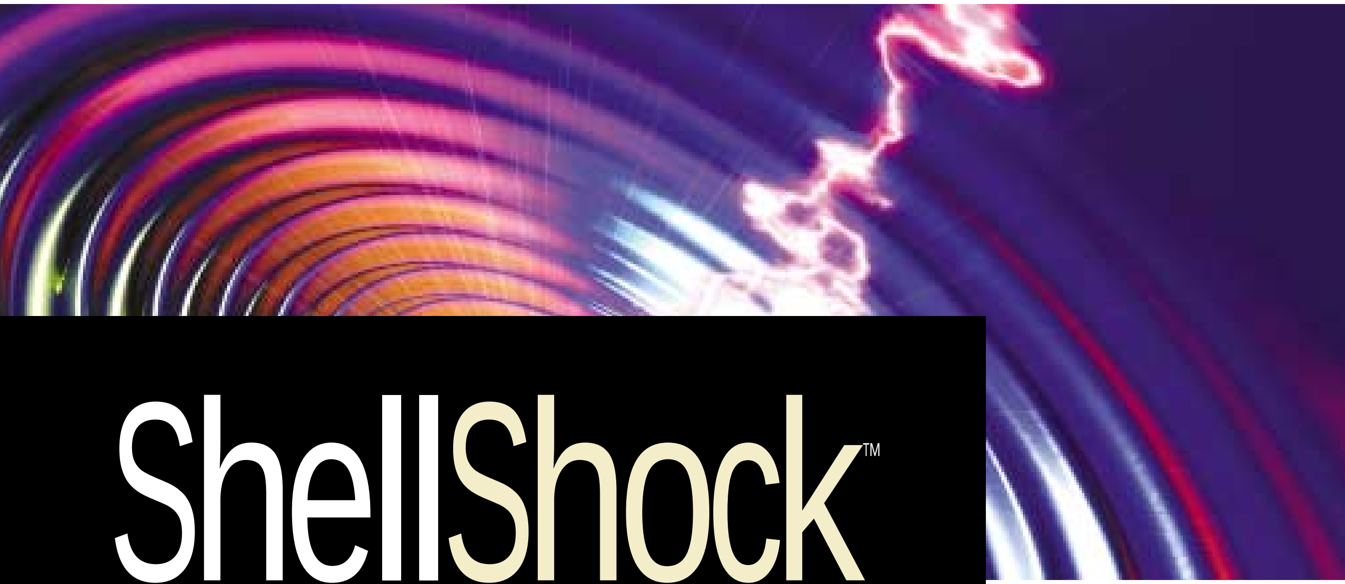
Access the Windows shell with native VCL components



ShellShock™

Integrate your programs with the Windows shell:

- *Display folders in the shell namespace in a tree view*
- *Display the contents of a folder in a list view*
- *Easily build custom file dialogs*
- *Perform file operations complete with animations*
- *Monitor the shell for events and notify your application when they occur*

TURBOPOWER®
Software Company

ShellShock™

TurboPower Software Company
Colorado Springs, CO

www.turbopower.com

Table of Contents

Chapter 1: Introduction

ShellShock contains visual and non-visual components that encapsulate much of the Windows shell. These components make it easy to deal with the complex Windows shell API.

In its most basic form the shell API can be considered the backend for Windows Explorer. When you copy or move files in Explorer you see an animation as the files are being processed. If a file already exists in the target location you are prompted whether to replace the file. In some cases you are prompted when Explorer needs to create a new directory. When deleting files in Explorer you are prompted before the file is deleted or sent to the Recycle Bin. When you format a floppy drive, the Format Drive dialog is displayed and you can proceed with the format from there. Windows Explorer also allows you to create shortcuts to files. File drag and drop operations from Explorer to other applications is possible as well. The Windows shell controls all these elements.

ShellShock's visual components give you tremendous flexibility in presenting the shell namespace (including files and folders) to your users. The shell tree view, list vew, and combo box components can be used together or independantly to allow your users to select folders, view the contents of a folder, create new folders, and much more. The navigator and dialog panel components can be used to easily create custom file dialogs, something that is lacking in the VCL dialog components.

The ShellShock shell components, shown in Table 1.1, allow you to implement shell operations like these in your own applications. Using these components you can develop professional applications that implement the shell user interface with the minimum amount of effort on your part.

Table 1.1: *ShellShock components*

Component	**Description**
TStShellTreeView	Displays folders in the shell namespace in a tree view.
TStShellListView	Displays the contents of a folder in the shell namespace in a list view.
TStShellComboBox	Works with TStShellListView to allow navigation of the shell's file system.
TStShellEnumerator	Allows your application to programmatically enumerate the contents of a folder in the shell namespace.

Table 1.1: *ShellShock components (continued)*

`TStShellNotification`	`Monitors the shell for events and notifies your application when a shell event occurs.`
`TStShellAbout`	`Displays the Windows shell About dialog.`
`TStBrowser`	`Encapsulates the shell Browse for Folder dialog.`
`TStFormatDrive`	`Allows you to format a removable disk.`
`TStFileOperation`	`Performs file operations (copy, delete, move, rename) complete with the Windows shell interface.`
`TStTrayIcon`	`Allows your application to implement a tray icon.`
`TStDropFiles`	`Allows your application to accept files dropped from Windows Explorer or other Windows programs that support drag and drop.`
`TStShortcut`	`Creates Windows shortcuts.`
`TStShellNavigator`	`Emulates the controls found at the top of the Windows common file dialog boxes.`
`TStDialogPanel`	`Emulates all the controls found on the Windows common file dialog boxes.`

Perhaps the most powerful shell components are the TStShellTreeView and TStShellListView visual components. These components encapsulate most of the functionality found in Windows Explorer. Using these components you can practically recreate Explorer if you choose. More likely, though, you will use these components to create a custom directory and file browser, a wizard of some sort, or any other application that needs to provide a view into the Windows shell.

It is important to realize that the Windows shell version can vary widely from one machine to another. For example, the Windows shell version may change if the user installs a new version of Internet Explorer or other Microsoft products. Add the fact that several variations of Windows exist, and you can expect to encounter a wide range of shell version and operating system combinations. These combinations can affect the way the ShellShock shell components work. We have made every effort to work around the known limitations of certain versions of the shell, but you may encounter situations where certain shell components operate differently on two different systems.

System Requirements

For optimal performance you must have the following hardware and software:

1. A computer capable of running Microsoft Windows 95, Windows 98, Windows NT, Windows 2000, or Windows XP. At least 32MB of RAM is recommended.
2. Borland Delphi Version 3 or above or C++Builder 3 or above.
3. An installation of all ShellShock files and example programs for one compiler requires about 20MB of disk space.

Installation

ShellShock can be installed directly from the CD-ROM.

The setup program

Insert the TurboPower Product Suite CD-ROM and follow the instructions presented by the setup program.

SETUP installs ShellShock in C:\Program Files\TurboPower\ShellShock by default. You can specify a different directory if desired. You can choose a full or partial installation. Full installation is recommended, but if you need to conserve disk space, use custom installation to install only selected portions of ShellShock.

Installing for multiple compiler versions

ShellShock supports all Delphi compilers version 3 and above and all version of C++Builder 3 and above. However, the compiled file format is different for each version. The ShellShock setup program allows you to select compiler support for each of the different version of Delphi or C++Builder.

By default, only the installed compilers will be selected when ShellShock is installed for the first time. If ShellShock is being re-installed or upgraded, only support for the previously supported compilers will be selected. If you have installed a new compiler and wish to install ShellShock support for that compiler, you must explicitly select support for that compiler.

Installing into C++Builder

ShellShock can be used with either Delphi or C++Builder. If you install the C++Builder help file and examples then you will find the help file in the ShellShock \Help\Cbuilder directory and the examples in ShellShock\Examples\Cbuilder.

Header and object files for each version of C++Builder are installed into the \HPP* directory located in the ShellShock root directory. For example, headers filed for C++Builder 5 will be located in the \HPP5 directory. If you need to generate header and object files to support other C++Builder compilers, you can either re-install ShellShock and select the new compiler, or use the DCC32.EXE program to compile the SSREG.PAS file using “-jphn” as the command line options.

Organization of this Manual

This manual is organized as follows:

- Chapter 1 is the introduction to the manual.
- Chapter 2 describes the support classes.
- Chapters 3 describes the visual components.
- Chapter 4 describes the non-visual components.
- A separate identifier index and subject index are provided.

Each chapter starts with a general discussion of the classes or procedures and functions discussed in that chapter. Generally this is followed by a discussion of special considerations or other items of interest when using that part of ShellShock.

Overview

A description of the class or component.

Hierarchy

Shows the ancestors of the class being described, generally stopping at a VCL class. The hierarchy also lists the unit in which each class is declared and the number of the first page of the documentation of each ancestor. Some classes in the hierarchy are identified with a number in a bullet: ❶. This indicates that some of the properties, methods, or events listed for the class being described are inherited from this ancestor and documented in the ancestor class.

Properties, methods, and events lists

The properties, methods, and events for the class or component are listed. Some of these may be identified with a number in a bullet: ❶. In these cases, they are documented in the ancestor class from which they are inherited.

Reference section

Details the properties, methods, and events of the class or component. These descriptions are in alphabetical order. They have the following format:

- Declaration of the property, method, or event.
- Default value for properties, if appropriate.
- A short, one-sentence purpose. A ➪symbol is used to mark the purpose to make it easy to skim through these descriptions.
- Description of the property, method, or event. Parameters are also described here.
- Examples are provided in many cases.
- The “See also” section lists other properties, methods, or events that are pertinent to this item.

Throughout the manual, the 💣symbol is used to mark a warning or caution. Please pay special attention to these items.

Naming conventions

To avoid class name conflicts with components and classes included with the compiler or from other third party suppliers, all ShellShock’s class names begin with “St.” Some of the components in ShellShock were originally part of our SysTools product. SysTools uses a component name prefix of “St”. To make conversion of existing applications easier, we left the component name prefix “St” for ShellShock.

“Custom” in a component name means that the component is a basis for descendant components. Components with “Custom” as part of the class name do not publish any properties. Instead, descendants publish the properties that are applicable to the derived component. If you create descendant components, use these custom classes instead of descending from the component class itself.

On-line help

Although this manual provides a complete discussion of ShellShock, keep in mind that there is an alternative source of information available. Once properly installed, help is available from within the IDE. Pressing <F1> with the caret or focus on an Abbrevia property, routine or component displays the help for that item.

Technical Support

The best way to get an answer to your technical support questions is to post it in the Async Professional newsgroup on our news server (news.turbopower.com). Many of our customers find the newsgroups a valuable resource where they can learn from others' experiences and share ideas in addition to getting answers to questions

To get the most from the newsgroups, it is recommended that you use dedicated newsreader software. You'll find a link to download a free newsreader program on our web site at www.turbopower.com/tpslive.

Newsgroups are public, so please do NOT post your product serial number, 16-character product unlocking code or any other private numbers (such as credit card numbers) in your messages.

The TurboPower KnowledgeBase is another excellent support option. It has hundreds of articles about TurboPower products accessible through an easy to use search engine (www.turbopower.com/search). The KnowledgeBase is open 24 hours a day, 7 days a week. So you will have another way to find answers to your questions even when we're not available.

Other support options are described in the product support brochure included with Async Professional. You can also read about support options at www.turbopower.com/support.

Chapter 2: Support Classes

The visual shell components rely upon several support classes. The support classes you are most likely to use in your own applications are TStShellItem and TStShellFolder. These two classes provide information about an item or a folder in the shell namespace. The TStCustomShellController class is a support class that is used internally by ShellShock. It can be used directly, but you will generally use TStEnumerator (a component descended from TStCustomShellController) rather than use TStCustomShellController directly. Other support classes include TStShellItemList and TStShellFolderList.

TStShellItem Class

TStShellItem provides details about an item in the shell namespace. An item may be a folder, a file, a shortcut, or one of several special shell items (such as items in the Control Panel). The properties of TStShellItem are all read-only. This is because the information contained in an item is obtained from the shell and can not be modified by your application. The ShellItems property of the TStShellListView and TStShellEnumerator components is a list of TStShellItems. You can use the ShellItems property to enumerate the items contained in a particular folder.

The bulk of the TStShellItem properties are Boolean properties corresponding to a shell attribute (IsFileFolder, for example). Read these properties to determine whether the attribute is set.

Hierarchy

TObject (VCL)

  TStShellItem (StShlCtl)

Properties

CanCopy	IsCompressed	LargeIcon
CanLink	IsDesktop	ParentFolder
CanPaste	IsDropTarget	Path
CanRename	IsFile	Pidl
ColText	IsFileFolder	OpenIcon
Date	IsFileSystem	OpenIconIndex
DisplayName	IsFileSystemAncestor	OverlayIconIndex
FileAttributes	IsFolder	SimplePidl
FileAttributeStr	IsGhosted	Size
HasPropSheet	IsHidden	SmallIcon
HasRemovableMedia	IsLink	SmallOpenIcon
HasSubFolder	IsReadOnly	TypeName
IconIndex	IsShared	

Methods

Assign	Create	Execute
CopyToClipboard	CreateFromPath	GetFolderSize
CutToClipboard	CreateFromPidl	PasteFromClipboard

Reference Section

Assign — method

```
procedure Assign(AValue : TStShellItem);
```

↳ Copies the contents of a TStShellItem to this object.

Call the Assign method to copy the contents of a TStShellItem object to another TStShellItem object. The Assign method is necessary to create an accurate copy.

CanCopy — read-only, run-time property

```
property CanCopy : Boolean
```

↳ Indicates whether the item can be copied to the clipboard.

CanCopy can be used to determine if the item can be copied or cut to the clipboard. Use this property to enable or disable menu items that pertain to both cut and copy clipboard operations.

See also: CanPaste, CopyToClipboard, CutToClipboard, PasteFromClipboard

CanLink — read-only, run-time property

```
property CanLink : Boolean
```

↳ Indicates whether a shortcut of the item can be created.

CanPaste — read-only, run-time property

```
property CanPaste : Boolean
```

↳ Indicates whether the item can receive shell items pasted from the clipboard.

CanPaste will be True if the clipboard contains a shell item identifier list and if the item accepts pasting. If not, CanPaste will be False. Use this property to enable or disable menu items that pertain to pasting of shell items.

See also: CanCopy, CopyToClipboard, CutToClipboard, PasteFromClipboard

CanRename — read-only, run-time property

```
property CanRename : Boolean
```

↳ Indicates whether the item can be renamed.

ColText — read-only, run-time property

```
property ColText : TStringList
```

↳ The list of strings that describe the details of the item.

ColText is the text that is displayed in columns 1 through n when the item is being displayed in a TStListView in report view mode. ControlPanel items, for example, have a column called “Description.” In this case, ColText will contain a single string containing the description of the Control Panel item. For file system objects the ColText property may contain up to four strings, one each for the Size, Type, Modified, and Attributes columns. Be aware, though, that in some cases ColText may not contain any strings. This will be the case for certain combinations of operating system and shell version that do not directly support retrieving file details from the shell. In those cases, the Size, TypeName, Date, and FileAttributes properties can be used to obtain the desired information. ColText is only valid if the item is contained in a TStShellListView component and if the list view’s ViewStyle property is set to vsReport.

See also: Date, FileAttributes, Size, TypeName

CopyToClipboard — method

```
procedure CopyToClipboard;
```

↳ Copies the shell item to the clipboard.

When the CopyToClipboard method is called, the shell item will be copied to the Windows clipboard as an item identifier list (a pidl). It can then be pasted into any application that allows pasting of shell item identifiers. For example, a file item can be copied to the clipboard and then pasted into a folder in Windows Explorer. TStShellTreeView and TStShellListView also have a CopyToClipboard method. When using these components, call their CopyToClipboard method rather than calling CopyToClipboard for an individual item. This is especially important in the case of TStShellListView where multiple items may be selected.

See also: CutToClipboard, PasteFromClipboard

Create constructor

```
constructor Create;
```

↳ Creates a new TStShellItem object.

The Create constructor creates a new TStShellItem object and initializes internal data fields.

See also: CreateFromPath, CreateFromPidl

CreateFromPath constructor

```
constructor CreateFromPath(Path : string);
```

↳ Creates a new TStShellItem object from a file system path.

The Create constructor creates a new TStShellItem object from a file system path or, if applicable, a path or file name. Path is the full path of the file system directory that represents the item to be created. If the item is a file, Path must contain the full path and file name of the file.

See also: Create, CreateFromPidl

CreateFromPidl method

```
constructor CreateFromPidl(Pidl : PItemIDList);
```

↳ Creates a new TStShellItem object with the given pidl.

The Create constructor creates a new TStShellItem object from an item identifier list. The PidlIn parameter is the item identifier list that represents the item to be created. PidlIn should be a fully qualified item identifier list. See the Pidl property for a description of item identifiers.

See also: Create, CreateFromPath, Pidl

CutToClipboard method

```
procedure CutToClipboard;
```

↳ Copies the shell item to the clipboard and removes the item after a clipboard paste operation.

When the CutToClipboard method is called, the shell item will be copied to the Windows clipboard as an item identifier list (a pidl). The item will not be removed from its current location until after a paste operation has been performed. Once in the clipboard, the item can be pasted into any application that allows pasting of shell item identifiers. For example, a file item can be cut to the clipboard and then pastes into a folder in Windows Explorer. TStShellTreeView and TStShellListView also have a CutToClipboard method. When using these components, call their CutToClipboard method rather than calling CutToClipboard for an individual item. This is especially important in the case of TStShellListView where multiple items may be selected.

See also: CopyToClipboard, PasteFromClipboard

Date — read-only, run-time property

```
property Date : TDateTime
```

The file date and time for items that are part of the file system.

When the item represents a file system object, Date contains the date and time of the last modification of the file. Date is ignored in cases where the item is not part of the file system.

See also: DisplayName, FileAttributes, Path, Size, TypeName

DisplayName — read-only, run-time property

```
property DisplayName : string
```

The textual description of the item as shown in Windows Explorer.

TStShellItem has two string properties that describe the shell item. The DisplayName property contains the display name as shown in Windows Explorer. The Path property contains the full path and file name of the item. For example, if the item points to the root of the C drive, the DisplayName property will contain “(C:)” and the Path property will contain “C:\”. If the item points to the file C:\Data\Data1.txt, the DisplayName will be “Data1.txt” and the Path will be “C:\Data\Data1.txt”.

Not all shell items are file system objects, of course. For items that are not part of the file system, the Path and DisplayName properties may contain identical strings. In some cases the Path property for items not part of the file system may be a globally unique identifier (GUID) that represents the folder to which the item points.

See also: Path

```
property FileAttributes : DWORD
```

Contains the file attributes for items that are part of the file system.

FileAttributes is a DWORD that contains one or more file system attribute flags (read-only, hidden, system file, etc.) as shown in the following table:

Value	**Meaning**
FILE_ATTRIBUTE_ARCHIVE	The file is marked for archiving. Applications use this value to mark files for backup or removal.
FILE_ATTRIBUTE_COMPRESSED	The file or directory is compressed. For a file, this means that all the data in the file is compressed. For a directory, this means that compression is the default for newly created files and subdirectories.
FILE_ATTRIBUTE_DIRECTORY	The file is a directory.
FILE_ATTRIBUTE_HIDDEN	The file is hidden. It is not included in an ordinary directory listing.
FILE_ATTRIBUTE_NORMAL	The file has no other attributes set. This value is valid only if used alone.
FILE_ATTRIBUTE_OFFLINE	The data of the file is not immediately available. Indicates that the file data has been physically moved to off-line storage.
FILE_ATTRIBUTE_READONLY	The file is read-only. Applications can read the file but cannot write to it or delete it.

Value	Meaning
FILE_ATTRIBUTE_SYSTEM	The file is part of the operating system or is used exclusively by it.
FILE_ATTRIBUTE_TEMPORARY	The file is being used for temporary storage. Applications should write to the file only if absolutely necessary. Most of the file's data remains in memory without being flushed to the media because the file will soon be deleted.

FileAttributes is ignored in cases where the item is not part of the file system.

See also: DisplayName, Date, FileAttributesStr, Path, Size, TypeName

FileAttributesStr — read-only, run-time property

```
property FileAttributesStr : string
```

The item's file attributes, expressed as a string.

FileAttributesStr returns the attributes of a file system item as it appears in Windows Explorer's Attributes column. For example, FileAttributesStr will return "HA" for a file that has the read-only and archive attributes. Use FileAttributesStr if you need to display file system attributes as a string.

See also: FileAttributes

Execute — method

```
function Execute : Boolean;
```

Executes (runs) the shell item.

What transpires when Execute is called depends on the item represented by the TStShellItem. In the case of executable files, the file is simply run. In the case of files types with an associated application, the associated application is run and the file loaded. For file types without an association, the shell's Open With dialog is displayed. For folders, the folder is opened in a new Explorer window. The behavior is essentially the same as you see when you double click a file or folder in Windows Explorer.

The following example shows how to execute the selected folder in a TStShellListView:

```
  StShellListView1.SelectedItem.Execute;
```

GetFolderSize method

```
function GetFolderSize(
  Recursive : Boolean; IncludeHidden : Boolean) : Cardinal;
```

↳ Returns the size of the folder and its contents.

Use GetFolderSize to determine the size of a folder, its contents, and optionally its subfolders. Recursive determines whether the folder's subfolders will be used in the calculation. IncludeHidden determines whether the size of hidden files and folders will be included. The return value is the size of the folder and its subfolders (if Recursive is set to True). GetFolderSize always returns 0 for non-folder items.

The following example shows how to get the size of the selected folder in a TStShellListView:

```
  Size :=
    StShellListView1.SelectedItem.GetFolderSize(True, False);
```

See also: Size

HasPropSheet read-only, run-time property

```
property HasPropSheet : Boolean
```

↳ Indicates whether the item has an associated property sheet.

A property sheet is the dialog you see when you right-click an item in Windows Explorer and choose Properties from the context menu. Property sheets are modeless—your application will continue to operate normally while the property sheet is being displayed.

See also: TStListView.ShowPropertySheet, TStTreeView.ShowPropertySheet

HasRemovableMedia read-only, run-time property

```
property HasRemovableMedia : Boolean
```

↳ Indicates whether the item represents a drive that has removable media.

HasSubFolder read-only, run-time property

```
property HasSubFolder : Boolean
```

↳ Indicates whether the item contains subfolders.

IconIndex — read-only, run-time property

```
property IconIndex : Integer
```

The index number in the system image list of the icon that represents the item.

The IconIndex property is used internally by the TStShellTreeView and TStShellListView components. Typically, IconIndex won't be used directly. Instead, use the SmallIcon or LargeIcon properties to get the large or small icon for a particular item.

See also: LargeIcon, OpenIconIndex, OverlayIconIndex, SmallIcon

IsCompressed — read-only, run-time property

```
property IsCompressed : Boolean
```

Indicates whether the item is compressed.

Most Windows operating systems support drive compression. On some operating systems (Windows NT, for example) individual files can be compressed as well as entire drives. IsCompressed only returns a valid value if the file or folder is compressed using the operating system's built-in compression mechanism. IsCompressed will return False for zip archives, for example, if those archives are not on a compressed drive.

IsDesktop — read-only, run-time property

```
property IsDesktop : Boolean
```

Indicates whether the item represents the Desktop folder.

IsDropTarget — read-only, run-time property

```
property IsDropTarget : Boolean
```

Indicates whether dragged objects can be dropped on the item.

IsFile — read-only, run-time property

```
property IsFile : Boolean
```

Indicates whether the item is a file in the file system.

See also: IsFileFolder, IsFileSystem, IsFolder

IsFileFolder — read-only, run-time property

```
property IsFileFolder : Boolean
```

Indicates whether the item is a folder in the file system.

See also: IsFile, IsFileSystem, IsFolder

IsFileSystem — read-only, run-time property

```
property IsFileSystem : Boolean
```

Indicates whether the item is a member of the file system.

See also: IsFile, IsFileFolder, IsFolder

IsFileSystemAncestor — read-only, run-time property

```
property IsFileSystemAncestor : Boolean
```

Indicates whether the item contains one or more file system folders.

IsFolder — read-only, run-time property

```
property IsFolder : Boolean
```

Indicates whether the item is a folder.

Folders are typically thought of as file system directories. Not all folders, however, are file system directories. The Control Panel and Printers objects are folders but are not part of the file system.

See also: IsFile, IsFileFolder, IsFileSystem

IsGhosted — read-only, run-time property

```
property IsGhosted : Boolean
```

Indicates whether the item should be displayed as ghosted.

An item may be ghosted if it is a hidden or file system object, or if the item has been marked for a clipboard cut operation.

See also: IsHidden

IsHidden — read-only, run-time property

```
property IsHidden : Boolean
```

Indicates whether the item is a hidden file or folder.

See also: IsGhosted

IsLink — read-only, run-time property

```
property IsLink : Boolean
```

Indicates whether the item is a shortcut.

A shortcut is a data object that contains information used to access another object in the system such as a file, folder, etc.

IsReadOnly — read-only, run-time property

```
property IsReadOnly : Boolean
```

Indicates whether the item is read-only.

