Separators

See Also

History List MenuAfter MaxHistory

The **Separators** property allows you to specify menu separators above and below the History List.

The possible values for the **Separators** property are:

msTop	Include separator above History List
msBottom	Include separator below History List
msBoth	Include both top and bottom separators
msNone	Do not create any separators

If you define a top level menu item for the sole purpose of displaying the History List, you will likely want to set Separators to **msNone**.

If you are displaying the History List within another menu, for example; to follow the Save As menu item within the File menu, you will likely set Separators to **msBoth**.

If the History List will appear at the top or bottom of a menu that has other items on it, you would use either **msTop** or **msBottom**.

If you have already placed a separator in your menu and you assign this to **MenuAfter**, then you will likely want to set Separators