IsShared — read-only, run-time property

```
property IsShared : Boolean
```

Indicates whether the item is a shared for network use.

LargeIcon — read-only, run-time property

```
property LargeIcon : TIcon
```

The large icon associated with the item.

Read the LargeIcon property if you need to display the large icon associated with a shell item.

See also: IconIndex, OpenIcon, SmallIcon

OpenIcon — read-only, run-time property

```
property OpenIcon : TIcon
```

The icon associated with the item in its open state.

Read the OpenIcon property if you need to display the icon associated with a shell item when the item is open. For example, the regular icon for a folder in the file system is a closed folder. The open icon for a folder that is currently selected is an open folder.

See also: LargeIcon, OpenIconIndex, SmallIcon

OpenIconIndex — read-only, run-time property

```
property OpenIconIndex : Integer
```

The index number in the system image list of the icon that is used to show the item in its open state.

See also: IconIndex, OpenIcon

OverlayIconIndex — read-only, run-time property

```
property OverlayIconIndex : Integer
```

The index number in the system image list of the icon that is used as an overlay image over the normal icon image.

Overlay icons are used by the shell for items that are shared, for shortcuts, and in other less obvious cases. OverlayIconIndex is used internally by the TStShellTreeView and TStShellListView components. Typically, OverlayIconIndex won't be used directly.

See also: IconIndex, LargeIcon, OpenIconIndex, SmallIcon

ParentFolder — read-only, run-time property

```
property ParentFolder : IShellFolder
```

A pointer to the IShellFolder object representing the item's parent.

ParentFolder is used internally by ShellShock. Normally, ParentFolder won't be accessed directly. ParentFolder may be used if you need to call Windows shell functions that require an IShellFolder.

PasteFromClipboard method

```
procedure PasteFromClipboard;
```

↳ Pastes the shell item from the clipboard into the folder represented by this shell item.

When you call PasteFromClipboard, the shell item identifier in the clipboard will be pasted to the folder represented by this item. The item in the clipboard may have been placed there by your program or by an external program such as Windows Explorer. TStShellTreeView and TStShellListView also have a PasteFromClipboard method. When using these components, you will normally call their PasteFromClipboard method rather than calling PasteFromClipboard for an individual item.

See also: CanPaste, CopyToClipboard, CutToClipboard

Path read-only, run-time property

```
property Path : string
```

↳ The path and file name of an item when the item is part of the file system.

TStShellItem has two string properties that describe the shell item. The DisplayName property contains the display name as shown in Windows Explorer. The Path property contains the full path and file name of the item. For example, if the item points to the root of the C drive, the DisplayName property will contain “(C:)” and the Path property will contain “C:\”. If the item points to the file C:\Data\Data1.txt, the DisplayName will be “Data1.txt” and the Path will be “C:\Data\Data1.txt”.

Not all shell items are file system objects, of course. For items that are not part of the file system, the Path and DisplayName properties may contain identical strings. In some cases the Path property for items not part of the file system may be a globally unique identifier (GUID) that represents the folder the item points to.

See also: DisplayName

Pidl run-time read-only property

```
property Pidl : PItemIDList
```

↳ The fully qualified item identifier list for the item.

Item identifiers are the heart of the Windows shell. An item identifier represents an item in the shell namespace, whether that item is a folder itself or an item in a folder. A pidl is a pointer to a list of item identifiers. A pidl may be a simple pidl containing a single item identifier, or a complex pidl containing a list of item identifiers representing the shell hierarchy. Pidls are constructed relative to the item's parent folder. A fully qualified pidl is an item identifier list that is relative to the Desktop folder.

Pidl is used internally by ShellShock. You probably won't need to use Pidl directly.

See also: SimplePidl

SimplePidl read-only, run-time property

```
property SimplePidl : PItemIDList
```

↳ A single item identifier for the item.

SimplePidl is an item identifier list containing a single item identifier. This item identifier is relative to the item's immediate parent folder. SimplePidl is used internally by ShellShock. You probably won't need to use SimplePidl directly.

See also: Pidl

Size read-only, run-time property

```
property Size : Integer
```

↳ Contains the size of items that are part of the file system.

Size contains the size of the item when the item is part of the file system. Size will be 0 for items that are not part of the file system.

See also: DisplayName, Date, FileAttributes, Path, TypeName

SmallIcon read-only, run-time property

```
property SmallIcon : TIcon
```

↳ The small icon associated with the item.

Read the SmallIcon property if you need to display the small icon associated with a shell item.

See also: IconIndex, LargeIcon, OpenIcon

SmallOpenIcon — read-only, run-time property

```
property SmallOpenIcon : TIcon
```

The small icon associated with the item in its open state.

See also: LargeIcon, OpenIcon, SmallIcon

TypeName — read-only, run-time property

```
property TypeName : string
```

The type description of items that are part of the file system.

TypeName is a string that describes a file system item. This is the text that you see in the Type column of Windows Explorer. For example, applications have a type name of “Application” and DLLs have a type name of “Application Extension.”

See also: Date, DisplayName, FileAttributes, Path, Size

TStShellFolder Class

TStShellFolder is a class derived from TStShellItem. It adds properties specific to shell folders to those found in TStShellItem.

Hierarchy

TObject (VCL)

❶ TStShellItem (StShlCtl) 10

    TStShellFolder (StShlCtl)

Properties

- ❶ CanCopy
- ❶ CanLink
- ❶ CanPaste
- ❶ CanRename
- ❶ ColText
- ❶ Date
- ❶ DisplayName
- ❶ FileAttributes
- FolderCount
- ❶ HasPropSheet
- ❶ HasRemovableMedia
- ❶ HasSubFolder
- HiddenCount
- ❶ IconIndex
- ❶ IsCompressed
- ❶ IsDesktop
- ❶ IsDropTarget
- ❶ IsFile
- ❶ IsFileFolder
- ❶ IsFileSystem
- ❶ IsFileSystemAncestor
- ❶ IsFolder
- ❶ IsGhosted
- ❶ IsLink
- ❶ IsReadOnly
- ❶ IsShared
- ItemCount
- ❶ LargeIcon
- ❶ ParentFolder
- ❶ Path
- ❶ Pidl
- ❶ OpenIcon
- ❶ OpenIconIndex
- ❶ OverlayIconIndex
- ❶ SimplePidl
- ❶ Size
- ❶ SmallIcon
- ❶ TypeName

Methods

- ❶ Assign
- ❶ CopyToClipboard
- ❶ CutToClipboard
- ❶ Create
- ❶ CreateFromPath
- ❶ CreateFromPidl
- ❶ PasteFromClipboard

Reference Section

FolderCount — read-only, run-time property

```
property FolderCount : Integer
```

↳ The number of items in the folder that are themselves folders.

HiddenCount — read-only, run-time property

```
property HiddenCount : Integer
```

↳ The number of hidden items in the folder.

ItemCount — read-only, run-time property

```
property ItemCount : Integer
```

↳ The number of non-folder items in the folder.

TStShellItemList Class

TStShellItemList maintains a list of TStShellItem objects. The ShellItems property of the TStShellController, TStShellEnumerator, TStShellListView components are instances of TStShellItemList.

Hierarchy

TObject (VCL)

TStShellItemList (StShlCtl)

Properties

Count	Items

Reference Section

Count read-only, run-time property

```
property Count : Integer
```

The number of items in the shell item list.

Items read-only, run-time property

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

Used to access the individual TStShellItem objects in the list.

The ShellItems property of the TStShellEnumerator and TStShellListView components are instances of TStShellItemList. The following code shows how to iterate through all the items in a TStShellListView and add their display name to a memo:

```
var
  I : Integer;
...
  for I := 0 to StShellListView1.ShellItems.Count - 1 do
    Memo1.Lines.Add(
      StShellListView1.ShellItems[I].DisplayName);
```

Note that Items is the default index property for TStShellItems. As such, it is not necessary to explicitly reference the Items property in Delphi. C++Builder users must reference the Items property directly. For example:

```
Memo1->Lines->Add(
  StShellListView1->ShellItems->Items[I]->DisplayName);114
TStShellFolderList Class
```

TStShellFolderList Class

TStShellFolderList maintains a list of TStShellFolder objects. The Folders property of the TStShellTreeView component is an instance of TStShellFolderList.

Hierarchy

TObject (VCL)

TStShellFolderList (StShlCtl)

Properties

Count	Items

Reference Section

Count read-only, run-time property

```
property Count : Integer
```

The number of items in the list.

Items read-only, run-time property

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

Used to access the individual TStShellFolder objects in the list.

The Folders property of the TStShellTreeView component is an instance of TStShellFolderList. It is possible to enumerate the folders in the tree view by iterating through the list of folders. However, it is not generally beneficial to do so because the list of items does not take the tree view’s hierarchy into account.

Chapter 3: Visual Components

ShellShock contains a rich set of visual components to aid in application development. The visual components are TStShellTreeView, TStShellListView, TStShellComboBox, TStShellNavigator, and TStDialogPanel.

TStShellTreeView, TStShellListView, and TStShellComboBox are used to provide a visual interface to the Windows shell. These components can be used separately or in combination. When used together, these components interact with each other. For example, hooking a TStShellListView to a TStShellTreeView gives you much of the functionality you see in Windows Explorer. TStShellTreeView displays the shell hierarchy in tree format. TStShellListView displays the contents of a folder. Selecting a folder in the tree view causes the list view to update its contents accordingly. If a folder in the list view is double-clicked, that folder's contents are loaded into the list view and the tree view expands to show that folder. Similarly, when a TStShellComboBox is hooked to the list view, changes to the combo box are reflected in the list view (and, subsequently, in the tree view as well). These three components could be used to build your own version of Windows Explorer without writing a single line of code.

TStShellTreeView and TStShellListView support a number of features found in Windows Explorer. Some of those features include drag and drop (between the tree view and list view as well as to and from other applications), support for the shell context menu, clipboard operations, and displaying the properties for a file or folder.

TStShellNavigator and TStDialogPanel are designed to aid in creating custom file dialog boxes. TStShellNavigator emulates the top portion of the Windows file dialog boxes. It is designed to use in conjunction with a TStShellListView. It contains a TStShellComboBox and buttons for moving back to the last folder visited, creating a new folder, moving up a level in the shell tree, and changing the style of the associated TStShellListView. TStDialogPanel is a visual component that contains all of the controls found on the common file dialog boxes. It provides an easy way to create custom file dialog boxes. Simply place a TStDialogPanel on a form and place other needed components around it. At run-time, the components on the TStDialogPanel work together to emulate the behavior of the Windows file dialog boxes.

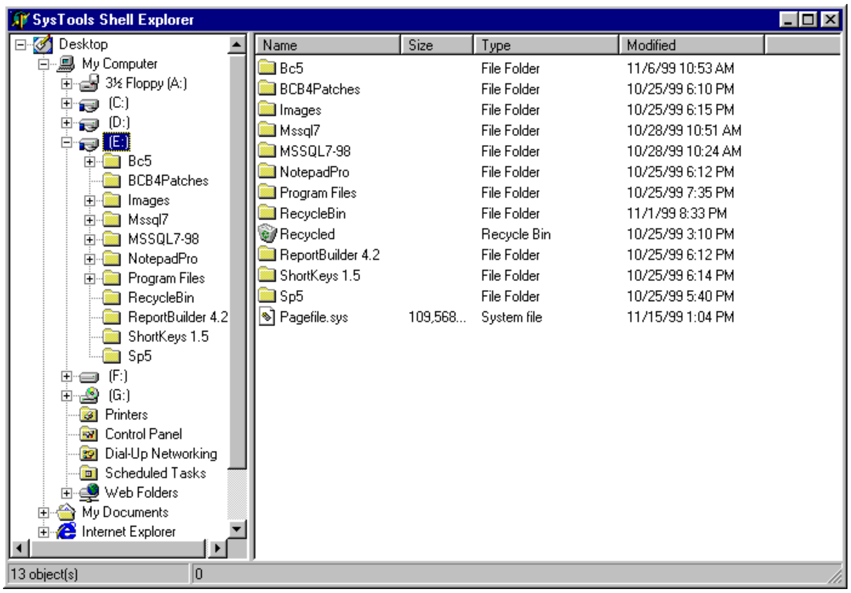
Overall, ShellShock's visual components provide the tools you need to display the contents of the Windows shell, to modify those contents, to create custom file dialog boxes, and much more.

3

TStShellTreeView Component

The TStShellTreeView component is a tree view component that emulates the left hand pane of Windows Explorer. It displays a list of folders in the shell along with each folder's associated icon. This includes system folders (such as the Control Panel, Printers, Network Neighborhood, and the Recycle Bin) as well as file system folders. Expanding a folder will display any folders contained within that folder. When you right-click on a folder, the appropriate shell context menu is displayed for that folder. When you select a folder, the OnItemSelected event is generated. The OnItemSelected event handler gives you a pointer to a TStShellFolder object. This object can be used to determine information about the folder selected.

TStShellTreeView can be used alone or can be used in conjunction with the TStShellListView component. When the TStShellTreeView is hooked to a TStShellListView, the list view is automatically filled with the contents of the folder selected in the tree view, as shown in Figure 3.1.



Name	Size	Type	Modified
Bc5		File Folder	11/6/99 10:53 AM
BCB4Patches		File Folder	10/25/99 6:10 PM
Images		File Folder	10/25/99 6:15 PM
Mssql7		File Folder	10/28/99 10:51 AM
MSSQL7-98		File Folder	10/28/99 10:24 AM
NotepadPro		File Folder	10/25/99 6:12 PM
Program Files		File Folder	10/25/99 7:35 PM
RecycleBin		File Folder	11/1/99 8:33 PM
Recycled		Recycle Bin	10/25/99 3:10 PM
ReportBuilder 4.2		File Folder	10/25/99 6:12 PM
ShortKeys 1.5		File Folder	10/25/99 6:14 PM
Sp5		File Folder	10/25/99 5:40 PM
Pagefile.sys	109,568...	System file	11/15/99 1:04 PM

Figure 3.1: ShellShock Shell Explorer example program.

A note on using alternate colors for tree view items

The CompressedColor property and the toColorCompressed value of the Options property work together to give you the option of displaying compressed files and folders in an alternate color. This feature is not available on versions of the VCL prior to 4 (in other words, Delphi 3 and C++Builder 3) due to the lack of support for custom drawing of tree view items.

Hierarchy

TCustomTreeViewComponent (VCL)

  TStCustomShellTreeView (StShlCtl)

    TStShellTreeView (StShlCtl)

Properties

CompressedColor	MaxNotifications	SpecialRootFolder
ExpandInterval	Options	SpecialStartInFolder
Filtered	RootFolder	StartInFolder
Folders	SelectedFolder	
ListView	ShellVersion	

Methods

AddFolder	PasteToClipboard	SelectFolder
CopyToClipboard	Refresh	SelectSpecialFolder
CutToClipboard	RenameFolder	ShowPropertySheet
DeleteFolder	Refresh	

Events

OnFilterItem	OnFolderSelected	OnShellChangeNotify

Reference Section

AddFolder method

```
function AddFolder(FolderName : string) : Boolean;
```

Creates a new file folder.

Call AddFolder to create a new folder in the file system. The new folder will be added to the currently selected node in the tree view. FolderName is the name of the folder to be added. If FolderName is an empty string the folder will be added with the name “New Folder.” AddFolder returns True if the folder was created or False if the folder could not be created.

See also: DeleteFolder, RenameFolder

CompressedColor property

```
property CompressedColor : TColor
```

Default: clBlue

The text color of compressed folders when the toColorCompressed option is on.

The text of compressed folders will be drawn with CompressedColor. Compressed folders are only drawn in this color if the Options property contains the toColorCompressed value. CompressedColor is not available in versions of Delphi and C++Builder prior to version 4.

See also: Options

CopyToClipboard method

```
procedure CopyToClipboard;
```

Copies the currently selected shell item to the clipboard.

When you call CopyToClipboard, the currently selected tree view node will be copied to the Windows clipboard as an item identifier list (a pidl). It can then be pasted into any application that allows pasting of shell item identifiers. For example, you can copy a file item to the clipboard and then paste the item into a folder in Windows Explorer. The CanCopy property of TStShellItem can be used to determine if the selected item can be copied to the clipboard.

See also: CutToClipboard, PasteFromClipboard, TStShellItem.CanCopy

CutToClipboard method

```
procedure CutToClipboard;
```

↳ Copies the shell item to the clipboard and removes the item after a clipboard paste operation.

When you call CutToClipboard, the currently selected tree view node will be copied to the Windows clipboard as an item identifier list (a pidl). The item will not be removed from its current location until after a paste operation has been performed. Once in the clipboard, the item (usually a folder) can be pasted into any application that allows pasting of shell item identifiers. For example, you can cut a folder to the clipboard and then paste it into a folder in Windows Explorer. The CanCopy property of the TStShellItem class can be used to determine if the selected item can be copied to the clipboard.

See also: CopyToClipboard, PasteFromClipboard, TStShellItem.CanCopy

DeleteFolder method

```
function DeleteFolder(
  Recycle : Boolean; Confirm : Boolean) : Boolean;
```

↳ Deletes the currently selected folder in the tree view.

Call DeleteFolder to delete the currently selected folder in the tree view. Recycle determines whether the deleted folder is sent to the Recycle Bin. Confirm determines whether a confirmation dialog will be displayed before deleting the folder. DeleteFolder returns True if the folder was deleted or False if the folder could not be deleted.

See also: AddFolder, RenameFolder

ExpandInterval property

```
property ExpandInterval : Integer
```

Default: 2000

↳ The amount of time, in milliseconds, the cursor hovers over a node before that node is automatically expanded during drag and drop operations.

When dragging a file or folder onto a TStShellTreeView, a node will automatically expand if the cursor pauses over the node for a period of time. The amount of time that passes before the node is expanded is determined by the ExpandInterval property. The default value for ExpandInterval is 2000 milliseconds (2 seconds). Set ExpandInterval to 0 if you don’t want nodes to expand automatically.

Filtered property

```
property Filtered : Boolean
```

Determines whether a filter should be applied to the tree view.

When Filtered is True, the OnFilterItem event will be called as each folder in the tree view is enumerated. You can provide code in an OnFilterItem event handler to limit the items that are displayed in the tree view.

See also: OnFilterItem

Folders property

```
property Folders : TStShellFolderList
```

The list of TStShellFolder objects that are represented by the tree view nodes.

Folders contains a list of TStShellFolder objects, one for each node in the tree view. It is not typically beneficial to iterate the objects in the Folders property because Folders is a flat list whereas a tree view is hierarchical. However, it is sometimes beneficial to be able to obtain the TStShellFolder object for a particular tree node. Each node's Data property contains the index into the Folders list for the TShellFolder object that represents the node. The following code shows how to obtain a TStShellFolder object for a particular tree node:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SI : TStShellItem;
  Node : TTreeNode;
begin
  Node := StShellTreeView1.TopItem.GetFirstChild;
  SI := StShellTreeView1.Folders[Integer(Node.Data)];
  Label1.Caption := SI.Path;
end;
```

See also: SelectedFolder

ListView property

```
property ListView : TStCustomShellListView
```

The TStShellListView that is associated with the tree view.

Set ListView to the TStShellListView that should be associated with the tree view. If ListView is assigned, the list view will automatically be updated when the folder selection in the tree view changes.

MaxNotifications property

```
property MaxNotifications : Integer
```

Default: 5

↳ The maximum number of shell event notifications that will be processed at one time.

TStShellTreeView uses an instance of the TStShellNotification component internally to watch for shell events. When a folder is created, for example, the shell sends notification of this fact. The TStShellNotification component detects this event and notifies TStShellTreeView so it can update its display. For events such as creating files or folders only one or two such notification events are sent. For some operations—deleting a large number of folders, for example—hundreds of notifications might be sent. In this case it is not necessary for TStShellTreeView to receive notification of every file or folder deletion. It is only necessary that it receive one notification so that it can update its display. You can adjust MaxNotifications from its default of 5 if you find that the tree view is not updating properly for certain shell operations. Set MaxNotifications to 0 for unlimited notifications.

OnFilterItem event

```
property OnFilterItem : TStEnumItemEvent
```

```
TStEnumItemEvent = procedure(
  Sender : TObject; ShellItem : TStShellItem;
  var Accept : Boolean) of object;
```

↳ Defines an event handler that is called for each item in the folder that is being enumerated.

The event handler assigned to OnFilterItem will be called for each item as a particular node in the tree view is enumerated, and when the Filtered property is True. Sender is the component that generated the event. ShellItem is a pointer to the TStShellItem object representing the shell item. Accept determines whether the item is added to the component’s Folders list (and whether it appears in the tree view). You can use OnFilterItem to filter the list of items based on any characteristics you choose. The following example shows how to filter the list so that it contains only folders that begin with the letter ‘t’:

```
  procedure TForm1.StShellTreeView1FilterItem(
    Sender: TObject; ShellItem: TStShellItem;
    var Accept: Boolean);
  begin
    Accept := (UpperCase(ShellItem.DisplayName[1]) = 'T')
  end;
```

Keep in mind that the filter is applied every time a node in the tree view is enumerated, not just when the top node is enumerated.

See also: Filtered, TStShellItem

OnFolderSelected event

```
property OnFolderSelected : TStFolderSelectedEvent
```

```
TStFolderSelectedEvent = procedure(
  Sender : TObject; Folder : TStShellFolder) of object;
```

↳ Defines an event handler that is called when the selected folder changes.

The method assigned to the OnFolderSelected event is called each time the tree view's selection changes. Sender is the component that generated the event. Folder is a pointer to the TStShellFolder object representing the selected folder.

Respond to the OnFolderSelected event when you want notification that the selected folder has changed or if you want to access the object represented by the Folder parameter.

The following example updates two labels on a form with the folder's display name and path when the tree view's selection changes:

```
procedure TForm1.StShellTreeView1FolderSelected(
  Sender: TObject; Folder: TStShellFolder);
begin
  Label1.Caption := Folder.DisplayName;
  Label2.Caption := Folder.Path
end;
```

You can also use the SelectedFolder property to obtain information about the currently selected folder.

See also: SelectedFolder, TStShellFolder

```
property OnShellChangeNotify : TStShellNotifyEvent3
```

```
TStShellNotifyEvent3 = procedure(Sender : TObject;
  OldShellItem : TStShellItem; NewShellItem : TStShellItem;
  Events : TStNotifyEventsSet) of object;
```

Defines an event handler that is called when an event occurs in the shell.

The method assigned to the OnShellChangeNotify event is called each time a shell event occurs. A shell event may occur based on a folder's contents changing, media being inserted into a removable media drive, a file or folder being added, renamed, or deleted, network drives being mapped, and so on.

Sender is the component that generated the event. OldShellItem is a pointer to the TStShellItem representing the shell item prior to the change (the original name of a folder that is being rename, for example). NewShellItem is a pointer to the TStShellItem object representing the new shell object. In some cases OldShellItem and NewShellItem will contain identical properties. In other cases NewShellItem could be nil so be sure to check for nil before using this parameter. Events is a set indicating which shell events occurred. See TStShellNotification for more information on the types of shell events that can occur.

See also: TStShellItem, TStShellNotification

```
property Options : TStTreeOptionsSet
```

```
TStTreeOptions = (toAllowDrag, toAllowDrop, toAllowRename,
  toColorCompressed, toExpandTopNode, toExtendedMenu, toShellMenu,
  toShowFiles, toShowHidden);
```

```
TStTreeOptionsSet = set of TStTreeOptions;
```

Default: [toExpandTopNode, toAllowRename, toAllowDrag, toAllowDrop, toShellMenu]

↳ A set of options that determine how the tree view operates.

The Options property is used to determine how the TStShellTreeView operates, its display properties, and whether items in the tree view can be renamed. The following table describes each of the possible options:

Option	**Meaning**
toAllowDrag	Determines whether the component is a drag source. When this option is set, users can drag items from the tree view to other folders in the tree view, to a TStShellListView, or to other applications that support dropping of file items.
toAllowDrop	Determines whether shell items can be dropped on folders in the tree view. The dragged items can come from within the tree view, from a TStShellListView, from Windows Explorer, or from any application that allows dragging of file items.
toAllowRename	Determines whether the user can rename items using in-place editing.
toColorCompressed	When this option is set, compressed folders are displayed in the color specified by the CompressedColor property. This option is not available in versions of the VCL prior to 4
toExpandTopNode	When this option is set, the top node of the tree view will automatically expand when the component is initially displayed.
toExtendedMenu	When this option is set, the shell context menu will show additional items if the Shift key is held down when the context menu is invoked. For example, under Windows NT the context menu for a file system object will show additional items called Compress and Uncompress. This option is ignored if Options does not include toShellMenu.

toShellMenu	When this option is set, right-clicking an item in the tree view will invoke the shell’s context menu for that item.
toShowFiles	Determines whether non-folder items are displayed in addition to folders. Normally only folders are shown but you want to show non-folder items (such as files) if you are creating a custom browser.
toShowHidden	Determines whether hidden folders and items are displayed.

See also: CompressedColor

PasteFromClipboard method

```
procedure PasteFromClipboard;
```

↳ Pastes a shell item in the clipboard into the currently selected tree view node.

When you call PasteFromClipboard, the shell item identifier in the clipboard will be pasted to the folder represented by the currently selected tree view node. The item in the clipboard may have been placed there by your program or by an external program such as Windows Explorer. The CanPaste property of TStShellItem can be used to determine whether or not the item represented by the currently selected node allows pasting.

See also: CanPaste, CopyToClipboard, CutToClipboard, TStShellItem.CanPaste

Refresh method

```
procedure Refresh(ANode : TTreeNode);
```

↳ Refreshes the contents of the tree view or of a specific node.

Call Refresh to force the contents of the TStShellTreeView to be updated. ANode is the node to refresh. To refresh the entire tree view, pass nil for ANode. To refresh a particular node, pass the TTreeNode representing that node.

RenameFolder method

```
function RenameFolder(
  NewName : string; Confirm : Boolean) : Boolean;
```

↳ Renames the currently selected folder in the tree view.

Call RenameFolder to rename the currently selected folder in the tree view. NewName is the new folder name that will be given to the folder. Confirm determines whether a confirmation dialog will be displayed before renaming the folder. RenameFolder return True if the folder was renamed, or False if the folder could not be renamed.

See also: AddFolder, DeleteFolder, Options(toAllowRename)

RootFolder property

```
property RootFolder : string
```

Default: Empty string

↳ The folder that will be used as the root of the tree view.

The file folder specified by RootFolder will be the root (top level) of the TStShellTreeView. The root folder can be specified by either the RootFolder property or the SpecialRootFolder property. Use RootFolder to specify a file folder as the tree view's top node. Use SpecialRootFolder to set the top node to system folders such as the Desktop or Network Neighborhood. If the specified folder is invalid, an EStInvalidFolder exception is raised.

See also: SelectFolder, SelectSpecialFolder, SpecialRootFolder, SpecialStartInFolder, StartInFolder

SelectFolder method

```
procedure SelectFolder(Path : string);
```

↳ Selects a particular file system folder in the tree view and makes it active.

Call SelectFolder to programmatically select a particular file system folder in the tree view. Path is the fully qualified path (drive letter and folder name) of the folder to select. The folder need not be visible in order to select it with SelectFolder. If the folder passed in Path does not exists or is not a child of the current root node, an ESsInvalidFolder exception is raised with a value of ssscInvalidFolder.

The following example selects the C:\WINNT folder in the tree view:

```
  StShellTreeView1.SelectFolder('c:\winnt');
```

See also: SelectSpecialFolder

SelectedFolder — read-only, run-time property

```
property SelectedFolder : TStShellFolder
```

↳ A pointer to the TStShellFolder representing the currently selected node in the tree view.

Read SelectedFolder to obtain information about the folder or item in the TStShellTreeView that is currently selected.

See also: OnFolderSelected, TStShellFolder

SelectSpecialFolder — method

```
procedure SelectSpecialFolder(Folder : TStSpecialRootFolder);
```

```
TStSpecialRootFolder = (sfAltStartup, sfAppData, sfBitBucket,
  sfCommonAltStartup, sfCommonDesktopDir, sfCommonFavorites,
  sfCommonPrograms, sfCommonStartMenu, sfCommonStartup,
  sfControls, sfCookies, sfDesktop, sfDesktopDir, sfDrives,
  sfFavorites, sfFonts, sfHistory, sfInternet, sfInternetCache,
  sfNetHood, sfNetwork, sfNone, sfPersonal, sfPrinters,
  sfPrintHood, sfPrograms, sfRecentFiles, sfSendTo, sfStartMenu,
  sfStartup, sfTemplates);
```

↳ Selects a shell folder in the tree view and makes it active.

Call SelectSpecialFolder to programmatically select one of Windows' special folders (Control Panel, Network, Printers, etc.). The folder need not be visible in order to select it with SelectFolder. Folder is the shell folder to select. If the folder passed in Folder does not exists, an ESsInvalidFolder exception is raised with a value of ssscInvalidFolder.

The following example selects the Control Panel folder in the tree view:

```
  StShellTreeView1.SelectSpecialFolder(sfControls);
```

See also: SelectFolder

ShellVersion — read-only, run-time property

```
property ShellVersion : Double
```

↳ The version number of Windows' SHELL32.DLL.

Read ShellVersion to determine the version of the Windows shell running on a particular machine. Some shell operations and special folders are only available on version 4.72 of the shell and later.

ShowPropertySheet method

```
function ShowPropertySheet : Boolean;
```

↳ Displays the property sheet associated with the currently selected node in the tree view.

When you call ShowPropertySheet, the property sheet for the selected node will be displayed. A property sheet is the dialog you see when you right-click an item in Windows Explorer and choose Properties from the context menu. Property sheets are modeless—your application will continue to operate normally while the property sheet is being displayed. ShowPropertySheet will return True if the item has a property sheet that can be displayed, or False if the item does not have a property sheet.

See also: TStShellItem.HasPropSheet

SpecialRootFolder property

```
property SpecialRootFolder : TStSpecialRootFolder
```

```
TStSpecialRootFolder = (sfAltStartup, sfAppData, sfBitBucket,
  sfCommonAltStartup, sfCommonDesktopDir, sfCommonFavorites,
  sfCommonPrograms, sfCommonStartMenu, sfCommonStartup,
  sfControls, sfCookies, sfDesktop, sfDesktopDir, sfDrives,
  sfFavorites, sfFonts, sfHistory, sfInternet, sfInternetCache,
  sfNetHood, sfNetwork, sfNone, sfPersonal, sfPrinters,
  sfPrintHood, sfPrograms, sfRecentFiles, sfSendTo, sfStartMenu,
  sfStartup, sfTemplates);
```

Default: sfDesktop

↳ Used to specify the shell folder that will be used as the root of the tree view.

The system folder specified by SpecialRootFolder will be the root (top level) of the TStShellTreeView. The root folder can be specified by either the RootFolder property or the SpecialRootFolder property. Use RootFolder to specify a file folder as the tree view's top node. Use SpecialRootFolder to set the top node to system folders such as the Desktop or Network Neighborhood. The list of special root folders is a list of system folders defined by Windows. Not every folder in the list is available on all versions of the shell. If a particular folder is invalid, an EStShellError exception is raised.

See also: RootFolder, SelectFolder, SelectSpecialFolder, SpecialStartInFolder, StartInFolder

SpecialStartInFolder property

```
property SpecialStartInFolder : TStSpecialRootFolder
```

```
TStSpecialRootFolder = (sfAltStartup, sfAppData, sfBitBucket,
  sfCommonAltStartup, sfCommonDesktopDir, sfCommonFavorites,
  sfCommonPrograms, sfCommonStartMenu, sfCommonStartup,
  sfControls, sfCookies, sfDesktop, sfDesktopDir, sfDrives,
  sfFavorites, sfFonts, sfHistory, sfInternet, sfInternetCache,
  sfNetHood, sfNetwork, sfNone, sfPersonal, sfPrinters,
  sfPrintHood, sfPrograms, sfRecentFiles, sfSendTo, sfStartMenu,
  sfStartup, sfTemplates);
```

↳ Determines the shell special folder node in the tree view that will be selected when the tree view first loads.

Set SpecialStartInFolder to one of Windows' special folders (Control Panel, Network, Printers, etc.) to force TStTreeView to select that folder when the tree view is first displayed. For example, you may want the Control Panel folder selected when the tree view is displayed. In that case you might set SpecialRootFolder to sfDrives so that "My Computer" is the top node of the tree view and SpecialStartInFolder to sfControls to automatically select the Control Panel.

SpecialStartInFolder is only used for folders other than file system folders. To force a particular file system folder to be the selected folder, use the StartInFolder property. Setting SpecialStartInFolder will clear the StartInFolder property.

See also: RootFolder, SpecialRootFolder, StartInFolder

StartInFolder property

```
property StartInFolder : string
```

↳ Determines the file system folder node in the tree view that will be selected when the tree view first loads.

Set StartInFolder to a file system folder to force TStTreeView to select that folder when the tree view is first displayed. For example, let's say you want the tree view to show the contents of the C: drive and the C:\WINNT folder selected when the tree view is displayed. In that case you would set RootFolder to "C:\" and StartInFolder to "C:\WINNT". The value of StartInFolder must be a fully qualified path (drive letter and folder name). No exception is raised if the folder specified in StartInFolder does not exist.

StartInFolder is only used for file system folders. To select other folder types, use the SpecialStartInFolder property. Setting StartInFolder will set SpecialStartInFolder to sfNone.

See also: RootFolder, SpecialRootFolder, SpecialStartInFolder

TStShellListView Component

The TStShellListView component is a list view component that emulates the list view found in the right-hand pane of Windows Explorer. It displays the folder and non-folder items in a particular shell folder. Shell folders include file folders and special folders such as the Control Panel, Printers, Network Neighborhood, the Recycle Bin, and so on. The list view can be configured to show large icons, small icons, a simple list, or report style. In report style, the list view shows details of the items in the list. For file folders this includes each item's size (non-folder items only), the item's type description, the date it was last modified, and its attributes.

The version of the Windows shell that is present on a particular system is often determined by the version of Internet Explorer installed on that system. It also varies with the particular operating system. This has some impact on how the details for a folder are displayed. When possible, the details for folder items are obtained from the shell. In some combinations of operating system and shell version, however, it is not always possible to get item details directly from the shell. Further, on some operating systems, the details that are displayed are dependent on the user's settings in Windows Explorer. Under Windows 98, for example, the Attributes column of the TStShellListView will not appear if the user has the option to show file attributes turned off in Windows Explorer.

When you select a folder in the list view, the OnItemSelected event is generated. This event handler gives you a pointer to a TStShellItem object. You can use this object to determine information about the selected item (see TStShellItem for details on the information that can be obtained).

TStShellListView can be used alone or in conjunction with a TStShellTreeView component, a TStShellComboBox component, or both. When the TStShellListView is hooked to a TStShellTreeView, the list view automatically filled with the contents of the folder selected in the tree view. In addition, double-clicking a folder item in the list view causes that item to be displayed in the associated tree view. Similarly, the TStShellComboBox component updates its display to reflect the folder being displayed in the list view.

If you do not assign a TPopupMenu to the PopupMenu property, a default popup menu is automatically created for you. The default popup is provided for testing purposes. It allows you to set the view style of the list view to one of the four view styles.

Some of the functionality found in TStShellListView is only supported on versions 4 and later of C++Builder and Delphi. One property not supported on prior versions of the VCL is Optimization. This property allows you to determine whether the list view will be optimized for enumeration speed or for display speed. This property is ignored on VCL 3.x and earlier because those versions of the VCL do not have support for virtual list views. The virtual list view is the key to the way the Optimization property works internally. As a result, applications using TStListView will perform slower when compiled with older versions of the compiler. This is not generally a problem with folders less than, say, 2,000 items. With folders greater than 2,000 items, the speed difference will be noticeable when compared to Windows Explorer.

Other features not supported on VCL version 3.x and earlier are the CompressedColor property and the loColorCompressed value of the Options property. These properties work together to give you the option of displaying compressed files and folders in an alternate color. This feature is not available on older versions of the VCL due to the lack of support for custom drawing of list view items.

Hierarchy

TCustomListViewComponent (VCL)

  TStCustomShellListView (StShlCtl)

    TStShellListView (StShlCtl)

Properties

- ComboBox
- CompressedColor
- FileFilter
- Filtered
- Folder
- MaxNotifications
- OpenDialogMode
- Optimization
- Options
- RootFolder
- SelectedItem
- SelectedItems
- ShellItems
- ShellVersion
- SpecialRootFolder
- TreeView

Methods

- AddFolder
- Clear
- CopyToClipboard
- CutToClipboard
- DeleteItem
- LoadFolder
- MoveUpOneLevel
- RenameItem
- Refresh
- PasteFromClipboard
- SetFolder
- ShowPropertySheet
- SortColumn

Events

- OnFilterItem
- OnItemDblClick
- OnItemSelected
- OnListFilled
- OnShellChangeNotify

Reference Section

AddFolder method

```
function AddFolder(FolderName : string) : Boolean;
```

Creates a new file folder.

Call AddFolder to create a new folder in the file system. The new folder will be added to the folder currently displayed in the list view. FolderName is the name of the folder to be added. If FolderName is an empty string the folder will be added with the name “New Folder.” AddFolder returns True if the folder was created, or False if the folder could not be created.

See also: DeleteItem, RenameFolder

Clear method

```
procedure Clear;
```

Clears the list view of all contents.

Call Clear to clear the list view of its contents. Calling Clear does not modify the RootFolder or SpecialRootFolder properties. To repopulate the list view, call Refresh.

See also: Refresh

ComboBox property

```
property ComboBox : TStCustomShellComboBox
```

The TStShellComboBox component associated with the list view.

When a TStShellComboBox is associated with the list view, the list view will automatically update when a folder is selected in the combo box. Likewise, if the folder in the list view changes, the combo box will be updated to show the folder that the list view is showing.

See also: TreeView

CompressedColor property

```
property CompressedColor : TColor
```

Default: clBlue

The text color of compressed folders when the loColorCompressed option is on.

The text of compressed folders will be drawn with CompressedColor. Compressed folders are only drawn in this color if the Options property contains the loColorCompressed value. CompressedColor is not available in versions of Delphi and C++Builder prior to version 4.

See also: Options

CopyToClipboard method

```
procedure CopyToClipboard;
```

Copies the currently selected shell item to the clipboard.

When you call CopyToClipboard, the currently selected list view item will be copied to the Windows clipboard as an item identifier list (a pidl). It can then be pasted into any application that allows pasting of shell item identifiers. For example, you can copy a file item to the clipboard and then paste the item into a folder in Windows Explorer. The CanCopy property of TStShellItem can be used to determine if the selected item can be copied to the clipboard.

See also: CutToClipboard, PasteFromClipboard, TStShellItem.CanCopy

CutToClipboard method

```
procedure CutToClipboard;
```

Copies the currently selected shell item to the clipboard and removes the item after a clipboard paste operation.

When you call CutToClipboard, the currently selected list view item will be copied to the Windows clipboard as an item identifier list (a pidl). The item will not be removed from its current location until after a paste operation has been performed. Once in the clipboard, the item can be pasted into any application that allows pasting of shell item identifiers. For example, you can cut a file to the clipboard and then paste it into a folder in Windows Explorer. The CanCopy property of TStShellItem can be used to determine if the selected item can be copied to the clipboard.

See also: CopyToClipboard, PasteFromClipboard, TStShellItem.CanCopy

DeleteItem method

```
function DeleteItem(
  Recycle : Boolean; Confirm : Boolean) : Boolean;
```

↳ Deletes the currently selected item in the list view.

Call DeleteItem to delete the currently selected item in the list view. Recycle determines whether the deleted item is sent to the Recycle Bin. Confirm determines whether a confirmation dialog will be displayed before the item is deleted. DeleteItem returns True if the item was deleted or False if the item could not be deleted.

See also: AddFolder, RenameItem

FileFilter property

```
property FileFilter : string
```

Default: Empty string

↳ Used to specify a filter that will be applied to the list view items.

Set FileFilter to one or more file types that should be included in the list view. All other file types will be removed from the view. FileFilter should consist of one or more file specs. If more than one file spec is supplied, the file specs must be separated by a semi-colon. For example:

```
  StShellListView.FileFilter := '*.txt;*.bat;*.ini';
```

See also: Filtered, OnFilterItem

Filtered property

```
property Filtered : Boolean
```

↳ Determines whether a filter should be applied to the list view items.

When Filtered is True, the items in the list view will be filtered in one of two ways. If an event handler is supplied for the OnFilterItem event, that event handler will be called as each item in the list view is enumerated. You can provide code in an OnFilterItem event handler to filter the items that are displayed in the list view. If OnFilterItem is not assigned, the contents of the FileFilter property will be used to filter the list. For simple filters, use the FileFilter property. For more complex filters, use the OnFilterItem event.

See also: FileFilter, OnFilterItem

Folder read-only, run-time property

```
property Folder : TStShellFolder
```

↳ A pointer to the TStShellFolder that represents the folder being displayed in the list view.

Read Folder to obtain information about the folder currently displayed in the list view.

See also: ShellItems

LoadFolder method

```
procedure LoadFolder(Folder : TStShellFolder);
```

```
TStShellFolder = class;
```

↳ Loads the contents of a TStShellFolder into the list view.

Normally you will use the RootFolder or SpecialRootFolder properties to load a folder into the list view. Use LoadFolder to populate the list view if you already have a TStShellFolder object to work with.

See also: RootFolder, SpecialRootFolder, TStShellFolder

MaxNotifications property

```
property MaxNotifications : Integer
```

Default: 5

↳ The maximum number of shell event notifications that will be processed at one time.

TStShellListView uses an instance of the TStShellNotification component internally to watch for shell events. When a folder is created, for example, the shell sends notification of this fact. The TStShellNotification component detects this event and notifies TStShellListView so it can update its display. For events such as creating files or folders only one or two such notification events are sent. For some operations—deleting a large number of folders, for example—hundreds of notifications might be sent. In this case it is not necessary for TStShellListView to receive notification of every file or folder deletion. It is only necessary that it receive one notification so that it can update its display. You can adjust MaxNotifications from its default of 5 if you find that the list view is not updating properly for certain shell operations. Set MaxNotifications to 0 for unlimited notifications.

MoveUpOneLevel method

```
procedure MoveUpOneLevel;
```

↳ Displays the contents of the current folder's parent.

MoveUpOneLevel can be used to navigate backwards through the shell hierarchy. For example, the standard file open dialog has a button that allows the user to move up one level from the current folder. If you were building a custom file open dialog, you would call MoveUpOneLevel to accomplish that effect.

OnFilterItem event

```
property OnFilterItem : TStEnumItemEvent
```

```
TStEnumItemEvent = procedure(Sender : TObject;
  ShellItem : TStShellItem; var Accept : Boolean) of object;
```

↳ Defines an event handler that is called for each item in the list view as the list is enumerated.

The event handler assigned to OnFilterItem will be called for each item as the list view is enumerated, and when the Filtered property is True. Sender is the component that generated the event. ShellItem is a pointer to the TStShellItem object representing the shell item. Accept determines whether the item is added to the component's ShellItems list (and whether it appears in the list view). You can use OnFilterItem to filter the list of items in the list view based on any characteristics you choose. The following example shows how to filter the list so that it contains only files that begin with the letter 't' and have an extension of ".TXT":

```
  procedure TForm1.StShellListView1FilterItem(
    Sender: TObject; ShellItem: TStShellItem;
    var Accept: Boolean);
  begin
    Accept :=
      (UpperCase(ExtractFileExt(ShellItem.Path)) = '.TXT')
      and (UpperCase(ShellItem.DisplayName[1]) = 'T');
  end;
```

See also: FileFilter, Filtered, TStShellItem

```
property OnItemDblClick : TStItemDblClickEvent

TStItemDblClickEvent = procedure(Sender : TObject;
  Item : TStShellItem; var DefaultAction : Boolean) of object;
```

Defines an event handler that is called when an item in the list view is double clicked.

Use this event when you want to modify the default behavior when an item in the list view is double clicked. The default double click behavior depends on the type of item selected. In the case of executable files, the file is simply run. In the case of files types with an associated application, the associated application is run and the file loaded. For file types without an association, the shell’s Open With dialog box is displayed. For folders, the folder is opened in a new Explorer window.

Sender is the component that generated the event. Item is a TStShellItem object representing the item that was double clicked. DefaultAction determines whether the default double click action should take place. DefaultAction is True by default. Set it to False to suppress the default action.

The following example shows how to disable the default action when an executable file is double clicked:

```
procedure TForm1.StShellListView1ItemDblClick(Sender: TObject;
  Item: TStShellItem; var DefaultAction: Boolean);
begin
  if Item.IsFile then
    if UpperCase(ExtractFileExt(Item.Path)) = '.EXE' then
      DefaultAction := False;
end;
```

See also: Execute, OnItemSelected

OnItemSelected event

```
property OnItemSelected : TStItemSelectedEvent
```

```
TStItemSelectedEvent = procedure(
  Sender : TObject; Item : TStShellItem) of object;
```

↳ Defines an event handler that is called when the selected item changes.

The method assigned to the OnItemSelected event is called each time the list view's selection changes. Sender is the component that generated the event. Item is a pointer to the TStShellItem object representing the selected item. Respond to the OnItemSelected event when you want notification that the selected item has changed. The following example updates two labels on a form with the item's display name and path when the list view's selection changes:

```
  procedure TForm1.StShellTreeView1ItemSelected(
    Sender: TObject; Item: TStShellItem);
  begin
    Label1.Caption := Item.DisplayName;
    Label2.Caption := Item.Path
  end;
```

The SelectedItem property can be used to obtain information about the currently selected item.

See also: SelectedItem, TStShellItem

OnListFilled event

```
property OnListFilled : TNotifyEvent
```

↳ Defines an event handler that is called when the list view has been filled.

Provide an event handler for the OnListFilled event if you need notification that the folder has been completely enumerated and the list view filled with the results of the enumeration.

OnShellChangeNotify event

```
property OnShellChangeNotify : TStShellNotifyEvent3
```

```
TStShellNotifyEvent = procedure(Sender : TObject;
  OldShellItem : TStShellItem; NewShellItem : TStShellItem;
  Events : TStNotifyEventsSet) of object;
```

Defines an event handler that is called when an event occurs in the shell.

The method assigned to the OnShellChangeNotify event is called each time a shell event occurs. A shell event may occur based on a folder's contents changing, media being inserted into a removable media drive, a file or folder being added, renamed, or deleted, network drives being mapped, and so on. Sender is the component that generated the event. OldShellItem is a pointer to the TStShellItem representing the shell item prior to the change (the original name of a folder that is being rename, for example). NewShellItem is a pointer to the TStShellItem object representing the new shell object. In some cases OldShellItem and NewShellItem will contain identical properties. In other cases NewShellItem could be nil so be sure to check for nil before using this parameter. Events is a set indicating which shell events occurred. See TStShellNotification for more information on the types of shell events that can occur.

See also: TStShellItem, TStShellNotification

OpenDialogMode property

```
property OpenDialogMode : Boolean
```

Default: False

Determines if the list view should behave as the list view in the common file open dialog box behaves.

Set OpenDialogMode to True if you are using the TStShellListView on a form that emulates the Windows file open dialog. Double-clicking an item in the list view normally causes the item to be executed, or its associated application to be executed in the case of document files. When OpenDialogMode is True the shell does not attempt to execute the item.

Optimization property

```
property Optimization : TStOptimization
```

```
TStOptimization = (otEnumerate, otDisplay);
```

Default: otEnumerate

↳ Determines whether the list view will be optimized for speed of enumeration or for display speed.

The Optimization property can be used to optimize the performance of the list view. A great deal of processor time is required to retrieve the icons for folder items. When Optimization is otEnumerate (the default), enumeration of shell folders is much faster because the icons for each item are not actually retrieved until the item is displayed in the list view. This is particularly important for folders containing a large number of items (over 2,000 items, for example). The drawback to this optimization type is that the list view will paint more slowly when scrolled. When Optimization is otDisplay, the list view displays items more quickly but it can take a long time to enumerate large folders. Leave Optimization set to otEnumerate unless you are sure you will be dealing with folders containing relatively few items.

Optimization is not available for versions of Delphi and C++Builder prior to version 4. This is due to the fact that the virtual list view is not supported in older versions of the VCL.

```
property Options : TStListOptionsSet
```

```
TStListOptions = (loAllowDrag,loAllowDrop,loAllowRename,
  loColorCompressed, loExtendedMenu, loOpenFoldersInNewWindow,
  loShellMenu, loShowHidden, loSortTypeByExt);
```

```
TStListOptionsSet = set of TStListOptions;
```

Default: [loAllowRename, loAllowDrag, loAllowDrop, loShellMenu]

Determines how the TStShellListView operates.

Options controls the TStShellListView display properties, and whether items in the list view can be renamed using in-place editing.

The following table describes each of the possible options:

Options	**Meaning**
loAllowDrag	Determines whether the component is a drag source. When this option is set, users can drag items from the list view to folders within the list view, to a TStShellTreeView, or to other applications.
loAllowDrop	Determines whether shell items can be dropped on folders in the list view. The dragged items can come from within the list view, from a TStShellTreeView, from Windows Explorer, or from any application that allows dragging of file items.
loAllowRename	Determines whether the user can rename items using in-place editing.
loColorCompressed	When this option is set, compressed folders are displayed in the color specified by the CompressedColor property. This option has no effect in versions of the VCL prior to 4.
loExtendedMenu	When this option is set, the shell context menu will show additional items if the Shift key is held down when the context menu is invoked. For example, under Windows NT the context menu for a file system object will show additional items called Compress and Uncompress. This option is ignored if Options does not include toShellMenu.

`loOpenFoldersInNewWindow`	`When this option is set, double-clicking a folder in the list view will cause Windows to open a new window containing the folder. When this option is off (the default), double-clicking a folder in the list view causes the list view to display the new folder.`
`loShellMenu`	`When this option is set, right-clicking an item in the tree view will invoke the shell's context menu for that item.`
`loShowHidden`	`Determines whether hidden folders and items are displayed.`
`loSortTypeByExt`	`When this option is set, the Type column in report mode is sorted by the actual extension of the file system item rather than by the type name. By contrast, Windows Explorer only sorts by the type name. This places all executables near the top of the list because the type name is of executables is "Application."`

See also: CompressedColor

PasteFromClipboard method

```
procedure PasteFromClipboard;
```

↳ Pastes a shell item in the clipboard into the currently selected list view item.

When you call PasteFromClipboard, the shell item identifier in the clipboard will be pasted to the folder represented by the currently selected list view item. The item in the clipboard may have been placed there by your program or by an external program such as Windows Explorer. The CanPaste property of TStShellItem can be used to determine whether or not the item represented by the currently selected node allows pasting.

See also: CanPaste, CopyToClipboard, CutToClipboard, TStShellItem.CanPaste

RenameItem method

```
function RenameItem(NewName : string; Confirm : Boolean) : Boolean;
```

↳ Renames an item in the list view.

Call RenameItem to rename the currently selected item in the list view. NewName is the new name that will be given to the item. Confirm determines whether a confirmation dialog will be displayed before the item is renamed. RenameItem returns True if the item was renamed, or False if the item could not be renamed.

See also: AddFolder, DeleteItem

Refresh — method

```
procedure Refresh;
```

Refreshes the contents of the folder being viewed in the list view.

Call Refresh to force a refresh of the folder being displayed in the list view. Refresh first clears the contents of the list view and then enumerates the folder to get the latest contents.

Note that the contents of the folder will be automatically updated when a file or folder is created, deleted, renamed, and so on. Therefore, it is not usually necessary to call Refresh from your applications.

See also: Clear

RootFolder — property

```
property RootFolder : string
```

Default: Empty string

The folder that will be displayed in the list view.

The file folder specified by RootFolder will be displayed in the TStShellListView when the list view is being used independent of a tree view. The folder can be specified by either the RootFolder property or the SpecialRootFolder property. Use RootFolder to specify a file folder, and SpecialRootFolder to specify a system folder (such as the Desktop or Network Neighborhood). If the specified folder is invalid, an EStInvalidFolder exception is raised. It is not necessary to set RootFolder when using the list view with an associated tree view.

See also: OnItemSelected, SpecialRootFolder

SelectedItem — read-only, run-time property

```
property SelectedItem : TStShellItem
```

A pointer to the TStShellItem representing the currently selected item in the list view.

Read SelectedItem to obtain information about the selected item in the TStShellListView. See the TStShellItem topic for details on the information provided by TStShellItem.

See also: OnItemSelected, TStShellItem

SelectedItems property

```
property SelectedItems : TStShellItemList
```

```
TStShellItemList = class
```

A list of items currently selected in the list view.

Use SelectedItems to enumerate the list of items currently selected in the list view. SelectedItems only has value if the MultiSelect property of the list view is set to True. The following example shows how to enumerate the selected items and put their display names in a memo component:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
begin
  for I := 0 to Pred(StShellListView1.SelectedItems.Count) do
    Memo1.Lines.Add(
      StShellListView1.SelectedItems[I].DisplayName);
end;
```

See also: SelectedItem, ShellItems

ShellItems read-only, run-time property

```
property ShellItems : TStShellItemList
```

The list of items in the list view.

ShellItems is a list of TStShellItem pointers, each of which represents an item in the shell folder. The most common use of ShellItems would be to iterate through all items in the list view. The following example iterates the items in the list view and adds the DisplayName of each item to a memo:

```
var
  I : Integer;
...
  for I := 0 to StShellListView1.ShellItems.Count - 1 do
    Memo1.Lines.Add(
      StShellListView1.ShellItems[I].DisplayName);
```

See the TStShellItem topic for details on the information available from TStShellItem.

See also: TStShellItem

ShellVersion — read-only, run-time property

```
property ShellVersion : Double
```

↳ The version number of Windows' SHELL32.DLL.

Read ShellVersion to determine the version of the Windows shell running on a particular machine. Some shell operations and special folders are only available on version 4.72 of the shell and later.

ShowPropertySheet — method

```
function ShowPropertySheet : Boolean;
```

↳ Displays the property sheet associated with the currently selected item in the list view.

When you call ShowPropertySheet, the property sheet for the selected item will be displayed. A property sheet is the dialog you see when you right-click an item in Windows Explorer and choose Properties from the context menu. Property sheets are modeless—your application will continue to operate normally while the property sheet is being displayed. ShowPropertySheet will return True if the item has a property sheet that can be displayed, or False if the item does not have a property sheet.

See also: TStShellItem.HasPropSheet

```
procedure SortColumn(
  Index : Integer; SortDir : TStSortDirection);
```

```
TStSortDirection = (sdAscending, sdDescending, sdToggle);
```

Sorts the specified column in either ascending or descending order when the list view is in details view.

Index is the column index of the column to sort. The first column is index 0, the second is index 1, and so on. SortDir is the direction of the sort. The following table lists the possible values for SortDir:

Value	Description
sdAscending	The contents of the folder will be sorted in ascending order. If the folder is a file system folder, folders will appear before files in the list.
sdDescending	The contents of the folder will be sorted in descending order. If the folder is a file system folder, files will appear before folders in the list.
sdToggle	Toggles the sort direction. If the folder was sorted in ascending order, it will now be sorted in descending order, and vice versa.

SpecialRootFolder property

```
property SpecialRootFolder : TStSpecialRootFolder
```

```
TStSpecialRootFolder = (
  sfAltStartup, sfAppData, sfBitBucket, sfCommonAltStartup,
  sfCommonDesktopDir, sfCommonFavorites, sfCommonPrograms,
  sfCommonStartMenu, sfCommonStartup, sfControls, sfCookies,
  sfDesktop, sfDesktopDir, sfDrives, sfFavorites, sfFonts,
  sfHistory, sfInternet, sfInternetCache, sfNetHood, sfNetwork,
  sfNone, sfPersonal, sfPrinters, sfPrintHood, sfPrograms,
  sfRecentFiles, sfSendTo, sfStartMenu, sfStartup, sfTemplates);
```

Default: sfNone

Specifies the shell folder that will be displayed in the list view.

The system folder specified by SpecialRootFolder will be displayed in the TStShellListView. The folder the list view displays can be specified by either the RootFolder property or the SpecialRootFolder property. Use RootFolder to specify a file folder, and SpecialRootFolder to specify a system folder (such as the Desktop or Network Neighborhood). The list of special root folders is a list of system folders defined by Windows. Not every folder in the list is available on all versions of the shell. If a particular folder is invalid, an EStShellError exception is raised. It is not necessary to specify SpecialRootFolder when the list view is associated with a tree view.

See also: RootFolder

TreeView property

```
property TreeView : TStCustomShellTreeView
```

The TStShellTreeView associated with the list view.

Set TreeView to the TStShellTreeView you want associated with the list view. When a tree view is associated with the list view, the list view is automatically updated when the tree view selection changes. Double-clicking a folder in the list view will change the list view contents and will expand the tree view to display the new folder.

See also: TStShellTreeView

TStShellComboBox Component

The TStShellComboBox component emulates the combo box found in Windows Explorer. TStShellComboBox is designed for use with a TStShellListView. When a folder is selected in the combo box, the associated list view will be updated to show the contents of the selected folder. Interaction between TStShellComboBox and TStShellListView is automatic and requires virtually no code. Figure 3.2 shows a custom file open dialog box that implements a TStShellComboBox and a TStShellListView.

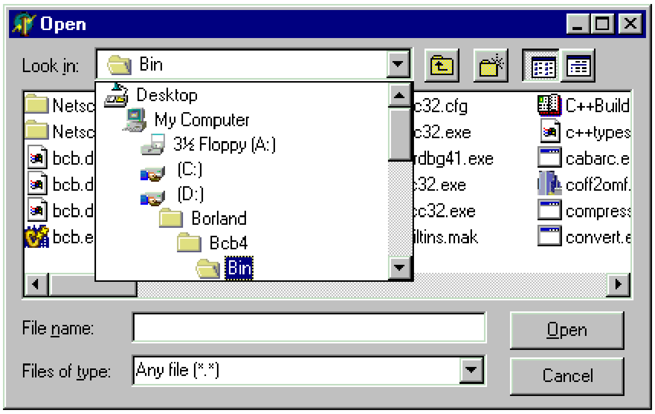


Figure 3.2: Custom file open dialog box.

TStShellComboBox has very few properties and methods since it is designed to interact with TStShellListView automatically.

Hierarchy

TCustomComboBox (VCL)

  TStCustomShellComboBox (StShlCtl)

    TStShellComboBox (StShlCtl)

Properties

ListView	SelectedFolder	ShellVersion

Events

OnFolderSelected

ListView property

```
property ListView : TStCustomShellListView
```

↳ The TStShellListView that is associated with the combo box.

Set ListView to the TStShellListView that should be associated with the combo box. If ListView is assigned, the list view will automatically be updated when the folder selection in the combo box changes.

OnFolderSelected event

```
property OnFolderSelected : TStFolderSelectedEvent

TStFolderSelectedEvent = procedure(
  Sender : TObject; Folder : TStShellFolder) of object;
```

↳ Defines an event handler that is called when the selected folder changes.

The method assigned to the OnFolderSelected event is called each time the combo box's selection changes. Sender is the component that generated the event. Folder is a pointer to the TStShellFolder object representing the selected folder. Respond to the OnFolderSelected event when you want notification that the selected folder has changed. The following example updates two labels on a form with the selected folder's display name and path when the combo box's selection changes:

```
procedure TForm1. StShellComboBox1FolderSelected(
  Sender: TObject; Folder: TStShellFolder);
begin
  Label1.Caption := Folder.DisplayName;
  Label2.Caption := Folder.Path
end;
```

You can also use the SelectedFolder property to obtain information about the currently selected folder.

See also: SelectedFolder, TStShellFolder

SelectedFolder — read-only, run-time property

```
property SelectedFolder : TStShellFolder
```

↳ A pointer to the TStShellFolder representing the currently selected folder in the combo box.

Read SelectedFolder to obtain information about the folder or item in the TStShellComboBox that is currently selected.

See also: OnFolderSelected, TStShellFolder

ShellVersion — read-only, run-time property

```
property ShellVersion : Double
```

↳ The version number of Windows' SHELL32.DLL.

Read ShellVersion to determine the version of the Windows shell running on a particular machine. Some shell operations and special folders are only available on version 4.72 of the shell and later.

TStShellNavigator Component

The TStShellNavigator component is a TPanel descendant that emulates the top of Windows' common file dialog boxes. It contains a combo box (a TStShellComboBox) and buttons for moving up a level, creating a new folder, changing the view style, and moving back to the last folder visited. The Buttons property determines which buttons are shown. The Style property allows you to choose between the Windows 95/98 style, the Windows 2000 style, and the Windows XP style. Figure 3.3 shows a TStShellNavigator on a form at run time (in Windows 2000 style).

The TStShellNavigator component emulates the top portion of the common file dialog boxes.

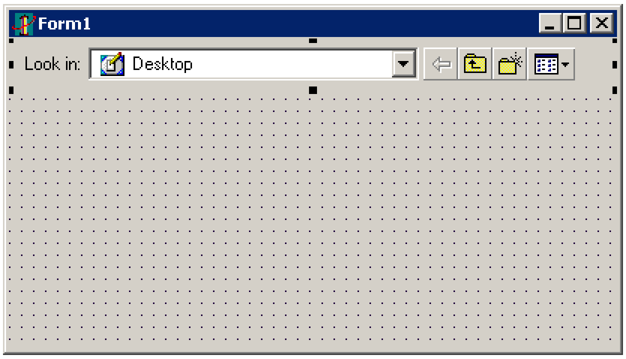


Figure 3.3: TStShellNavigator on form at run time.

TStShellNavigator can be used alone, but it is designed for use with a TStShellListView component. Place a TStShellNavigator on a form, add a TStShellListView, and set the navigator's ListView property to the list view just added. The two components work together; changing the folder in navigator results in the list view changing to that folder, and vice versa. The buttons on the navigator automatically effect changes in the list view (changing the view style, for example). The interaction between the navigator and list view is automatic. It is not necessary to write code to obtain the behavior you would expect from these two components.

TStShellNavigator can be used to build custom forms or dialog boxes. If you want to build a custom file open or save dialog, though, you should use TStDialogPanel instead.

Hierarchy

TCustomPanel (VCL)

  TStCustomShellNavigator (SsShlDlg)

    TStShellNavigator (SsShlDlg)

Properties

BackButton	DetailsButton	MoveUpButton
Buttons	LeftOffset	NewFolderButton
ComboBox	ListButton	Style
ComboBoxLabel	ListView	ViewButton

Events

OnButtonClick	OnFolderSelected	OnViewStyleChanging

Reference Section

BackButton — read-only, run-time property

```
property BackButton : TSsSpeedButton
```

```
TSsSpeedButton = class(TSpeedButton)
```

↳ The Back button on the navigator.

A list of folders visited is maintained within TStShellNavigator. When the Back button is clicked, the previous folder visited is loaded in the associated list view. The combo box also changes to reflect the new folder.

The behavior of this button is automatic. It is not normally necessary to get to the properties of the button itself. However, access to the button is provided in case you need to perform some specialized processing in your applications. If you wish to suppress the default behavior for the button, use the OnButtonClick event and set the DefaultAction parameter to False.

See also: Buttons, DetailsButton, ListButton, MoveUpButton, NewFolderButton, ViewButton

Buttons — property

```
property Buttons : TSsNavigatorButtonsSet
```

```
TSsNavigatorButtons = (
  nbBack, nbMoveUp, nbNewFolder, nbList, nbDetails, nbView);
```

```
TSsNavigatorButtonsSet = set of TSsNavigatorButtons;
```

Default: nbBack, nbMoveUp, nbNewFolder, nbList, nbDetails, nbView

↳ Determines the visible buttons on the navigator bar.

Use the Buttons property to determine which buttons are visible on the navigator bar. You may not want to allow users to create a new folder, for example. In this case you can remove the nbNewFolder value from the Buttons property to hide the New Folder button.

The buttons available on the navigator vary with the value of the Style property. If the Style property is set to nsWin2k or nsWinXP, the nbList and nbDetails values are ignored. Similarly, if Style is Win9x, the nbView value is ignored.

The following table describes the buttons and explains to which style they apply:

Value	Action	Style
nbBack	Moves back to the last folder visited.	All
nbMoveUp	Moves to the next highest folder in the shell tree.	All
nbNewFolder	Creates a new folder under the folder being shown in the list view.	All
nbList	Changes the list view to list (small icon) view.	nsWin9x only
nbDetails	Changes the list view to details (report) view.	nsWin9x only
nbView	Allows the user to select one of four view styles for the list view.	nsWin2k only

See also: BackButton, DetailsButton, ListButton, MoveUpButton, NewFolderButton, Style, ViewButton

ComboBox — read-only, run-time property

```
property ComboBox : TSsShellComboBox
```

```
TSsShellComboBox = class(TStShellComboBox)
```

The combo box on the navigator that contains drives and special folders.

The ComboBox property is a TStShellComboBox descendant. It contains folders as seen in Windows common dialog boxes, Explorer, and so on. If the ListView property is assigned, selecting a folder in the combo box will result in the list view loading the selected folder. See TStShellComboBox for full information on the capabilities of the combo box.

The behavior of the combo box is automatic when a TStShellListView is attached to the navigator. Access to the combo box is provided in case you need to perform some specialized processing in your applications.

See also: ComboBoxLabel, TStShellComboBox

ComboBoxLabel — run-time property

```
property ComboBoxLabel : TLabel
```

Default: “Look in:”

↳ The label displayed to the left of the navigator’s combo box.

It is not generally necessary to change the value of this label.

See also: ComboBox

DetailsButton — read-only, run-time property

```
property DetailsButton : TSsSpeedButton
```

```
TSsSpeedButton = class(TSpeedButton)
```

↳ The Details button on the navigator.

The Details button changes the associated list view to vsReport style (details view). It is only available when the Style property is nbWin9x.

The behavior of this button is automatic. It is not normally necessary to get to the properties of the button itself. However, access to the button is provided in case you need to perform some specialized processing in your applications. If you wish to suppress the default behavior for the button, use the OnButtonClick event and set the DefaultAction parameter to False.

See also: BackButton, Buttons, ListButton, MoveUpButton, NewFolderButton, Style, ViewButton

LeftOffset — property

```
property LeftOffset : Integer
```

↳ The horizontal position of the first component on the navigator.

Change LeftOffset if you wish to move the components on the navigator along their horizontal axis. The navigator’s label, combo box, and buttons are all created relative to one another. Changing LeftOffset results in all components being moved.

ListButton read-only, run-time property

```
property ListButton : TSsSpeedButton
```

```
TSsSpeedButton = class(TSpeedButton)
```

The List button on the navigator.

The List button changes the associated list view to vsList style. It is only available when the Style property is nbWin9x.

The behavior of this button is automatic. It is not normally necessary to get to the properties of the button itself. However, access to the button is provided in case you need to perform some specialized processing in your applications. If you wish to suppress the default behavior for the button, use the OnButtonClick event and set the DefaultAction parameter to False.

See also: BackButton, Buttons, DetailsButton, MoveUpButton, NewFolderButton, Style, ViewButton

ListView property

```
property ListView : TStCustomShellListView
```

```
TStCustomShellListView = class(TCustomListView, IDropTarget)
```

The list view that will be automatically updated when the navigator's combo box and buttons are used.

ListView is a TStCustomShellListView descendant (a TStShellListView in most cases). It is not necessary to use TStShellNavigator with a list view but it is designed for this purpose. Once ListView is assigned, changes to the navigator will automatically be reflected in the list view. Changing the folder in the navigator's combo box, for example will result in the list view loading the contents of that folder, clicking one of the view style buttons will result in the list view's view changing, and so on.

Changes to the list view will also result in changes to the navigator's combo box. For example, double-clicking a folder in the list view will cause the list view to load the new folder's contents and will update the combo box accordingly.

MoveUpButton — read-only, run-time property

```
property MoveUpButton : TSsSpeedButton
```

```
TSsSpeedButton = class(TSpeedButton)
```

↳ The Move Up button on the navigator.

The Move Up button moves the associated list view to the parent of the current folder. The contents of the new folder are loaded into the list view and the combo box is updated accordingly.

The behavior of this button is automatic. It is not normally necessary to get to the properties of the button itself. However, access to the button is provided in case you need to perform some specialized processing in your applications. If you wish to suppress the default behavior for the button, use the OnButtonClick event and set the DefaultAction parameter to False.

See also: BackButton, Buttons, DetailsButton, ListButton, NewFolderButton, ViewButton

NewFolderButton — read-only, run-time property

```
property NewFolderButton : TSsSpeedButton
```

```
TSsSpeedButton = class(TSpeedButton)
```

↳ The New Folder button on the navigator.

Clicking the New Folder button causes a new folder to be created in the list view. This assumes, of course, that the current folder is a file system folder. A new folder is created with a default name and the user can then rename the folder.

The behavior of this button is automatic. It is not normally necessary to get to the properties of the button itself. However, access to the button is provided in case you need to perform some specialized processing in your applications. If you wish to suppress the default behavior for the button, use the OnButtonClick event and set the DefaultAction parameter to False.

See also: BackButton, Buttons, DetailsButton, ListButton, MoveUpButton, ViewButton

```
property OnButtonClick : TSsNavigatorButtonClickEvent
```

```
TSsNavigatorButtonClickEvent = procedure(
  Sender : TObject; Button : TSsNavigatorButtons;
  var DefaultAction : Boolean) of object;
```

```
TSsNavigatorButtons = (
  nbBack, nbMoveUp, nbNewFolder, nbList, nbDetails, nbView);
```

```
TSsNavigatorButtonsSet = set of TSsNavigatorButtons;
```

Defines an event handler that is called when one of the navigator's buttons is clicked.

Sender is the component that generated the event. Button is the button that was clicked. DefaultAction determines whether the button's automatic behavior takes place. DefaultAction is True by default. Set DefaultAction to False to suppress the button's normal processing.

Each of the navigator's buttons operates automatically when a list view is attached to the ListView property. In some cases, however, you may wish to override the default behavior for a button. Use the OnButtonClick event to suppress the default behavior for a button. Note that the View button (nsWin2k and nsWinXP styles) has a popup menu. Selecting a view style from the View button's popup menu will result in the OnButtonClick event firing, but the OnViewStyleChanging event provides information on the view style that was selected.

The following example shows how to suppress the New Folder button's default behavior:

```
  procedure TForm1.StShellNavigator1ButtonClick(Sender: TObject;
    Button: TSsNavigatorButtons; var DefaultAction: Boolean);
  begin
    if Button = nbNewFolder then begin
      DefaultAction := False;
      // Your customized New Folder processing
    end;
  end;
```

See also: BackButton, Buttons, DetailsButton, ListButton, ListView, MoveUpButton, NewFolderButton, OnViewStyleChanging, ViewButton

OnFolderSelected event

```
property OnFolderSelected : TStFolderSelectedEvent
```

```
TStFolderSelectedEvent = procedure(
  Sender : TObject; Folder : TStShellFolder) of object;
```

Defines an event handler that is called when a folder is selected in the combo box.

Sender is the component that generated the event. Folder is a pointer to a TStShellFolder object representing the selected folder.

OnViewStyleChanging event

```
property OnViewStyleChanging : TSsNavigatorViewStyleChangingEvent
```

```
TSsNavigatorViewStyleChangingEvent = procedure(
  Sender : TObject; Style : TViewStyle;
  var DefaultAction : Boolean) of object;
```

Defines an event handler that is called when an item on the View button's popup menu is selected.

Sender is the component that generated the event. Style is the list view style that was selected from the View button (see TViewStyle in the VCL help for more information). DefaultAction determines whether the associated list view's layout will change as a result of the selection. DefaultAction is True by default. Set DefaultAction to False to prevent the default action from taking place.

Style property

```
property Style : TSsNavigatorStyle
```

```
TSsNavigatorStyle = (nsWin9x, nsWin2k, nsWinXP);
```

Sets the look and feel of the navigator buttons.

The Style property allows you to choose from three styles: Windows 95/98, Windows 2000, and Windows XP. The Style property determines the buttons visible on the navigator and their images. See the Buttons property for a description of the buttons displayed for each style.

See also: Buttons

ViewButton — read-only, run-time property

```
property ViewButton : TSsMenuButton

TSsMenuButton = class(TCustomControl)
```

The speed button associated with the View button on the navigator.

The View button (Windows 2000 and Windows XP styles only) is a button with a drop-down menu. The menu contains items named Large Icons, Small Icons, List, and Details. Selecting an item on the drop-down menu sets the associated list view's ViewStyle property and updates the list view's display with the new style.

The behavior of this button is automatic. It is not normally necessary to get to the properties of the button itself. However, access to the button is provided in case you need to perform some specialized processing in your applications. If you wish to suppress the default behavior for the button, use the OnViewStyleChanging event and set the DefaultAction parameter to False.

See also: BackButton, Buttons, DetailsButton, ListButton, MoveUpButton, NewFolderButton, OnViewStyleChanging

TStDialogPanel Component

The TStDialogPanel component is a TPanel descendent that emulates the controls found on the Windows common file dialog boxes. Its primary purpose is to build custom file open and save dialog boxes. Add a TStDialogPanel to a form and add other components to the form as needed. The Style property allows you to choose between the Windows 95/98 style, the Windows 2000 style, and the Windows XP style. The different styles determine the buttons that are displayed, and the glyphs for those buttons. Figure 3.4 shows a TStDialogPanel on a form at design time (Windows 2000 style).

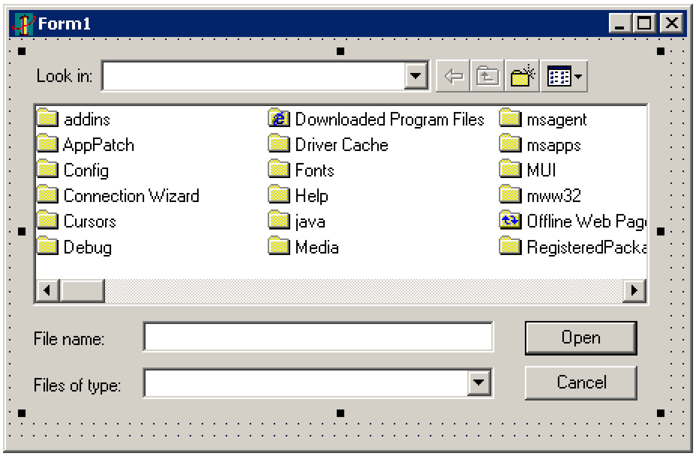


Figure 3.4: TStDialogPanel on form at design time.

The TStDialogPanel component emulates the controls found on the Windows common dialog boxes.

TStDialogPanel is designed to work automatically. In most cases you only have to place the component on a form and read the FileName property when the form closes. TStDialogPanel can be used to create both Open and Save dialog boxes. Simply change the OpenButtonCaption property as needed.

TStDialogPanel has many properties in common with TOpenDialog and TSaveDialog. Those properties include DefaultExt, FileName, Filter, FilterIndex, and InitialDir. These properties work exactly like their TOpenDialog and TSaveDialog counterparts.

To use TStDialogPanel, place it on a form to be used in your application. Show the form and read the ModalResult property of the form. If ModalResult is mrOk, read the panel's FileName property to determine the selected file. For example:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Form2.ShowModal = mrOk then
    Label1.Caption := Form2.StDialogPanel1.FileName;
end;
```

For the most part, TStDialogPanel operates like the common file dialog boxes. There are some differences, though. The file name edit control, for example, does not support incremental file and folder searches.

Hierarchy

TCustomPanel (VCL)

  TStCustomDialogPanel (SsShlDlg)

    TStDialogPanel (SsShlDlg)

Properties

AllowResize	FileTypeComobBox	Navigator
CancelButton	FileTypeLabel	OpenButton
DefaultExt	Filter	OpenButtonCaption
FileName	FilterIndex	ParentForm
FileNameEdit	InitialDir	Style
FileNameLabel	ListView	

Events

OnItemClick	OnOpenButtonClick
OnItemDblClick	OnCancelButtonClick

Reference Section

AllowResize property

```
property AllowResize : Boolean
```

Default: True

↳ Determines whether the controls on the panel resize when the component is resized.

When AllowResize is True, the controls on the panel resize and reposition as the panel is resized. A common use for TStDialogPanel is to place it on a form and set its Align property to alClient. As the parent form is resized, the TStDialogPanel's components resize to fit the new space. No attempt is made to restrict the maximum or minimum size of the panel. That task is left to the developer. When AllowResize is False, the controls keep their original size and position regardless of the size of the panel.

CancelButton read-only, run-time property

```
property CancelButton : TSsPanelButton
```

```
TSsPanelButton = class(TButton)
```

↳ The Cancel button on the panel.

When the Cancel button is clicked, the TStDialogPanel component attempts to determine the form on which the panel resides. If the form can be determined, its ModalResult property is set to mrCancel and the form is closed. This takes place when the Esc key is pressed as well. If the form cannot be determined, no action takes place. Rather than relying on TStDialogPanel to determine the parent form, you can specify the parent form via the ParentForm property.

The behavior of this button is automatic. It is not normally necessary to get to the properties of the button itself. However, access to the button is provided in case you need to perform some specialized processing in your applications. If you wish to suppress the default behavior for the button, use the OnCancelButtonClick event and set the DefaultAction parameter to False.

See also: OpenButton, ParentForm

DefaultExt property

```
property DefaultExt : string
```

Default: Empty string

↳ The default extension that will be appended to a file name typed into the edit box if no extension is provided.

Users can select a file name from the list view or type one into the file name edit control directly. DefaultExt will be appended to the end of the file name typed in directly, if no extension is provided.

See also: FileName, FileNameEdit

FileName property

```
property FileName : string
```

Default: Empty string

↳ The path and file name of the selected file.

Read FileName to determine the file that was selected by the user. Set FileName prior to showing the form if you want a file name to appear in the file name edit control when the component is first displayed.

The following example illustrates the use of the FileName property. It assumes a TStDialogPanel that resides directly on a form.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Form2.ShowModal = mrOk then
    Label1.Caption := Form2.StDialogPanel1.FileName;
end;
```

See also: FileNameEdit

FileNameEdit — read-only, run-time property

```
property FileNameEdit : TSsEdit
```

```
TSsEdit = class(TEdit)
```

↳ The edit control on the panel that contains the selected file.

FileNameEdit contains the name of the selected file. It is populated when the user selects a file from the list view. The user can also type a file name directly into the edit control (when the panel is being used as a Save dialog, for example). When the Open button is clicked or the Enter key is hit, the contents of FileNameEdit are copied to the FileName property. Also, the value of the FileName property is copied to the file name edit control when the panel is first displayed.

The behavior of FileNameEdit is automatic. The automatic behavior is determined by a number of factors, including the text entered. For example, if a folder name is typed into the edit control and that folder exists within the current folder, the list view changes to that folder when the user hits the Enter key. Similarly, if a full drive and folder is typed in, the list view switches to that folder. It should not be necessary to directly modify the properties of the file name edit control, but access to it is provided should you ever find the need to do so.

See also: FileName, FileNameLabel

FileNameLabel — read-only, run-time property

```
property FileNameLabel : TLabel
```

↳ The text label that appears to the left of the file name edit control.

Default: “File name:”

It will rarely be necessary to change any of the properties of this label. Access to it is provided in case you need to change any of its properties.

See also: FileNameEdit, FileTypeLabel

FileTypeComboBox read-only, run-time property

```
property FileTypeComboBox : TSsComboBox
```

```
TSsComboBox = class(TComboBox)
```

↳ The combo box that allows you to choose the type of file to display in the list view.

FileTypeComboBox operates in the same way as the “Files of type:” combo box on the standard Windows file dialog box. The Filter property determines the contents of the combo box. If Filter is an empty string, the combo box is empty. Selecting an item in the combo box filters the list of files shown in the list view.

The behavior of FileTypeComboBox is automatic. It should not be necessary to directly modify the properties of the combo box, but access to it is provided should you ever find the need to do so.

See also: FileTypeLabel, Filter, FilterIndex

FileTypeLabel read-only, run-time property

```
property FileTypeLabel : TLabel
```

↳ The label that appears to the left of the filter combo box.

Default: “Files of type:”

It will rarely be necessary to change any of the properties of this label. Access to it is provided in case you need to change any of its properties.

See also: FileNameLabel, FileTypeComboBox

```
property Filter : string
```

Default: Empty string

The filter string that is used to populate the filter combo box.

The Filter property determines the contents of the filter combo box. If Filter is an empty string, the combo box is empty. Selecting an item in the combo box filters the list of files shown in the list view.

The Filter property must follow a pre-defined format. Each filter consists of two parts, thereby making up a filter set. The first part is the text that will be displayed in the filter combo box. The second part is the file mask used to filter the list view. The two parts are separated with a pipe character (a vertical bar). Any number of filter sets can be provided. The filter sets are also separated with a pipe character. The filter for text files, then, would be:

```
Text files (*.txt)|*.txt
```

A filter for text files and all files would look like the following:

```
All files (*.*)|*.*|Text files (*.txt)|*.txt
```

To include multiple file masks in a single filter, separate the masks with semicolons.
For example:

```
All files (*.*)|*.*|Delphi files|*.pas;*.dpk;*.dpr
```

The Filter property can be set at design time (directly or using the property editor) or at run time via code.

See also: FilterIndex, FileTypeComboBox

FilterIndex property

```
property FilterIndex : Integer
```

Default: 0

↳ The index value of the filter displayed in the filter combo box when the panel is first displayed.

Set FilterIndex to the value of the filter you want initially shown in the filter combo box. The first filter is index 0, the second is index 1, and so on. For example, assume you have the following filter:

```
  All files (*.*)|*.*|Text files (*.txt)|*.txt
```

Given this filter, set FilterIndex to 1 to force the combo box to initially display "Text files (*.txt)" and for the filter to include only text files.

See also: FileTypeComboBox, Filter

InitialDir property

```
property InitialDir : string
```

Default: Empty string

↳ The folder that will initially be displayed in the list view when the panel is first shown.

Set InitialDir to the file system folder you want displayed in the list view when the panel is initially displayed. Setting InitialDir to "C:\WINNT", for example, will cause the list view to display the contents of the WINNT folder on drive C when the panel is initially displayed. InitialDir must include the full path and folder name (if applicable). The file specified in InitialDir must be a valid drive or drive and folder combination or an exception is raised.

See also: FileName

ListView read-only, run-time property

```
property ListView : TSsShellListView
```

```
TSsShellListView = class(TStShellListView)
```

↳ The list view that displays the contents of a folder.

ListView is a TStShellListView descendant. It displays the contents of a folder and behaves like the list view found on the standard Windows file dialog boxes.

The behavior of ListView is automatic. The default behavior depends on the action performed in the list view. Clicking on a file in the list view results in that file's name being copied to the file edit control. Double-clicking a file will result in the file's path and filename being copied to the FileName property and the Open button clicked. Double-clicking a folder results in that folder being loaded into the list view.

It should not be necessary to directly modify the properties of the list view, but access to it is provided should you ever find the need to do so. See TStShellListView on page 48 for complete information on the capabilities of the panel's list view.

See also: FileNameEdit, TStShellListView

Navigator read-only, run-time property

```
property Navigator : TStShellNavigator
```

```
TStShellNavigator = class(TStCustomShellNavigator)
```

↳ The navigator bar at the top of the panel.

Navigator is a TStShellNavigator component. It provides a combo box file folder selection, as well as buttons to create a new folder, change the list view's display style, moving up a level in the folder hierarchy, and so on. See TStShellNavigator on page 70 for a full description of the capabilities of this component.

The behavior of the Navigator is automatic. It should not be necessary to directly modify the properties of the list view, but access to it is provided should you ever find the need to do so.

OnItemClick event

```
property OnItemClick : TSsClickEvent
```

```
TSsClickEvent = procedure(
  Sender : TObject; var DefaultAction : Boolean) of object;
```

Defines an event handler that is called when an item in the list view is clicked.

Sender is the component that generated the event. DefaultAction determines whether the list view's default click behavior takes place. DefaultAction is True by default. See ListView for a description of the default single click behavior.

You can override the default behavior by responding to the OnItemClick event and setting the DefaultAction property to False.

See also: ListView

OnItemDblClick event

```
property OnItemDblClick : TSsClickEvent
```

```
TSsClickEvent = procedure(
  Sender : TObject; var DefaultAction : Boolean) of object;
```

Defines an event handler that is called when an item in the list view is double clicked.

Sender is the component that generated the event. DefaultAction determines whether the list view's default click behavior takes place. DefaultAction is True by default. See ListView for a description of the default double-click behavior.

You can override the default behavior by responding to the OnItemDlbClick event and setting the DefaultAction property to False.

See also: ListView

OnOpenButtonClick event

```
property OnOpenButtonClick : TSsClickEvent
```

```
TSsClickEvent = procedure(
  Sender : TObject; var DefaultAction : Boolean) of object;
```

Defines an event handler that is called when the Open button is clicked.

Sender is the component that generated the event. DefaultAction determines whether the default click behavior takes place. DefaultAction is True by default. This event is generated both as a result of the user clicking the Open button with the mouse and when the Enter key is pressed while the file name edit has focus. To override the default behavior, set DefaultAction to False.

See also: OnCancelButtonClick, OpenButton

OnCancelButtonClick event

```
property OnCancelButtonClick : TSsClickEvent
```

```
TSsClickEvent = procedure(
  Sender : TObject; var DefaultAction : Boolean) of object;
```

Defines an event handler that is called when the Cancel button is clicked.

Sender is the component that generated the event. DefaultAction determines whether the default click behavior takes place. DefaultAction is True by default. This event is generated both as a result of the user clicking the Cancel button with the mouse and when the Esc key is pressed. To override the default behavior, set DefaultAction to False.

See also: OnOpenButtonClick, OpenButton

OpenButton — read-only, run-time property

```
property OpenButton : TSsPanelButton
```

```
TSsPanelButton = class(TButton)
```

↳ The Open button on the panel.

When the Open button is clicked and the file name edit is not blank, the TStDialogPanel component attempts to determine the form on which the panel resides. If the form can be determined, its ModalResult property is set to mrOk, the complete path and filename of the selected file is copied to the FileName property, and the form is closed. This takes place when the Enter key is pressed as well. If the form cannot be determined, no action takes place. Rather than relying on TStDialogPanel to determine the parent form, you can specify the parent form via the ParentForm property.

The behavior of this button is automatic. It is not normally necessary to get to the properties of the button itself. However, access to the button is provided in case you need to perform some specialized processing in your applications. If you wish to suppress the default behavior for the button, use the OnOpenButtonClick event and set the DefaultAction parameter to False.

See also: CancelButton, FileName, OnOpenButtonClick, ParentForm

OpenButtonCaption — property

```
property OpenButtonCaption : string
```

Default: “Open”

↳ The caption displayed on the panel’s Open button.

TStDialogPanel can be used to create an Open dialog box or a Save dialog box. The term “Open button” is a bit misleading since it can be both an Open button and a Save button (or any other type of button you want). The difference is only in the caption displayed on the Open button. Other than that, the operation of the panel is exactly the same.

See also: OpenButton

ParentForm property

```
property ParentForm : TForm
```

↳ The form on which the panel resides.

When the Open or Cancel buttons are clicked, the TStDialogPanel component attempts to determine the form on which the panel resides. If the form can be determined, its ModalResult property is set to mrOk or mrCancel and the form is closed. If the form cannot be determined, no action takes place.

Rather than relying on TStDialogPanel to determine the parent form, you can specify the parent form via the ParentForm property. It is not necessary to assign a value to this property in all cases. Only use ParentForm in those cases where the default action does not occur when the Open and Cancel buttons are clicked.

See also: CancelButton, OpenButton

Style property

```
property Style : TSsNavigatorStyle
```

```
TSsNavigatorStyle = (nsWin9x, nsWin2k, nsWinXP);
```

Default: nsWin2k

↳ Sets the look and feel of the panel buttons.

The Style property allows you to choose from three styles: Windows 95/98, Windows 2000, and Windows XP. The Style property determines the buttons visible on the navigator and their images. It simply passes the style value on to the Navigator. See TStShellNavigator.Style and TStShellNavigator.Buttons property for a description of the styles and the buttons displayed for each style.

See also: TStShellNavigator.Buttons, TStShellNavigator.Style

Chapter 4: Non-Visual Components

ShellShock includes non-visual components that allow you to perform a number of shell operations. These components fit into two categories; components that invoke a shell dialog box or animation, and components that work entirely behind the scenes.

The first category of non-visual components includes TStShellAbout, TStBrowser, TStFileOperation, and TStFormatDrive. TStShellAbout simply displays the Windows About dialog box. TStBrowser is a wrapper around the Windows “Browse for Folder” dialog box. Its primary use is to allow your users to select a folder. TStFileOperation is used to copy, move, rename, or delete files. During lengthy operations, a progress window is displayed, complete with animation. This is the same progress window you see when moving, copying, or deleting files in Windows Explorer. TStFormatDrive is used to format a diskette. It displays the dialog box you see when you format a diskette from Explorer.

The second category of non-visual component includes TStDropFiles, TStShortcut, TStTrayIcon, TStShellNotification, and TStShellEnumerator. TStDropFiles allows your forms to accept files dropped from applications such as Windows Explorer. TStShortcut allows you to create, interrogate, and modify shortcuts. TStTrayIcon is used for creating applications that display an icon in the system tray. You can even simulate an animated tray icon.

Perhaps the most powerful non-visual components are TStShellNotification and TStShellEnumerator. TStShellNotification lets you watch the shell namespace for changes and notifies you of those changes. You can monitor the entire shell namespace, or a particular folder. For example, you can configure TStShellNotification to notify you when a new folder is created on the D drive. Or you can watch a specific folder for new file creations. TStShellEnumerator allows you to enumerate a shell folder to determine its contents. TStShellEnumerator is derived from TStCustomControl, the heart of the TStShellTreeView and TStShellListView components. TStShellEnumerator allow you to do behind the scenes what these two visual components do.

TStCustomShellController Component

TStCustomShellController is a component that is used internally and shared by the TStShellListView, TStShellTreView, and TStShellComboBox components. It should not be necessary to create a new instance of this component directly. However, it is documented here because it serves as the base class for the TStShellEnumerator component. Note that not all properties and methods are listed here, but only the most important.

Use of TStCustomShellController requires an understanding of shell item identifier lists (pidls) and the IShellFolder interface. Explaining these critical shell objects is beyond the scope of this manual.

Hierarchy

TComponent (VCL)

  TStCustomShellController (StShlCtl)

Properties

DesktopFolder	LargeFolderImages	SmallFolderImage

Methods

Create	GetFileInfo	ShowPropertySheet
GetDisplayName	RenameItem	

Reference Section

Create constructor

```
constructor Create(AOwner : TComponent); override;
```

Creates a TStCustomShellController object, passing AOwner as the owner of the component.

DesktopFolder read-only, run-time property

```
property DesktopFolder : IShellFolder
```

A pointer to the IShellFolder interface representing the shell namespace.

Use DesktopFolder if you need to call methods that require an IShellFolder representing the root of the shell namespace (the Desktop folder).

GetDisplayName method

```
function GetDisplayName(Folder : IShellFolder;
  Pidl : PItemIDList; Flags : DWORD) : string;
```

Returns the display name of an item identifier.

Call GetDisplayName to obtain the display name for an item in the shell namespace. Folder is the parent folder for the item identifier. Pidl is the item identifier list for the item relative to the parent folder. The Flags parameter is used to determine how the display name will be returned. Pass SHGDN_NORMAL to get the name of the item as displayed by Windows Explorer. Pass SHGDN_FORPARSING to get the path and file name for file system objects.

GetFileInfo method

```
procedure GetFileInfo(Pidl: PItemIDList;
  var Attributes : Cardinal; var IconIndex : Integer;
  var OpenIconIndex : Integer; var DisplayName : string);
```

↳ Retrieves information about a shell item.

Call GetFileInfo to retrieve a shell item's attributes, icon index, icon index of the item when it is selected (in a tree view, for example), and the display name. Pidl is a pointer to the item identifier list for the item. Pidl must be a fully qualified pidl (relative to the Desktop). When GetFileInfo returns, Attributes will contain the shell attributes for the item (see the IShellFolder:GetAttributesOf topic in the Win32 API Help for a list of attribute values). IconIndex will contain the index in the system image list for the item and OpenIconIndex will contain the index of the icon in the item's open state. DisplayName will contain the display name of the item as shown in Windows Explorer.

The IconIndex and OpenIconIndex values can be used in conjunction with the LargeFolderImages and SmallFolderImages properties to obtain the display icon for a shell item. DisplayName will be the same value returned by the GetDisplayName method when the SHGDN_NORMAL flag is passed.

See also: GetDisplayName, LargeFolderImages, SmallFolderImages

LargeFolderImages read-only, run-time property

```
property LargeFolderImages : TImageList
```

↳ The image list that contains the large icons for shell items.

LargeFolderImages is a list of large icons for shell items. The image list represented by LargeFolderImages is hooked to the system image list. Initially the list only contains a handful of icons. As additional icons are requested (by calling the GetFileInfo method) those icons will be added to the image list represented by LargeFolderImages.

See also: GetFileInfo, SmallFolderImages

RenameItem method

```
function RenameItem(
  SI : TStShellItem; NewName : string) : Boolean;
```

↳ Renames the specified item.

SI is the TStShellItem to rename. NewName is the new name for the item. RenameItem returns True if the item was successfully renamed, or False if an error occurred renaming the item.

The following example searches a folder for a file named TEXT.TXT and renames it to TEXT.BAK (using a TStShellEnumerator component):

```
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I : Integer;
    SI : TStShellItem;
  begin
    Enumerator.RootFolder := 'd:\';
    Enumerator.Execute;
    for I := 0 to Pred(Enumerator.ShellItems.Count) do begin
      SI := Enumerator.ShellItems[I];
      if SI.DisplayName = 'test.txt' then
        Enumerator.RenameItem(SI, 'test.bak');
    end;
  end;
```

ShowPropertySheet — method

```
function ShowPropertySheet(
  const SI : TStShellItem) : Boolean;
```

Displays the property sheet for the specified item.

SI is the TStShellItem for which the property sheet should be displayed. The property sheet displayed is the same you see when you right click an item in Windows Explorer and choose Properties from the popup menu.

ShowPropertySheet returns True if the property sheet was displayed, or False if an error occurred.

SmallFolderImages — read-only, run-time property

```
property SmallFolderImages : TImageList
```

The image list that contains the small icons for shell items.

SmallFolderImages is a list of small icons for shell items. The image list represented by SmallFolderImages is hooked to the system image list. Initially the list only contains a handful of icons. As additional icons are requested (by calling the GetFileInfo method) those icons will be added to the image list represented by SmallFolderImages.

See also: GetFileInfo, LargeFolderImages

TStShellEnumerator Component

The TStShellEnumerator component can be used to iterate a folder in the shell namespace. TStShellEnumerator is derived from TStCustomController. To use this component, first set the RootFolder, RootPidl, or SpecialRootFolder property to determine the root folder for the enumeration. Next call the Execute method. When Execute returns, the ShellItems property will contain a list of items in the folder.

Hierarchy

TComponent (VCL)

❶ TStCustomShellController (StShlCtl) 94

TStShellEnumerator (StShlCtl)

Properties

Folder	❶ SmallFolderImages	SpecialRootFolder
❶ DesktopFolder	RootFolder	SortDirection
❶ LargeFolderImages	RootPidl	Sorted
❶ ListView	ShellItems	
Options	ShellVersion	

Methods

❶ Create	❶ GetDisplayName
Execute	❶ GetFileInfo

Events

OnEnumItem

Execute method

```
procedure Execute;
```

Enumerates the folder.

Call Execute to enumerate the folder specified by the RootFolder, RootPidl, or SpecialRootFolder property. When Execute returns, the ShellItems property will contain a list of items in the folder.

See also: RootFolder, RootPidl, ShellItems, SpecialRootFolder

Folder read-only, run-time property

```
property Folder : TStShellFolder
```

The TStShellFolder item that represents the root folder for the enumeration.

See also: TStShellFolder

OnEnumItem event

```
property OnEnumItem : TStEnumItemEvent

TStEnumItemEvent = procedure(Sender : TObject;
  ShellItem : TStShellItem; var Accept : Boolean) of object;
```

Defines an event that is fired for each item in the root folder.

The event handler assigned to OnEnumItem will be called for each item in the root folder as the folder is being enumerated. Sender is the component that generated the event. ShellItem is a pointer to the TStShellItem object representing the shell item. Accept determines whether the item is added to the component's ShellItems list. You can use OnEnumItem to filter the list of items based on any characteristics you choose. The following example shows how to filter the list so that it contains only folders and items with an extension of .TXT:

```
  procedure TForm1.StShellEnumerator1EnumItem(
    Sender: TObject; ShellItem: TStShellItem; var Accept: Boolean);
  begin
    if ShellItem.IsFileSystem and
       not ShellItem.IsFileFolder then
      Accept := (
        UpperCase(ExtractFileExt(ShellItem.Path)) = '.TXT');
  end;
```

Options property

```
property Options : TStEnumeratorOptionsSet

TStEnumeratorOptions = (
  eoIncludeFolders, eoIncludeHidden, eoIncludeNonFolders);

TStEnumeratorOptionsSet = set of TStEnumeratorOptions;
```

Default: [eoIncludeFolders, eoIncludeNonFolders]

Determines the types of items that are to be enumerated.

Set Options to specify the items that will be included in the enumeration. The following table lists the options and gives a description of each:

Option	**Description**
eoIncludeFolders	Folder items will be included in the enumeration.
eoIncludeHidden	Hidden folders and items will be included in the enumeration.
eoIncludeNonFolders	Non-folder items will be included in the enumeration.

RootFolder property

```
property RootFolder : string
```

Default: Empty string

The file folder that will be used as the base folder for the enumeration.

The file folder specified by RootFolder will be the base folder for the enumeration. If the specified folder is invalid, an EStInvalidFolder exception is raised.

The base folder for the enumeration can be specified by the RootFolder, RootPidl, or the SpecialRootFolder property. Use RootFolder to specify a file folder as the base folder. Use RootPidl if you already have a pidl that you want to use as the base folder. Use SpecialRootFolder to set the base folder to system folders such as the Desktop or Network Neighborhood.

See also: RootPidl, SpecialRootFolder

RootPidl property

```
property RootPidl : PItemIDList
```

Default: nil

↳ The pidl that will be used as the base folder for the enumeration.

Use RootPidl when you want to enumerate a shell folder for which you already have a pidl.

The base folder for the enumeration can be specified by the RootFolder, RootPidl, or the SpecialRootFolder property. Use RootFolder to specify a file folder as the base folder. Use RootPidl if you already have a pidl that you want to use as the base folder. Use SpecialRootFolder to set the base folder to system folders such as the Desktop or Network Neighborhood.

See also: RootFolder, SpecialRootFolder

ShellItems read-only property

```
property ShellItems : TStShellItemList
```

↳ The list of items returned as a result of the enumeration.

ShellItems is a list of TStShellItem pointers, each of which represents an item in the target folder. After calling the Execute method, the ShellItems property can be used to iterate all items in the target folder, or to reference a particular shell item by index.

The following example iterates the items returned after enumerating, and adds the DisplayName of each item to a memo. By default, the list of items is sorted.

```
var
  I : Integer;
...
  ShellEnumerator.Execute;
  for I := 0 to ShellEnumerator.ShellItems.Count - 1 do
    Memo1.Lines.Add(
      ShellEnumerator.ShellItems[I].DisplayName);
```

See the TStShellItem topic for details on the information available from TStShellItem.

See also: Execute, Sorted, SortDirection, TStShellItem

ShellVersion — read-only, run-time property

```
property ShellVersion : Double
```

↳ The version number of Windows' SHELL32.DLL.

Read ShellVersion to determine the version of the Windows shell running on a particular machine. Some shell operations and special folders are only available on version 4.72 of the shell and later.

SortDirection — property

```
property SortDirection : TStSortDirection
```

```
TStSortDirection = (sdAscending, sdDescending);
```

Default: sdAscending

↳ Determines the sort direction when the list of shell items is sorted.

When the Sorted property is True, the list of items in the ShellItems property is sorted according to the value of SortDirection. See Sorted for a description of how Windows sorts shell items.

See also: ShellItems, Sorted

Sorted — property

```
property Sorted : Boolean
```

Default: True

↳ Determines whether the list of shell items is sorted.

When Sorted is True, the list of shell items in the ShellItems property is sorted. The list is sorted according to Windows' sorting rules for sorting of shell items. In the case of file system folders, for example, an ascending sort will result in any folders in the target folder being placed first in the list, followed by any files in the target folder. Both the files and the folders are sorted in ascending order. If the list is sorted in descending order, files will be come first in the list in descending order, followed by any folders in the target folder. The SortDirection property dictates the sort direction.

See also: ShellItems, SortDirection

```
property SpecialRootFolder : TStSpecialRootFolder
```

```
TStSpecialRootFolder = (sfAltStartup, sfAppData, sfBitBucket,
  sfCommonAltStartup, sfCommonDesktopDir, sfCommonFavorites,
  sfCommonPrograms, sfCommonStartMenu, sfCommonStartup,
  sfControls, sfCookies, sfDesktop, sfDesktopDir, sfDrives,
  sfFavorites, sfFonts, sfHistory, sfInternet, sfInternetCache,
  sfNetHood, sfNetwork, sfNone, sfPersonal, sfPrinters,
  sfPrintHood, sfPrograms, sfRecentFiles, sfSendTo,
  sfStartMenu, sfStartup, sfTemplates);
```

Default: sfNone

Specifies the shell folder that will be used as the base folder for the enumeration.

The system folder specified by SpecialRootFolder will be the base folder for the enumeration. The base folder for the enumeration can be specified by the RootFolder, RootPidl, or the SpecialRootFolder property. Use RootFolder to specify a file folder as the base folder. Use RootPidl if you already have a pidl that you want to use as the base folder. Use SpecialRootFolder to set the base folder to system folders such as the Desktop or Network Neighborhood. The list of special root folders is a list of system folders defined by Windows. Not every folder in the list is available on all versions of the shell. If a particular folder is invalid, an EStShellError exception is raised.

See also: RootFolder, RootPidl

TStShellNotification Component

The TStShellNotification component monitors changes to the shell and notifies your application of these changes through VCL events. Shell changes include files and folders being modified, created, renamed, or deleted, drives being added or removed, media being inserted or removed from removable media drives, and icons for shell items changing. VCL events are provided for each of these shell changes, as well as a global shell change event for any shell change that occurs.

Shell events can be monitored by folder or the entire shell namespace can be monitored. You can specify the folder to monitor via the WatchFolder or SpecialWatchFolder properties. You can also monitor all subfolders of the specified folder by setting the WatchSubFolders property to True. Some events are global events will occur regardless of the value of the watched folder. This applies to events such as OnAssociationChange, OnDriveAdd, OnDriveRemove, OnImageListChange, OnMediaInsert, OnMediaRemove, OnNetShare, OnNetUnShare, and OnServerDisconnect.

TStShellNotification uses several undocumented Windows functions. Possibly for this reason, some events that you expect would occur based on a change to the shell may not occur at all, or events may occur that you did not anticipate. For example, changing the attributes of a file will generally result in an OnFileChange event being fired rather than an OnAttributeChange event. (Changing attributes on Windows 95 may not generate a shell event of any kind.) Another example is the case of sharing a drive or folder. On Windows NT, an OnNetShare event is generated both when an object is shared and when it is unshared. The OnNetUnShare event never occurs on NT.

Finally, understand that you may receive notification for a particular event more than once. For example, adding a network drive will usually result in two OnDriveAdd events being generated on some operating systems.

Hierarchy

TComponent (VCL)

❶ TStCustomShellController (StShlCtl) 94

TStShellNotification (StShlCtl)

Properties

Active	NotifyEvents	WatchFolder
❶ DesktopFolder	ShellVersion	WatchSubFolders
❶ LargeFolderImages	❶ SmallFolderImages	WatchPidl
MaxNotifications	SpecialWatchFolder	

Methods

❶ Create	❶ GetDisplayName	❶ GetFileInfo

Events

OnAssociationChange	OnFileDelete	OnMediaInsert
OnAttributeChange	OnFileRename	OnMediaRemove
OnDriveAdd	OnFolderChange	OnNetShare
OnDriveRemove	OnFolderCreate	OnNetUnShare
OnDriveSpaceChange	OnFolderDelete	OnServerDisconnect
OnFileChange	OnFolderRename	OnShellChangeNotify
OnFileCreate	OnImageListChange	OnShellDriveAdd

Reference Section

Active — property

```
property Active : Boolean
```

↳ Determines whether notifications will be received.

Set Active to True to begin receiving shell change notifications, and False to disable notifications.

MaxNotifications — property

```
property MaxNotifications : Integer
```

Default: 0

↳ The maximum number of shell event notifications that will be processed at one time.

Shell operations result in any number of shell notifications being sent. When a folder is created, for example, the shell sends notification of this fact. TStShellNotification detects this event and generates the appropriate VCL event. For events such as creating files or folders only one or two such notification events are sent. For some operations—deleting a large number of folders, for example—hundreds of notifications might be sent. This could result in application performance problems as TStShellNotification processes each event. You can adjust MaxNotifications from its default of 0 (unlimited notifications) if you encounter performance problems.

NotifyEvents property

```
property NotifyEvents : TStNotifyEventsSet
```

```
TStNotifyEvents = (neAssociationChange, neAttributesChange,
  neFileChange, neFileCreate, neFileDelete, neFileRename,
  neDriveAdd, neDriveRemove, neShellDriveAdd,
  neDriveSpaceChange, neMediaInsert, neMediaRemove,
  neFolderCreate, neFolderDelete, neFolderRename,
  neFolderUpdate, neNetShare, neNetUnShare,
  neServerDisconnect, neImageListChange);
```

```
TStNotifyEventsSet = set of TStNotifyEvents;
```

Default: All events

Determines the shell events that will be monitored.

If you are only interested in receiving notification on specific events, only set those events in the NotifyEvents property. If, for example, you are only interested in knowing when media is inserted or removed from removable media devices, set NotifyEvents to neMediaInsert and neMediaRemove.

OnAssociationChange event

```
property FOnAssociationChange : TNotifyEvent
```

Defines an event handler that is called when a file association changes.

The OnAssociationChange event is fired when a file association changes. No meaningful information is available from the shell regarding file association changes.

OnAttributeChange event

```
property OnAttributeChange : TStShellNotifyEvent2
```

```
TStShellNotifyEvent2 = procedure(
  Sender : TObject; OldShellItem : TStShellItem;
  NewShellItem : TStShellItem) of object;
```

Defines an event handler that is called when a file's attributes change.

In theory, the OnAttributeChange event handler is fired when the attributes of a file in the watched folder change. In most cases, however, an OnFileChange event occurs instead. Under Windows 95, no event is generated when the file association changes.

See also: OnFileChange

OnDriveAdd — event

```
property OnDriveAdd : TStShellNotifyEvent1
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

↳ Defines an event handler that is called when a drive is added to the system.

The OnDriveAdd event is fired when a drive is added to the system. This event usually occurs when a network drive is mapped. ShellItem is a pointer to the TStShellItem object associated with the drive that is added.

See also: OnDriveRemove

OnDriveRemove — event

```
property OnDriveRemove : TStShellNotifyEvent1
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

↳ Defines an event handler that is called when a drive is removed from the system.

The OnDriveRemove event is fired when a drive is removed from the system. This event usually occurs when a network drive is disconnected. ShellItem is a pointer to the TStShellItem object associated with the drive that is removed.

See also: OnDriveAdd, TStShellItem

OnDriveSpaceChange — event

```
property OnDriveSpaceChange : TStShellNotifyEvent4
```

```
TStShellNotifyEvent4 =
  procedure(Sender : TObject; Drive : DWORD) of object;
```

↳ Defines an event handler that is called when the amount of free space on a drive changes.

The OnDriveSpaceChange event is fired when the amount of free space on a drive changes. Drives is a bit set that indicates which drives have changed. Bits 0 through 25 represent drives A: through Z:. If, for example, the free space on drive C: changes, Drive will be equal to 4. If the free space on both the C: and D: drives changed, Drive will be equal to 12. An OnFolderChange event also occurs when the drive space changes.

See also: OnFolderChange

OnFileChange event

```
property OnFileChange : TStShellNotifyEvent1
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

Defines an event handler that is called when a file in the watched folder changes.

The OnFileChange event handler is fired when a file in the watched folder changes. ShellItem is a pointer to the TStShellItem object that represents the file being changed. This event is usually generated when the attributes of a file change. Contrary to what you might expect, it is not typically generated when a file is modified and saved by an external program.

See also: OnAttributeChange, OnFileCreate, OnFileDelete, OnFileRename

OnFileCreate event

```
property OnFileCreate : TStShellNotifyEvent1
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

Defines an event handler that is called when a file in the watched folder is created.

The OnFileCreate event handler is fired when the shell creates a file in the watched folder. ShellItem is a pointer to the TStShellItem object that represents the file being created. The OnFileCreate event will not be fired when an external program creates a file in the watched folder. This event will, however, be generated when a file is added to the folder by the shell (such as the result of a paste operation or a file move).

See also: OnFileChange, OnFileDelete, OnFileRename

```
property OnFileDelete : TStShellNotifyEvent2

TStShellNotifyEvent2 = procedure(
  Sender : TObject; OldShellItem : TStShellItem;
  NewShellItem : TStShellItem) of object;
```

Defines an event handler that is called when a file in the watched folder is deleted.

The OnFileDelete event handler is fired when the shell deletes a file in the watched folder. OldShellItem is a pointer to the TStShellItem object that represents the file before it is deleted. NewShellItem is a pointer to the TStShellItem object that represents the file after it is deleted.

Files that are sent to the Recycle Bin may not result in an OnFileDelete event being generated on all operating systems. In some cases an OnRenameFile event is generated instead. This is because an item sent to the Recycle Bin is renamed but not actually deleted. When a file is deleted on Windows NT and Windows 98, an OnFileDelete event is generated. On Windows 95, an OnFileRename event occurs instead. Shell version may be a factor as well as operating system. The point to remember is that you should be prepared to respond to both the OnFileDelete and OnFileRename event if you want to be notified of file deletions.

Consider the case where a file called Test.txt is deleted to the Recycle Bin. In that case the DisplayName property of OldShellItem will be “Test.txt” and the DisplayName property of NewShellItem will be a new file name generated by the shell (“dc10.txt”, for example). If the file is being deleted rather than sent to the Recycle Bin, NewShellItem will be nil. Be sure to check NewShellItem for nil before attempting to use it.

The OnFileDelete event is not generated when a file is programmatically deleted using the DeleteFile function. The event will only be generated when a file is deleted using the shell (such as when you delete a file using the TStFileOperation component or through Windows Explorer).

See also: OnFileChange, OnFileCreate, OnFileRename, TStFileOperation

OnFileRename event

```
property OnFileRename : TStShellNotifyEvent2
```

```
TStShellNotifyEvent2 = procedure(
  Sender : TObject; OldShellItem : TStShellItem;
  NewShellItem : TStShellItem) of object;
```

↳ Defines an event handler that is called when a file in the watched folder is renamed.

The OnFileRename event handler is fired when the shell renames a file in the watched folder. OldShellItem is a pointer to the TStShellItem object that represents the file before it is renamed. NewShellItem is a pointer to the TStShellItem object that represents the file after it is renamed. The event will be generated when a file is renamed using the shell (when a file is renamed using the TStFileOperation component, for example).

Consider the case where a file called Test.txt is renamed to MyFile.txt. In that case the DisplayName property of OldShellItem will be “Test.txt” and the DisplayName property of NewShellItem will be “MyFile.txt.”

See also: OnFileChange, OnFileCreate, OnFileDelete, TStFileOperation

OnFolderChange event

```
property OnFolderChange : TStShellNotifyEvent1
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

↳ Defines an event handler that is called when the watched folder changes.

The OnFolderChange event handler is fired when the watched folder changes. ShellItem is a pointer to the TStShellItem object that represents the folder being changed. This event is generated when the folder’s drive space changes (as the result of a file or folder being created, deleted, or modified) or when the folder’s attributes change.

See also: OnDriveSpaceChange, OnFolderCreate, OnFolderDelete, OnFolderRename

OnFolderCreate event

```
property OnFolderCreate : TStShellNotifyEvent1
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

Defines an event handler that is called when a new folder in the watched folder is created.

The OnFolderCreate event handler is fired when the shell creates a folder in the watched folder. ShellItem is a pointer to the TStShellItem object that represents the folder being created. The OnFolderCreate event is not fired when an external program creates a folder. This event is, however, generated when a folder is created by the shell (such as the result of a paste operation or a file move).

See also: OnFolderChange, OnFolderDelete, OnFolderRename

OnFolderDelete event

```
property OnFolderDelete : TStShellNotifyEvent2
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

Defines an event handler that is called when a folder in the watched folder is deleted.

The OnFolderDelete event handler is fired when the shell deletes a folder in the watched folder. ShellItem is a pointer to the TStShellItem object that represents the folder that is being deleted. Unlike files sent to the Recycle Bin, folders that are deleted to the Recycle Bin result in an OnFolderDelete event rather than an OnFolderRename event on all operating systems.

The OnFolderDelete event is not generated when a file is programmatically deleted unless the folder is deleted using the shell (such as when a folder is deleted using the TStFileOperation component or through Windows Explorer).

See also: OnFolderChange, OnFolderCreate, OnFolderRename, TStFileOperation

OnFolderRename event

```
property OnFolderRename : TStShellNotifyEvent2
```

```
TStShellNotifyEvent2 = procedure(
  Sender : TObject; OldShellItem : TStShellItem;
  NewShellItem : TStShellItem) of object;
```

Defines an event handler that is called when a folder in the watched folder is renamed.

The OnFolderRename event handler is fired when the shell renames a folder. OldShellItem is a pointer to the TStShellItem object that represents the folder before it is renamed, and NewShellItem is a pointer to the TStShellItem object that represents the folder after it is renamed. The event is generated when a file is renamed using the shell (this is the case when a file is renamed using the TStFileOperation component, for example).

Consider the case where a folder called New Folder is renamed to MyFiles. In that case, the DisplayName property of OldShellItem is “New Folder” and the DisplayName property of NewShellItem is “MyFiles.”

See also: OnFolderChange, OnFolderCreate, OnFolderDelete, TStFileOperation

OnImageListChange event

```
property OnImageListChange : TNotifyEvent
```

Defines an event handler that is called when an image in the system image list changes.

The OnImageListChange event is fired when an image in the system image list changes. Usually this event occurs as the result of a CD being inserted into a CD-ROM drive. No meaningful information is available from the shell regarding file association changes.

OnMediaInsert event

```
property OnMediaInsert : TStShellNotifyEvent1
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

Defines an event handler that is called when media is inserted into a removable media device.

ShellItem is a pointer to the TStShellItem object that represents the drive into which the media was inserted. If, for example, media was inserted into a CD-ROM on the D: drive, the Path property of ShellItem is “D:\”. The OnMediaInsert event is not generated when a diskette is inserted into a floppy drive. This event is usually accompanied by an OnImageListChange event.

See also: OnMediaRemove

OnMediaRemove event

```
property OnMediaRemove : TStShellNotifyEvent1
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

↳ Defines an event handler that is called when media is removed from a removable media device.

ShellItem is a pointer to the TStShellItem object that represents the drive from which the media was removed. If, for example, a CD was removed from a CD-ROM on the D: drive, the Path property of ShellItem is “D:\”. The OnMediaRemove event is not generated when a diskette is removed from a floppy drive. This event is usually accompanied by an OnImageListChange event.

See also: OnMediaInsert

OnNetShare event

```
property OnNetShare : TStShellNotifyEvent1
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

↳ Defines an event handler that is called when a folder or drive is shared.

ShellItem is a pointer to the TStShellItem object that represents the folder that is being shared. If, for example, the E: drive is being shared, the Path property of ShellItem is “E:\”. On Windows NT the OnNetShare event is generated when a folder is shared and when a folder is unshared.

See also: OnNetUnShare

OnNetUnShare event

```
property OnNetUnShare : TStShellNotifyEvent1
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

↳ Defines an event handler that is called when a folder or drive is unshared.

ShellItem is a pointer to the TStShellItem object that represents the folder for which the share is being removed. If, for example, the E: drive is being unshared, the Path property of ShellItem is “E:\”. On Windows NT the OnNetUnShare event is never generated. Instead, the OnNetShare event is generated when the folder is unshared.

See also: OnNetUnShare

OnServerDisconnect event

```
property OnServerDisconnect : TStShellNotifyEvent1
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

↳ Defines an event handler that is called when a remote machine disconnects from a network.

ShellItem is a pointer to the TStShellItem object that represents the drive that is being disconnected.

See also: OnDriveAdd, OnDriveRemove, OnShellDriveAdd

OnShellChangeNotify event

```
property OnShellChangeNotify : TStShellNotifyEvent3
```

```
TStShellNotifyEvent3 = procedure(Sender : TObject;
  OldShellItem : TStShellItem; NewShellItem : TStShellItem;
  Events : TStNotifyEventsSet) of object;
```

↳ Defines an event handler that is called when any shell event occurs.

The OnShellChangeNotify event is fired any time a shell event occurs. OldShellItem is a pointer to the TStShellItem object that represents the original shell item. NewShellItem is a pointer to the TStShellItem object that represents the new shell item. In some cases, NewShellItem may be nil so be sure to check this parameter for nil before attempting to use it. Events is a set that contains the shell events that were generated. In some cases several events may be sent at one time so be sure to check the Events property to determine which events occurred.

OnShellDriveAdd event

```
property OnShellDriveAdd : TStShellNotifyEvent1
```

```
TStShellNotifyEvent1 = procedure(
  Sender : TObject; ShellItem : TStShellItem) of object;
```

↳ Defines an event handler that is called when a drive is added by the shell GUI.

ShellItem is a pointer to the TStShellItem object that represents the drive that is added.

See also: OnDriveAdd, OnDriveRemove

ShellVersion — read-only, run-time property

```
property ShellVersion : Double
```

↳ The version number of Windows' SHELL32.DLL.

Read ShellVersion to determine the version of the Windows shell running on a particular machine.

SpecialWatchFolder — property

```
property SpecialWatchFolder : TStSpecialRootFolder
```

```
TStSpecialRootFolder = (sfAltStartup, sfAppData, sfBitBucket,
  sfCommonAltStartup, sfCommonDesktopDir, sfCommonFavorites,
  sfCommonPrograms, sfCommonStartMenu, sfCommonStartup,
  sfControls, sfCookies, sfDesktop, sfDesktopDir, sfDrives,
  sfFavorites, sfFonts, sfHistory, sfInternet, sfInternetCache,
  sfNetHood, sfNetwork, sfNone, sfPersonal, sfPrinters,
  sfPrintHood, sfPrograms, sfRecentFiles, sfSendTo,
  sfStartMenu, sfStartup, sfTemplates);
```

Default: sfNone

↳ Specifies the shell folder that is monitored.

The system folder specified by SpecialWatchFolder is the folder monitored for shell events. The watch folder can be specified by either the WatchFolder property, the SpecialWatchFolder property, or the WatchPidl property. Use WatchFolder to specify a file folder, and SpecialWatchFolder to specify a system folder (such as the Desktop or Network Neighborhood). Use WatchPidl to specify any folder for which you already have a PIDL. The list of special watch folders is a list of system folders defined by Windows. Not every folder in the list is available on all versions of the shell. If a particular folder is invalid, an EStShellError exception is raised.

See also: WatchFolder, WatchPidl

WatchFolder property

```
property WatchFolder : string
```

Default: Empty string

Specifies the file system folder that is monitored.

The system folder specified by WatchFolder is the folder monitored for shell events. The watch folder can be specified by either the WatchFolder property, the WatchPidl property, or the SpecialWatchFolder property. Use WatchFolder to specify a file folder, and SpecialWatchFolder to specify a system folder (such as the Desktop or Network Neighborhood). If the specified folder is invalid, an EStShellError exception is raised.

See also: SpecialWatchFolder, WatchPidl, WatchSubFolders

WatchPidl property

```
property WatchPidl : PItemIDList
```

Specifies the folder that is monitored, using a pidl.

The system folder specified by SpecialWatchFolder is the folder monitored for shell events. The watch folder can be specified by either the WatchFolder property, the SpecialWatchFolder property, or the WatchPidl property. Use WatchFolder to specify a file folder, and SpecialWatchFolder to specify a system folder (such as the Desktop or Network Neighborhood). Use WatchPidl to specify any folder for which you already have a PIDL. The list of special watch folders is a list of system folders defined by Windows. Not every folder in the list is available on all versions of the shell. If a particular folder is invalid, an EStShellError exception is raised.

The system folder specified by WatchFolder is the folder monitored for shell events. The watch folder can be specified by either the WatchFolder property, the WatchPidl property, or the SpecialWatchFolder property. Use WatchFolder to specify a file folder, and SpecialWatchFolder to specify a system folder (such as the Desktop or Network Neighborhood). If the specified folder is invalid, an EStShellError exception is raised.

See also: SpecialWatchFolder, WatchFolder, WatchSubFolders

WatchSubFolders property

```
property WatchSubFolders : string
```

Default: False

↳ Determines whether subfolders are monitored in addition to the watch folder.

When WatchSubFolders is True, shell events will be generated when a shell event occurs in the watched folder and any subfolders within the watched folder.

See also: SpecialWatchFolder, WatchFolder

TStShellAbout Component

The TStShellAbout component shows the standard Windows shell About dialog. This is the dialog you see when you choose Help |About from Windows Explorer. The dialog shows the operating system name, the operating system version, and system information such as the amount of memory available to Windows. The TStShellAbout component allows you to specify an icon for the dialog box, the dialog box's title, and additional text.

Hierarchy

TComponent (VCL)

  TSsComponent (StBase)

    TSsShellComponent (StShBase)

      TStCustomShellAbout (StAbout)

        TStShellAbout (StAbout)

Properties

AdditionalText	Icon	Version
Caption	TitleText	

Methods

Execute

Reference Section

AdditionalText — property

```
property AdditionalText : string
```

Specifies any additional information you want displayed on the dialog box.

AdditionalText is the text that appears on the dialog box below the Microsoft copyright notice. Windows leaves space for two lines of text on the dialog box so AdditionalText should be limited to 80-90 characters. If AdditionalText is blank, this area of the dialog box is blank.

See also: Caption, TitleText

Caption — property

```
property Caption : string
```

Specifies any additional text you want displayed in the title bar of the dialog box.

Caption is the text that appears in the title bar of the dialog box. Windows will automatically prepend the word "About" to Caption and display that text in the title bar. For example, if you set Caption to "Memory Sleuth", the text, "About Memory Sleuth" will appear in the dialog box's title bar. If Caption is an empty string, the word "About" alone will appear in the title bar. The text in Caption will also appear on the first line of text in the dialog box if the TitleText property is blank.

See also: AdditionalText, TitleText

Execute — method

```
function Execute : Boolean;
```

Displays the shell about dialog box.

Execute returns True if the dialog is shown, and False if Windows is unable to display the dialog box. In practice, there should be little or no reason for Execute to return False unless the operating system is already in a precarious state.

Icon — property

```
property Icon : TIcon
```

Specifies the icon to be displayed in the upper-left corner of the dialog box.

If Icon is nil the default Windows icon is used.

TitleText — property

```
property TitleText : string
```

Specifies the text displayed on the first line of the dialog box.

TitleText would typically be set to your application’s name. TitleText will appear on the first line of the dialog box after the text “Microsoft (R).” If TitleText is an empty string, the text in Caption will be used in place of TitleText. If neither TitleText or Caption are specified, the first line of the dialog box contains the text “Microsoft (R).”

See also: Caption

Version — read-only property

```
property Version : string
```

The current version of ShellShock.

Version is provided so you can identify your ShellShock version if you need technical support. The ShellShock about box can be displayed by double-clicking this property or selecting the dialog button to the right of the property value.

TStBrowser Component

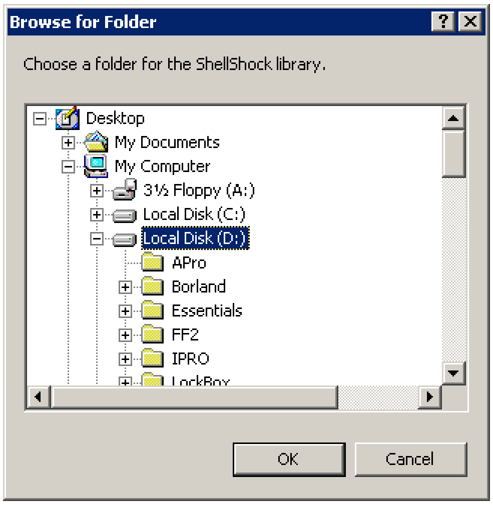
4

The TStBrowser component is a wrapper around the Windows shell browser dialog. This dialog allows you to browse the shell for a folder, a printer, or a network computer. The most common use for TStBrowser is likely to be browsing for folders. When used in this way, TStBrowser becomes a “directory picker” dialog box. In some versions of the Windows shell you can also browse for files with TStBrowser. Figure 4.1 shows the Windows Browser for Folder dialog box displayed by TStBrowser.

Since TStBrowser is tied to the Windows shell, its capabilities vary with the version of the Windows shell you have installed. The shell version varies depending on the operating system you are using and with the version number of Internet Explorer you have installed. The shell version is determined by the version of SHELL32.DLL. The later versions of the Windows shell allow you to browse for files as well as for folders. They also give you the option of displaying an edit control at the top of the dialog. The edit control allows the user to enter a folder, printer, or computer name rather than using the mouse to browse for the object. To check the version of the shell you can read the ShellVersion property.

TStBrowser has a rich set of options that enable you to stipulate how the browser operates. These options are controlled through the Options property. See the Options property in the Reference section for more information.

To incorporate the browser’s functionality without the standard Windows dialog box, see the ShellShock TStShellTreeView (page 34) and TStShellListView (page 48) components.


Browse for Folder

Choose a folder for the ShellShock library.

Figure 4.1: TStBrowser displays the Windows Browse for Folder dialog box.

Hierarchy

TComponent (VCL)

  TSsComponent (StBase)

    TSsShellComponent (StShBase)

      TStCustomBrowser (StBrowsr)

        TStBrowser (StBrowsr)

Properties

AdditionalText	OKEnabled	ShellVersion
Caption	Options	SpecialRootFolder
DisplayName	Path	SpecialRootFolderID
Handle	Position	StatusText
IDList	RootFolder	Version
ImageIndex	SelectedFolder	

Methods

Execute

Events

OnShow	OnSelChanged

Reference Section

AdditionalText — property

```
property AdditionalText : string
```

↳ Defines the text that appears on the browser dialog box.

AdditionalText appears just below the title bar and just above the text specified in the StatusText property.

Windows only allocates space for two lines of text for this field, so AdditionalText should be limited to 80-90 characters.

If AdditionalText is blank, this area of the dialog box is blank.

See also: Caption, StatusText

Caption — property

```
property Caption : string
```

↳ Specifies the text that appears in the title bar of the dialog box.

If Caption is an empty string, Windows automatically supplies text based on the type of object being browsed. For example, when browsing for a folder the dialog box title is “Browse for Folder,” when browsing for a printer the dialog title is “Browse for Printer,” and so on.

Set Caption if you want a custom title for the browser dialog box.

See also: AdditionalText

DisplayName — read-only, run-time property

```
property DisplayName : string
```

↳ Shows the friendly name of the selected object.

Read DisplayName to get the friendly name of the selected object. For example, let’s say you were browsing for printers and selected the printer “HP LaserJet” on the network server, “PrintServer.” In that case DisplayName will contain “HP LaserJet.”

Read the Path property to get the actual UNC path for the selected object.

See also: Path

Execute — method

```
function Execute : Boolean;
```

↳ Displays the browser dialog box.

Call Execute to display the browser dialog box. Execute returns True if the dialog box was closed with the OK button, and False if the dialog box was closed with the Cancel button or the close box. Read the Path property to determine the path to the selected object.

See also: DisplayName, Path

Handle — read-only, run-time property

```
property Handle : Integer
```

↳ The window handle of the browser dialog box.

Read Handle if you need the Window handle of the browser dialog box for Windows API calls. Handle is only valid in the OnShow or OnSelChanged events. Handle is 0 after the dialog box closes.

See also: OnSelChanged, OnShow

IDList — read-only, run-time property

```
property IDList : PItemIDList
```

↳ Points to the item ID list for the selected object.

Read IDList if you want to call Windows API functions that require a pointer to an item ID list. Use of IDList is for advanced users. Most users can ignore this property.

See also: Path

ImageIndex — read-only, run-time property

```
property ImageIndex : Integer
```

↳ Shows the index of the Windows image associated with a selected item.

Read ImageIndex to get the image index of the icon associated with the selected item. For example, the image index for a printer is 10, the image index for a folder is 13, the image index for a network computer is 9, and so on.

No official list of images and their corresponding numbers exists, so some degree of experimentation will be required to determine image mappings.

OKEnabled run-time property

```
property OKEnabled : Boolean
```

Default: True

↳ Determines whether the OK button on the browser dialog box is enabled.

Set OKEnabled to False to disable the OK button while the dialog box is active. You can only set OKEnabled in an OnSelChanged event handler.

The following example prevents the user from selecting the root directory on the D drive by disabling the OK button if D:\ is selected:

```
procedure TForm1.StBrowser1SelChanged(Sender : TObject);
begin
  if StBrowser1.SelectedFolder = 'D:\' then
  StBrowser1.OKEnabled := False
  else
  StBrowser1.OKEnabled := True;
end;
```

OKEnabled is automatically forced to True each time the Execute method is called. In most cases, you don’t have to worry about enabling or disabling the OK button on the browse dialog box. The OK button is automatically enabled and disabled based on the Options property and based on the current selection in the dialog box.

See also: Execute, OnSelChanged, Options

OnSelChanged event

```
property OnSelChanged : TNotifyEvent
```

↳ Defines an event handler that is called each time the selection changes.

Use OnSelChanged to determine the currently selected object in the browse dialog box. The SelectedFolder property will give you the friendly name of the selected folder, and the IDList property will give you a pointer to the item ID list of the selected object.

The following example displays the selected folder in a label on the main form of an application:

```
procedure TForm1.StBrowser1SelChanged(Sender : TObject);
begin
  Label1.Caption := StBrowser1.SelectedFolder;
end;
```

See also: IDList, SelectedFolder

OnShow — event

```
property OnShow : TNotifyEvent
```

Defines an event handler that is called just before the dialog box is visible.

Use OnShow to perform any initialization required prior to the browse dialog box being shown.

The following example uses OnShow to position the dialog in the upper-left corner of the desktop:

```
  procedure TForm1.StBrowser1Show(Sender : TObject);
  begin
    SetWindowPos(
      StBrowser1.Handle, 0, 0, 0, 0, 0, SWP_NOSIZE or WP_NOZORDER);
  end;
```

See also: Handle

Options — property

```
property Options : TStBrowseOptionsSet
```

```
TStBrowseOptionsSet = set of TStBrowseOptions;
```

```
TStBrowseOptions = (
  boBrowseForComputer, boBrowseForPrinter, boDontGoBelowDomain,
  boReturnOnlyAncestors, boReturnOnlyDirs, boShowFiles, boEditBox);
```

Default: boReturnOnlyDirs

Determines how the browser dialog box operates.

Set the browse options to control how the browse dialog box should behave. The following table describes the possible values of the Options property:

Value	**Description**
boBrowseForComputer	The browse dialog box will only enable the OK button when a computer name is selected.
boBrowseForPrinter	The browse dialog box will only enable the OK button when a printer is selected.
boDontGoBelowDomain	The browse dialog box will not show any computers below the domain level.
boReturnOnlyAncestors	The browse dialog box will only enable the OK button when a file system ancestor is selected (a computer name as opposed to a shared drive under a computer).

`boReturnOnlyDirs`	`The browse dialog box enables the OK button only if a file system directory is selected.`
`boShowFiles`	`Shows files in directories in addition to directories. Shell version 4.71 or later only.`
`boEditBox`	`Places an edit box above the browse window in the browse dialog. Shell version 4.71 or later only.`

Be aware that combinations of these values can produce results that may not be what you expect. For example, if Options contains both boBrowseForComputer and boBrowseForPrinter, only computers with printers are displayed.

Note that the boShowFiles and boEditBox elements can only be used with shell versions 4.71 and later. If you attempt to use one of these options with earlier versions of the shell, an EStShellError exception is raised. If your application enables these flags be sure you are prepared to handle this exception when calling the Execute method.

See also: Execute

Path read-only, run-time property

```
property Path : string
```

Contains the UNC path of the selected object.

Read Path to get the actual UNC path of the selected object. For example, if you were browsing for printers and selected the printer “HP LaserJet” on the network server, “PrintServer.” In that case the Path property will contain “\\PrintServer\HP Laser Jet.” Read DisplayName to get the friendly name of the selected object.

See also: DisplayName

Position property

```
property Position : TStBrowsePosition

TStBrowsePosition = (bpDefault, bpScreenCenter);
```

Default: bpScreenCenter

Determines the position of the browse dialog box when it is initially displayed.

If Position is set to bpDefault, the browse dialog box appears in a position as determined by Windows (usually slightly down and to the right of the application’s upper-left corner). If Position is set to bpScreenCenter, the browse dialog box is centered on the screen.

See also: OnShow

RootFolder property

```
property RootFolder : string
```

↳ Determines the browse dialog box's root folder.

Set RootFolder to force the browse dialog box to start with a particular drive or directory. Users cannot navigate above RootDir. If, for example you set RootDir to "D:\" only directories of drive D: will be displayed in the browse dialog box. To force the browse dialog box to start with a particular directory, yet allow users to browse above that directory, use SelectedFolder instead of RootFolder. You can also set the root folder by specifying one of Windows' special folders (see SpecialRootFolder). If SpecialRootFolder contains a value other than sfNone, the RootFolder property is ignored.

See also: SelectedFolder, SpecialRootFolder

SelectedFolder property

```
property SelectedFolder : string
```

↳ Contains the currently selected folder.

Read SelectedFolder from an OnSelChanged event handler to obtain the path to the selected folder while the dialog box is active. SelectedFolder returns the UNC path for directories only. When the user selects a computer or printer, SelectedFolder is an empty string.

To force the browse dialog box to select a particular folder on start-up, set SelectedFolder to the desired start folder before calling Execute. If the directory in SelectedFolder does not exist, the browse dialog box will start with the MyComputer object.

See also: Execute, OnSelChanged

ShellVersion property

```
property ShellVersion : Double
```

↳ Contains the version number of SHELL32.DLL.

Certain TStBrowser operations only function with version 4.70 or later of the Windows shell (SHELL32.DLL). Read ShellVersion to determine the version number of the shell.

```
property SpecialRootFolder : TStSpecialRootFolder
```

```
TStSpecialRootFolder = (sfAltStartup, sfAppData, sfBitBucket,
  sfCommonAltStartup, sfCommonDesktopDir, sfCommonFavorites,
  sfCommonPrograms, sfCommonStartMenu, sfCommonStartup,
  sfControls, sfCookies, sfDesktop, sfDesktopDir, sfDrives,
  sfFavorites, sfFonts, sfHistory, sfInternet, sfInternetCache,
  sfNetHood, sfNetwork, sfNone, sfPersonal, sfPrinters,
  sfPrintHood, sfPrograms, sfRecentFiles, sfSendTo, sfStartMenu,
  sfStartup, sfTemplates);
```

Default: sfNone

Contains the Windows folder which serves as the root folder for the dialog box.

SpecialRootFolder provides an alternate way of specifying the root folder for the browser dialog box (as opposed to using the RootFolder property). Set SpecialRootFolder to one of the elements of the TStSpecialRootFolder enumeration. If SpecialRootFolder contains a value other than sfNone, the RootFolder property is ignored.

Some of the elements of TStSpecialRootFolder are only applicable to version 4.70 of the shell and later. If you specify an invalid root folder for the installed shell, an EStShellError exception is raised. You should be prepared to handle this exception when you set SpecialRootFolder.

The elements of TStSpecialFolder particular to shell version 4.70 and later are:

SfInternet	SfCommonFavorites	SfHistory
SfAltStartup	SfInternetCache	
SfCommonAltStartup	SfCookies	

See also: RootFolder, SpecialRootFolderID

SpecialRootFolderID property

```
property SpecialRootFolder : TStSpecialRootFolder
```

Default: 0

Sets the root folder by ID number.

SpecialRootFolderID is an integer value that can specify the special root folder by index. Normally you will use SpecialRootFolder or RootFolder to set the root folder for the browser dialog box. If, however, you have your own shell extension (or the ID of any other installed shell extension) you can use SpecialRootFolderID to specify that shell extension as the root folder for the browser dialog box.

See also: RootFolder, SpecialRootFolder

StatusText property

```
property StatusText : string
```

A string which appears just above the browse window of the browser dialog box.

StatusText should be no longer than one line of text (about 40 characters). This text appears above the browse window and below the text specified in AdditionalText (if any).

See also: AdditionalText, Caption

Version read-only property

```
property Version : string
```

The current version of ShellShock.

Version is provided so you can identify your ShellShock version if you need technical support. You can display the ShellShock about box by double-clicking this property or selecting the dialog button to the right of the property value.

TStFileOperation Component

The TStFileOperation component provides an interface to shell file operations. The shell file operation allows you to provide animation for your file operations, as shown in Figure 4.2. In addition, the shell file operations give you the benefit of confirmation dialogs. The confirmation dialog boxes are displayed when a file is about to be overwritten or deleted, when a directory needs to be created, and so on. File operations include copy, rename, move, and delete. File operations apply to directories as well as individual files.

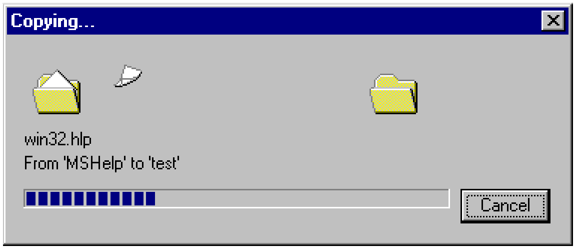


Figure 4.2: Windows provides a status dialog box for some file operations.

The progress dialog box is not displayed in all cases. If you are copying a single file, the file operation happens quickly enough that animation is not required. In that case the animation is not displayed. You can specifically request silent operation, in which case the animation dialog box is not shown at all.

Hierarchy

TComponent (VCL)

  TSsComponent (StBase)

    TSsShellComponent (StShBase)

      TStCustomFileOperation (StFileOp)

        TStFileOperation (StFileOp)

Properties

ConfirmFiles	ErrorString	SimpleText
Destination	Operation	SourceFiles
Error	Options	Version

Methods

Execute

Events

OnError

Reference Section

ConfirmFiles property

```
property ConfirmFiles : Boolean
```

Default: True

↳ Determines whether or not the list of source files is checked for validity before continuing with the file operation.

When ConfirmFiles is True, each file in SourceFiles is checked for existence before executing the file operation. If a file in the SourceFiles list is not valid, an EStFileNotFoundError exception is raised. This check is performed when you call the Execute method. Setting ConfirmFiles to True allows you to catch “file not found” errors before the shell file operations dialog box starts the file operation. ConfirmFiles does not perform checks on paths containing wildcards or on directories.

See also: Execute

Destination property

```
property Destination : string
```

↳ Names the destination directory for the file operation.

Set Destination to the target directory for copied, moved, or renamed files. Destination is ignored for delete operations. Destination must be a directory (as opposed to a file). To copy and rename files use the SourceFiles property. When renaming files using wildcards be sure to specify the directory in Destination. For example to rename all .PAS files in the E:\Test directory to .BAK you should set SourceFiles to “E:\Test\*.PAS” and Destination to “E:\Test\*.BAK.” If Destination does not exist on the target drive, the user will be prompted to create the directory. If the Options property includes foNoConfirmMkDir, the directory will be created without prompting the user.

See also: Execute, Operation, SourceFiles

Error read-only, run-time property

```
property Error : Integer
```

↳ Returns the error code of the last operation.

Check Error in an OnError event handler or after Execute returns. If Execute returns False, you should check the value of Error to determine why the file operation failed. Error could contain any of the Windows file operation error codes. If the user cancelled the file operation, Error will contain ERROR_CANCELLED.

See also: ErrorString, Execute, OnError

ErrorString read-only, run-time property

```
property ErrorString : string
```

↳ Contains a text description of the last error.

ErrorString can be used to display a message to your users in the event an error occurs during the file operation.

See also: Error, OnError

Execute method

```
function Execute : Boolean;
```

↳ Starts the file operation.

Call Execute to perform the file operation specified in the Operation property. The source for the operation (single file, multiple files, or directories) is determined by the SourceFile property. The destination is determined by the Destination property (for a directory) or the SourceFiles property (for individual destination files). Execute returns True if the operation succeeded or False if the operation failed or was cancelled by the user. The OnError event will be generated if an error occurs during the file operation. Check the value of the Error property to determine the error condition which caused the file operation to fail. An EStFileOpError exception is raised if Destination or SourceFiles property is blank, or if the SourceFiles property contains invalid mappings. This exception is not raised if the Destination is blank and the file operation is foDelete since the Destination property is ignored when deleting files. If ConfirmFiles is True and a file in the SourceFiles list is invalid, an EStFileNotFoundError exception is raised.

See also: Destination, Error, ErrorString, OnError, SourceFiles

OnError event

```
property OnError : TNotifyEvent
```

Defines an event handler that is called if an error occurs during the file operation.

Use OnError to determine if an error occurred during the file operation. An error could occur because a file wasn't found, because the destination drive is full, because a file was marked read-only, or for any number of other reasons. The OnError event will also be generated if the user clicks the Cancel button during a file operation. Check the value of the Error property to determine the cause of the error. Read ErrorString for a text description of the error.

The following example shows an OnError event handler which displays an error message to the user:

```
procedure TForm1.StFileOperation1Error(Sender : TObject);
begin
  MessageDlg(StFileOperation1.ErrorString, mtError, [mbOk], 0);
end;
```

See also: Error, ErrorString, Execute

Operation property

```
property Operation : TStFileOp

TStFileOp = (fopCopy, fopDelete, fopMove, fopRename);
```

Default: fopCopy

Defines the file operation to perform.

Operation is the file operation that will be carried out when the Execute method is called.

See also: Destination, Execute, SourceFiles

```
property Options : TStFileOpOptionsSet

TStFileOpOptions  = (
  foAllowUndo, foFilesOnly, foNoConfirmation, foNoConfirmMkDir,
  foNoErrorUI, foRenameCollision, foSilent, foSimpleProgress);

TStFileOpOptionsSet = set of TStFileOpOptions;
```

Default: foAllowUndo, foFilesOnly, foRenameCollision

Determines how the file operation should be carried out.

The following values are defined:

Value	**Meaning**
foAllowUndo	Allow operations to be undone, if applicable. Specifically, files deleted with the foDelete operation will be sent to the recycle bin rather than being permanently deleted.
foFilesOnly	If a wildcard is specified (such as *.*) the operation will only be carried out on files. No directories are affected by the operation.
foNoConfirmation	Carries out the operation without asking for any confirmation from the user. Same as the user clicking “Yes to All” on the confirmation dialog box.
foNoConfirmMkDir	Doesn’t prompt the user for confirmation if a directory needs to be created.
foNoErrorUI	No dialogs will be displayed to the user when an error occurs.
foRenameCollision	Renames the new file if a file with that name already exists in the destination directory.
foSilent	Carries out the operation without showing any animation dialog boxes.
foSimpleProgress	Displays a progress dialog box, but does not display the file names.

The following example creates an instance of TStFileOperation dynamically, sets the Options property to copy a directory silently, and performs the copy:

```
procedure TForm1.Button1Click(Sender : TObject);
var
  FileOp : TStFileOperation;
begin
  FileOp := TStFileOperation.Create(Self);
  with FileOp do begin
    Operation := fopCopy;
    SourceFiles.Clear;
    SourceFiles.Add('c:\code\*.*');
    Destination := 'c:\backup';
    Options := [
      foNoErrorUI, foSilent, foNoConfirmation, foConfirmMkDir];
    Screen.Cursor := crHourGlass;
    try
      Execute;
      if Error <> 0 then
        MessageDlg(ErrorString, mtError, [mbOk], 0);
    finally
      Screen.Cursor := crDefault;
      Free;
    end;
  end
end;
```

See also: Execute, Operation

SimpleText property

```
property SimpleText : string
```

Contains additional text that will be displayed on the dialog box if the foSimpleProgress option is True.

If the Options property includes foSimpleProgress, the text in SimpleText will be displayed on the progress dialog box in place of the file name. If Options does not include foSimpleProgress, SimpleText is ignored.

See also: Options

SourceFiles property

```
property SourceFiles : TStrings
```

Defines the path for the file or files on which the file operation will act.

Set SourceFiles to the file or files that the file operation will act on. SourceFiles may include an individual file, a list of files, or one or more directories with wildcards. If no path is provided the files are assumed to be in the current directory. SourceFiles is an instance of TStrings. If you perform multiple file operations, be sure to clear SourceFiles before adding new files or directories to the list. If the ConfirmFiles property is True, the list of files is checked for validity before performing the file operation.

Use SourceFiles when you want to copy or move files and rename the files at the same time. When used in this way, the entries in SourceFiles must follow the TStrings Name=Value format, where Name is the source file name and directory and Value is the destination file name and directory. For example, if the Operation property was set to foCopy and the following file mappings in the SourceFiles property:

```
  c:\shellshock\stfileop.pas=c:\backup\stfileop.bak
  c:\shellshock\stbase.pas=c:\backup\stbase.bak
  c:\shellshock\stshbase.pas=c:\backup\stshbase.bak
```

Given this file mapping, the files would be copied to the C:\BACKUP directory and given a .BAK extension. If the file names are not qualified with paths, the current directory is used. If SourceFiles contains file mappings, Destination is ignored. If either side of the equal sign is blank, an EStFileOpError exception is raised when the Execute method is called.

See also: ConfirmFiles, Destination, Execute, SourceFiles

Version read-only property

```
property Version : string
```

The current version of ShellShock.

Version is provided so you can identify your ShellShock version if you need technical support. You can display the ShellShock about box by double-clicking this property or selecting the dialog button to the right of the property value.

TStFormatDrive Component

The TStFormatDrive component is used to format a removable drive. Simply set the Drive property to the drive you wish to format and call the Execute method. When you call Execute, Windows will display the Format dialog box. If an error occurs during formatting, Execute will return False. You can read the Error property to determine the cause of the error.

Hierarchy

TComponent (VCL)

TSsComponent (StBase)

TSsShellComponent (StShBase)

TStCustomFormatDrive (StFormat)

TStFormatDrive (StFormat)

Properties

Drive	ErrorString	Version
Error	Options	

Methods

Execute

Events

OnDisketteError

Reference Section

Drive — property

```
property Drive : string
```

Specifies the removable drive to format.

Set Drive to the drive letter of the removable drive you wish to format. If Drive is invalid (nonexistent, network, or fixed drive), an EStShellFormatError exception is raised when the Execute method is called.

See also: Execute

Error — read-only, run-time property

```
property Error : Integer
```

Contains the error code of the last operation.

Read Error to determine the result of the format operation. There are two places you will check Error. The first is in your OnDisketteError event handler. Error will contain the error code Windows returns as the result of trying to access the drive (as determined by the Drive property). The second place to check Error is after Execute returns. If Execute returns False, check the value of Error to determine why formatting failed. After Execute returns, Error will contain one of the following values:

Value	**Meaning**
SHFMT_ERROR	The format failed.
SHFMT_CANCEL	The user cancelled the format with the Cancel or Close button.
SHFMT_NOFORMAT	The diskette could not be formatted.

If no error occurred, Execute returns True and Error will be 0. Read ErrorString to get a text description of the error.

See also: ErrorString, Execute, OnDisketteError

ErrorString — read-only, run-time property

```
property ErrorString : string
```

↳ Contains a text description of the last error.

ErrorString can be used to display a message to your users in the event an error occurs during the format operation.

See also: Error

Execute — method

```
function Execute : Boolean;
```

↳ Displays the shell format dialog box.

Call Execute to display the format dialog box. Execute returns True if the format completed successfully, and False if the format failed or was cancelled by the user. If Execute returns False, check the value of the Error property to determine the cause of the failure. There are no provisions for formatting a diskette silently (without showing the format dialog box.)

See also: Error, ErrorString

OnDisketteError — event

```
property OnDisketteError : TStDisketteErrorEvent

TStDisketteErrorEvent = procedure(
  Sender : TObject; var Retry : Boolean) of object;
```

↳ Defines an event handler that is called when an error occurs on the drive being formatted.

When Execute is called, the OnDisketteError event will be generated if an error occurs on the target drive. When OnDisketteError is defined, Windows critical errors are trapped within the component and passed to the OnDisketteError event handler. This prevents the standard windows error dialog box or, worse, the infamous Windows blue screen. Set Retry to True to allow the format operation to continue, or False to abort the format operation. Retry is False by default.

The Error property can be used to determine the cause of the error. An error might occur if there is no diskette in the drive, if the diskette in the drive is damaged, if the diskette in the drive is not formatted, or if the diskette in the drive is write protected. Some of the possible error codes are:

Error Code	Description
ERROR_NOT_READY	The disk drive is not ready. The most likely cause is no diskette in the drive or the drive door is open.
ERROR_WRITE_PROTECT	The diskette in the drive is write protected.
ERROR_CRC	CRC error, probably due to a damaged diskette.
ERROR_READ_FAULT	The diskette cannot be read, probably due to a damaged diskette.

This list represents the most common error conditions, and not all possible errors. ErrorString can be used to display a message to the user. The following example shows an OnDisketteError event handler which displays a message and gives the user the option to abort or retry:

```
procedure TForm1.StFormatDrive1DisketteError(
  Sender : TObject; var Retry : Boolean);
  var
    Res : Word;
  begin
    Res := MessageDlg(StFormatDrive1.ErrorString,
      mtError, [mbRetry, mbAbort], 0);
    Retry := (Res = mrRetry);
  end;
```

See also: Error, ErrorString

4

Options property

```
property Options : TStFormatOptionsSet
```

```
TStFormatOptionsSet = set of TStFormatOptions;
```

```
TStFormatOptions = (fmtSystemOnly, fmtFull);
```

↳ Determines which options are enabled on the format dialog box.

When fmtFull is True, the “Quick (erase)” check box on the format dialog box is not checked (the check box is called “Quick Format” on Windows NT). When fmtFull is False, the “Quick (erase)” check box is checked. The fmtSystemOnly element only applies to the Windows 95 family of operating systems. When fmtSystemOnly is True, the “Copy system files” check box is checked and all other options (Quick and Full) are disabled. Under Windows NT this value is ignored. In either case, the user can override the default settings after the dialog box is displayed.

Version read-only property

```
property Version : string
```

↳ The current version of ShellShock.

Version is provided so you can identify your ShellShock version if you need technical support. The ShellShock about box can be displayed by double-clicking this property or selecting the dialog button to the right of the property value.

TStTrayIcon Component

The TStTrayIcon component encapsulates an icon in the system tray. Using TStTrayIcon you can have tray icon support with a minimum amount of coding. The default values for the TStTrayIcon properties cause your application to behave as most tray applications behave. Specifically, the application will allow these behaviors:

- The icon is automatically placed in the system tray at start-up.
- Double-clicking the tray icon restores the application.
- When the application is restored the application’s tray icon remains in the tray.
- Clicking the close box when the application is in its normal state causes the application to go to the tray, but does not close the application.
- Clicking the minimize button when the application is in its normal state minimizes the application to the task bar, but does not send the application to the system tray.

There is no clear set of rules on how a tray icon application should operate. Some applications go to the tray when minimized while others only go to the tray when the close box is clicked. Some tray icon applications leave the icon in the tray when the application is restored, others remove the icon from the tray. The behavior of your application when accepting the default TStTrayIcon properties represents the most common tray icon application behavior. If you want your application to start minimized be sure to set the WindowState property of the main form to wsMinimized.

You can override the default behavior by changing the CloseToTray, MinimizeToTray, and HideOnRestore properties. To allow the system close box to close the application rather than sending the application to the tray, set the CloseToTray to False. Likewise, set the MinimizeToTray property to True if you want the minimize button to send the application to the tray rather than to the task bar. To remove the tray icon from the tray when the application is restored, set the HideOnRestore parameter to True.

Note: TStTrayIcon is designed to be used only on an application’s main form.

Hierarchy

TComponent (VCL)

  TSsComponent (StBase)

    TSsShellComponent (StShBase)

      TStCustomTrayIcon (StTrIcon)

        TStTrayIcon (StTrIcon)

Properties

Active	Icon	PopupMenu
Animate	ImageIndex	ShowHint
CloseToTray	Images	Version
HideOnRestore	Interval	
Hint	MinimizeToTray	

Methods

AddToTray	MinimizeApplication
DeleteFromTray	RestoreApplication

Events

OnClick	OnMinimize
OnDblClick	OnRestore

Reference Section

Active property

```
property Active : Boolean
```

Default: True

Determines whether or not the tray icon is active and displayed in the tray.

Read Active to determine whether or not the icon is in the tray. Set Active to True to place the icon in the system tray, or False to remove the Icon from the system tray. If the icon cannot be added or removed, an EStTrayIconError exception is raised.

See also: AddToTray

AddToTray method

```
procedure AddToTray;
```

Adds the icon to the system tray.

Call AddToTray to add the icon to the system tray. The displayed icon will be the icon specified in the Icon property. If Icon is nil, the icon will be taken from the ImageList property. The ImageIndex property determines the icon in the image list to display in the tray. If both Icon and Images are nil, the application's icon is used. If the icon cannot be added to the system tray an EStTrayIconError exception is raised. Normally you will use the Active property rather than calling AddToTray.

See also: Active, DeleteFromTray

Animate property

```
property Animate : Boolean
```

Default: False

↳ Determines whether or not the tray icon is animated.

Set Animate to True to start animation or False to stop animation. TStTrayIcon uses an image list to perform the animation rather than a true animated icon. The ImageIndex property contains the list of images in the animation. The animation speed is determined by the Interval property.

When animation stops, a static icon will be displayed. The static icon is either the icon defined by the Icon property or by the ImageIndex property. If Icon is assigned, it will be used as the static icon. If Icon is nil, the icon in the image list as determined by the ImageIndex property is used as the static icon.

See also: ImageIndex, Images, Interval

CloseToTray property

```
property CloseToTray : Boolean
```

Default: True

↳ Determines the behavior of the close box of the application.

Most tray icon programs don't terminate when the application's close box is clicked. Instead, clicking the close box typically sends the application back to the tray. This is the default behavior of TStTrayIcon. If you want your application to terminate when the close box is clicked, set CloseToTray to False.

See also: MinimizeToTray

DeleteFromTray method

```
procedure DeleteFromTray;
```

↳ Removes the icon from the system tray.

Call DeleteFromTray to remove the icon from the system tray. You will typically use the Active property rather than calling DeleteFromTray directly. An EStTrayIconError exception is raised if an error occurs removing the tray icon.

See also: Active, AddToTray

HideOnRestore — property

```
property HideOnRestore : Boolean
```

Default: False

Determines the behavior of the tray icon when the application is restored.

Most tray icon programs leave the icon in the tray when the application is restored. If you don't want your icon left in the tray, set HideOnRestore to False.

Hint — property

```
property Hint : string
```

Defines the hint text for the tray icon.

Hint is the hint text that is displayed when the mouse cursor pauses over the tray icon. The ShowHint property determines whether or not the hint is shown.

See also: ShowHint

Icon — property

```
property Icon : TIcon
```

Sets the icon that will be displayed in the system tray.

Set Icon at design time or run time to set the icon that is displayed in the system tray. If Icon is not assigned, the application's icon will be used. If the icon cannot be set, an EStTrayIconError exception is raised.

See also: ImageIndex, Images

ImageIndex — property

```
property ImageIndex : Integer
```

Default: 0

Sets the index in the image list that is be used for the tray icon.

Set ImageIndex to the image in ImageList that you want displayed when animation is off. If you specify an image index, do not assign an icon to the Icon property. If Icon is assigned then it is used for the static icon and ImageIndex is ignored.

See also: Animate, Images, Icon

Images property

```
property Images : TImageList
```

↳ Sets the list of icons that can be displayed in the system tray.

The images in ImageList can be bitmaps or icons. When Animate is true, TStTrayIcon will cycle through the images in ImageList to provide animation. You can use ImageList even if you are not using animation. In that case ImageList is a list of static icons from which to choose. Set the ImageIndex property to the index of the image you want displayed in the system tray.

See also: Animate, ImageIndex, Icon

Interval property

```
property Interval : Integer
```

Default: 250

↳ Sets the timing interval between animation frames, in milliseconds.

Set Interval to an integer value to control the animation speed. The practical minimum value for Interval is 55ms. Setting Interval to a value less than 55ms will not result in any appreciable difference in the animation rate. A value of 1000ms will result in the frame changing once per second.

See also: Animate, Images

MinimizeApplication method

```
procedure MinimizeApplication;
```

↳ Minimizes the application and enables the tray icon.

Calling Application.Minimize does not result in the correct messages being sent to TStTrayIcon. Use MinimizeApplication rather than Application.Minimize to ensure that the icon is added to the system tray when the application is minimized.

See also: RestoreApplication

MinimizeToTray property

```
property MinimizeToTray : Boolean
```

Default: False

Determines the behavior of the minimize button on the application.

There are two schools of thought regarding the minimize button and tray icon applications. One says that the application should minimize as normal when the minimize button of the application is clicked. The other school of thought says that the application should be sent to the tray when the minimize button is clicked. The default for MinimizeToTray is False. If you want your application to go to the tray when the minimize button is clicked, set MinimizeToTray to True.

See also: CloseToTray

OnClick event

```
property OnClick : TTrayClickEvent
```

```
TTrayClickEvent = procedure(Sender : TObject;
  Button : TMouseButton; Shift : TShiftState) of object;
```

Defines an event handler that is called when the tray icon is clicked.

Respond to OnClick to perform some operation when the icon is clicked. Button contains the mouse button that was clicked. Shift indicates whether any of the Alt, Ctrl, or Shift keys were down when the icon was clicked. Note that if the PopupMenu property is assigned, the popup menu automatically appears when the right button is clicked. There is no need to respond to OnClick to display the popup menu.

See also: OnDblClick, PopupMenu

OnDblClick event

```
property OnDblClick : TTrayDblClickEvent
```

```
TTrayDblClickEvent = procedure(
  Sender : TObject; Button : TMouseButton; Shift : TShiftState;
  var RestoreApp : Boolean) of object;
```

↳ Defines an event handler that is called when the tray icon is double-clicked.

Respond to OnDblClick to perform application-specific tasks when the tray icon is double-clicked. Button contains the mouse button that was clicked. Shift indicates whether any of the Alt, Ctrl, or Shift keys were down when the icon was clicked. By default, double clicking the tray icon will restore the application. If you don't want the application restored when the icon is double-clicked, set the RestoreApp parameter to False in your OnDblClick event handler.

See also: OnClick

OnMinimize event

```
property OnMinimize : TNotifyEvent
```

↳ Defines an event handler that is called when the application is minimized.

The OnMinimize event will be called when the application owning the TStTrayIcon component is minimized. Respond to this event if you wish to display the tray icon when your application is minimized and Active is False.

See also: OnRestore

OnRestore event

```
property OnRestore : TNotifyEvent
```

↳ Defines an event handler that is called when the application is restored.

When the application owning the TStTrayIcon component is restored, the tray icon stays in the system tray. If you want to remove the tray icon when the application is restored, set the HideOnRestore property True.

See also: HideOnRestore, OnMinimize

PopupMenu — property

```
property PopupMenu : TPopupMenu
```

Specifies the popup menu that will be displayed when the tray icon is right-clicked.

Set PopupMenu to a TPopupMenu on your form. When the tray icon is right-clicked, the popup menu will be displayed.

RestoreApplication — method

```
procedure RestoreApplication;
```

Restores a minimized tray icon form.

Call RestoreApplication to restore a minimized tray icon form. If HideOnRestore is True, the icon is removed from the system tray.

See also: HideOnRestore, MinimizeApplication

ShowHint — property

```
property ShowHint : Boolean
```

Default: True

Determines whether the tray icon's hint is displayed when the mouse cursor pauses over the tray icon.

Set ShowHint to True to show the tray icon's hint text (set by the Hint property). Set ShowHint to False to suppress the hint text.

See also: Hint

Version — read-only property

```
property Version : string
```

The current version of ShellShock.

Version is provided so you can identify your ShellShock version if you need technical support. The ShellShock about box can be displayed by double-clicking this property or selecting the dialog button to the right of the property value.

TStDropFiles Component

The TStDropFiles component is used to enable dropping of files onto your application. The DropTarget property determines the component that accepts dropped files. DropTarget can be set to any TWinControl descendant, including a form (such as an application's main form). The Active property determines whether or not dropping of files is currently enabled. When files are dropped on the drop target, the OnDropFiles event is generated. The Files property contains a list of the files dropped.

Hierarchy

TComponent (VCL)

- TSsComponent (StBase)
  - TSsShellComponent (StShBase)
    - TStCustomDropFiles (StDrop)
      - TStDropFiles (StDrop)

Properties

Active	DropTarget	TargetStringList
Count	Files	Version

Events

OnDropFiles

Reference Section

Active — property

```
property Active : Boolean
```

Determines whether the drop target accepts dropped files.

Set Active to True to accept dropped files, or False to stop accepting dropped files. When Active is True, the drag files cursor is displayed when the mouse cursor is moved over the drop target. When Active is False, the no-drop cursor is displayed when the mouse cursor is moved over the drop target.

See also: DropTarget

Count — read-only, run-time property

```
property Count : Integer
```

Displays the number of files dropped on the drop target.

Read Count to determine the number of dropped files. Read Count in an OnDropFiles event handler. The Files property contains a list of the dropped files.

See also: Files, OnDropFiles

DropTarget — property

```
property DropTarget : TWinControl
```

The control that accepts dropped files.

DropTarget can be set to any TWinControl descendant. Typical drop targets include memos, rich edits, list boxes, tree views, list views, and forms (although many other possibilities exist). If DropTarget is set to a control not derived from TWinControl an EStDropFilesError exception is raised. The OnDropFiles event is generated when files are dropped on the drop target and Active is True.

See also: Count, Files, OnDropFiles

Files — read-only, run-time property

```
property Files : TStrings
```

↳ Contains a list of the files dropped on the drop target.

Read Files in the OnDropFiles event. The Count property contains the number of files dropped.

See also: Count, OnDropFiles, TargetStringList

OnDropFiles — event

```
property OnDropFiles : TStDropFilesEvent
```

```
TStDropFilesEvent = procedure(
  Sender : TObject; Point : TPoint) of object;
```

↳ Defines an event handler that is called when files are dropped on the drop target.

Respond to the OnDropFiles event to determine the number of files dropped, and the file names of each file dropped. The Files property contains a list of the files dropped, and the Count property contains the number of files dropped. OnDropFiles will only be generated when the Active property is True.

See also: Active, Count, Files

TargetStringList — read-only, run-time property

```
property TargetStringList : TStrings
```

↳ A TStrings object that receives dropped files.

TargetStringList can be used to assign a TStrings object (or descendant) that will automatically receive the files dropped on the drop target. For example, let’s say you had a list box that you wanted to fill with the files dropped on the application. In that case, you would set DropTarget to the main form and TargetStringList to the list box’s Items property. For example:

```
  StDropFiles.TargetStringList := ListBox.Items;
```

When files are dropped on the drop target (the form in this example), the list box will be filled with a list of the dropped files. If TargetStringList is used, it is not necessary to specifically respond to the OnDropFiles event.

See also: OnDropFiles

Version — read-only property

```
property Version : string
```

The current version of ShellShock.

Version is provided so you can identify your ShellShock version if you need technical support. The ShellShock about box can be displayed by double-clicking this property or selecting the dialog button to the right of the property value.

TStShortcut Component

TStShortcut is a component which creates Windows shortcuts (shell links). Shortcuts can be created in many locations: on the desktop, on the Start menu, on the Programs menu, in a directory, in the Documents (recent files) list, or in one of several other special folder locations. Simply set the target file name and shortcut destination, and call the CreateShortcut method. You can also resolve a shortcut. Resolving a shortcut means to get information about the shortcut's target file including the file name, show command, "start in" directory, and hotkey.

Hierarchy

TComponent (VCL)

- TSsComponent (StBase)
  - TSsShellComponent (StShBase)
    - TStCustomShortcut (StShrtct)
      - TStShortcut (StShrtct)

Properties

AutoName	IconIndex	ShowCommand
Description	IconPath	SpecialFolder
DestinationDir	LocationType	StartInDir
FileName	Parameters	Version
HotKey	ShortcutFileName	

Methods

CreateShortcut	ResolveShortcut

Reference Section

AutoName property

```
property AutoName : Boolean
```

Default: True

Determines whether the shortcut will be automatically given a traditional shortcut name.

When AutoName is True, the shortcut name will be automatically generated. The shortcut name is the text “Shortcut to” followed by the file name. For example, if the FileName property is set to C:\MYAPP.EXE, TStShortcut will create a shortcut with a description of “Shortcut to Myapp.exe.” The actual shortcut file name will be “Shortcut to Myapp.exe.lnk” (all shortcuts have a file name extension of LNK). If AutoName is False, the shortcut name will be the text contained in the Description property.

See also: Description, FileName, ShortcutFileName

CreateShortcut method

```
function CreateShortcut  : Boolean;
```

Creates the shortcut.

Call CreateShortcut to perform the actual creation of the shortcut. CreateShortcut returns True if the shortcut was created successfully, or False if the shortcut was not created. Set the FileName and Location properties (at a minimum) before calling CreateShortcut. Optional properties include Description, ShowCommand, HotKey, and StartInDir.

CreateShortcut will raise an EStShortcutError exception if FileName is not a valid file, if an error occurs initializing the Windows COM interface, or if the Location is set to slDirectory and the Directory property is blank.

See also: Description, FileName, HotKey, Location

Description — read-only, run-time property

```
property Description : string
```

Contains the description of the shortcut.

Description has two uses. The first use is to set the shortcut name. If AutoName is False, the text of Description is used as the shortcut file name. This is also the text that appears below the shortcut when the shortcut is created on the desktop. If AutoName is True, Description is ignored.

The second use of Description is more theoretical than real. The Microsoft documentation for the shortcut operations says that the Description text will be saved with the shortcut and used as the shortcut's caption. This does not appear to work as designed, but there is always the chance that it will be fixed in Windows 2000. TStShortcut saves the description text along with the shortcut in the event that this description will someday be used by the operating system.

See also: AutoName, CreateShortcut

DestinationDir — property

```
property DestinationDir : string
```

The directory where the shortcut will be saved.

When saving a shortcut to a directory, set Location to slDirectory and the DestinationDir property to the directory where the shortcut is to be created. DestinationDir is ignored if Location is not slDirectory.

See also: Location

FileName — property

```
property FileName : string
```

Determines the shortcut's target file name.

FileName is the file on disk to which the shortcut is linked. If FileName is not a valid file on disk, an EStShortcutError exception is raised when the CreateShortcut method is called. Read FileName after resolving a shortcut to determine the shortcut's target file name.

See also: CreateShortcut, ResolveShortcut, ShortcutFileName

HotKey property

```
property HotKey : Word
```

Default: 0

Specifies the keyboard hotkey combination associated with the shortcut.

Set HotKey to the hotkey combination for the shortcut. Read HotKey after resolving a shortcut to obtain the shortcut's hotkey.

See also: CreateShortcut, ResolveShortcut

IconIndex property

```
property IconIndex : Integer
```

Default: 0

Specifies an icon index for the shortcut.

IconIndex and IconPath are used together to specify the location of an icon to use for the shortcut. IconPath is the location of a file containing the icon. IconIndex is the index of the icon within the file specified by IconPath.

See also: IconPath

IconPath property

```
property IconPath : string
```

Specifies the location of a file containing the icon for the shortcut.

IconIndex and IconPath are used together to specify the location of an icon to use for the shortcut. IconPath is the location of a file containing the icon. IconIndex is the index of the icon within the file specified by IconPath.

See also: IconIndex

LocationType property

```
property LocationType : TShortcutLocation
```

```
TStLocationType = (ltWorkingDir, ltSpecialFolder, ltDirectory);
```

Default: ltSpecialFolder

↳ Describes the location type for the shortcut.

LocationType can be one of the following values:

Value	Meaning
ltWorkingDir	The shortcut will be created in the current working directory.
ltSpecialFolder	The shortcut will be created in the special folder identified by the SpecialFolder property.
ltDirectory	The shortcut will be created in the directory defined by the DestinationDir property.

The default location for a shortcut is the Windows desktop. For this reason, the default value for LocationType is ltSpecialFolder and the default for the SpecialFolder property is sfDesktop.

See also: DestinationDir, SpecialFolder

Parameters property

```
property Parameters : string
```

↳ Contains the command-line arguments for the shortcut.

Set Parameters prior to creating a shortcut to specify the command-line arguments for the shortcut. Read Parameters after calling ResolveShortcut to determine the shortcut's command-line arguments.

ResolveShortcut method

```
function ResolveShortcut : Boolean;
```

↳ Resolves a shortcut to provide information about the target file.

You have probably seen Windows attempting to resolve a shortcut when looking for a missing file. When Windows is resolving a shortcut the Missing Shortcut dialog is displayed with a flashlight moving back and forth across the dialog. To resolve a shortcut set the ShortcutFileName property to a shortcut file and call ResolveShortcut. ResolveShortcut returns True on success, or False if an error occurs. If ResolveShortcut returns True, the Description, FileName, HotKey, ShowCommand, and StartInDir properties will contain information about the shortcut. An EStShortcutError exception is raised if the ShortcutFileName property is blank, if the file does not exist, or if the file is not a shortcut file.

See also: FileName, CreateShortcut, Description, HotKey, ShortcutFileName, ShowCommand, StartInDir

ShortcutFileName property

```
property ShortcutFileName : string
```

↳ Defines the file name of the shortcut.

Shortcuts are technically called link files. A link file has a file name extension of LNK. ShortcutFileName is the actual shortcut link file. Read ShortcutFileName to determine the name of the shortcut file after calling CreateShortcut. When resolving a shortcut, set the ShortcutFileName property prior to calling ResolveShortcut. Note that you do not need to set ShortcutFileName when creating shortcuts. ShortcutFileName is automatically generated when CreateShortcut is called.

See also: CreateShortcut, FileName, ResolveShortcut

ShowCommand property

```
property ShowCommand : TShowState
```

```
TShowState = (ssNormal, ssMinimized, ssMaximized);
```

Default: ssNormal

↳ The Windows show command of the shortcut.

Set ShowCommand prior to calling CreateShortcut to set the show command for the shortcut. Read ShowCommand after calling ResolveShortcut to determine the show command for a shortcut.

See also: CreateShortcut, ResolveShortcut

SpecialFolder property

```
property SpecialFolder : TStSpecialRootFolder
```

```
TStSpecialRootFolder = (sfAltStartup, sfAppData, sfBitBucket,
  sfCommonAltStartup, sfCommonDesktopDir, sfCommonFavorites,
  sfCommonPrograms, sfCommonStartMenu, sfCommonStartup,
  sfControls, sfCookies, sfDesktop, sfDesktopDir, sfDrives,
  sfFavorites, sfFonts, sfHistory, sfInternet, sfInternetCache,
  sfNetHood, sfNetwork, sfNone, sfPersonal, sfPrinters,
  sfPrintHood, sfPrograms, sfRecentFiles, sfSendTo, sfStartMenu,
  sfStartup, sfTemplates);
```

Default: sfDesktop

↳ Identifies the special folder location when the FolderType property is set to ftSpecialFolder.

This is the folder where the shortcut will be created when CreateShortcut is called. Some of the elements of TStSpecialRootFolder are only applicable to version 4.70 of the shell and later. If you specify an invalid root folder for the installed shell, an EStShellError exception is raised. You should be prepared to handle this exception when you set SpecialFolder. The elements specific to shell version 4.70 and later are:

```
  sfInternet
  sfAltStartup
  sfCommonAltStartup
  sfCommonFavorites
  sfInternetCache
  sfCookies
  sfHistory
```

See also: CreateShortcut, LocationType

StartInDir — property

```
property StartInDir : string
```

↳ Specifies the working directory for the shortcut.

Set StartInDir to a directory in which the application associated with the shortcut should start. This directory serves as the working directory for the application. Read StartInDir after resolving a shortcut to determine the shortcut's working directory.

See also: CreateShortcut, ResolveShortcut

Version — read-only property

```
property Version : string
```

↳ The current version of ShellShock.

Version is provided so you can identify your ShellShock version if you need technical support. The ShellShock about box can be displayed by double-clicking this property or selecting the dialog button to the right of the property value.

4

Identifier Index

Subject Index

For over fifteen years you've depended on TurboPower to provide the best tools and libraries for your development tasks. Now try Orpheus 4 and Async Professional 4—two of TurboPower's best selling products—risk free. Both are fully compatible with Borland Delphi and C++Builder, and are backed with our expert support and 60-day money back guarantee.

2001 READERS CHOICE Delphi INFORMANT MAGAZINE COMPANY OF THE YEAR

The TurboPower family of tools-
Winners of 6 *Delphi Informant* Readers' Choice Awards
for 2001! Company of the Year in 2000 and 2001.

DEVELOP, DEBUG, OPTIMIZE
FROM START TO FINISH, TURBOPOWER
HELPS YOU BUILD YOUR BEST

ORPHEUS 4™

Make the first impression count with new Orpheus 4, the award-winning user interface construction kit from TurboPower Software Company. Your customers will feel comfortable the minute your application starts, resulting in them spending less time learning how to use it and more time getting to know what makes it special.

ASYNC PROFESSIONAL 4™

Async Professional 4 VCL delivers full serial port control, powerful faxing power, Internet-ready Winsock access, and real world business telephony all in one easy to use, incredibly powerful package. New APRO VCL adds the advanced new features you've demanded like speech synthesis and recognition (even over the phone!), IP Telephony (supporting both audio and video), GSM/SMS paging, visual state machines, a new XML-based modem database, and more.

Try the full range of TurboPower products. Download free Trial-Run Editions from our Web site.

www.turbopower.com

ShellShock requires Microsoft Windows (XP, 2000, Me, NT or 9X) and Borland Delphi 3 and above, or C++Builder 3 and above

TurboPower Software Company
©2002, TurboPower Software Co.

TURBOPOWER® Software Company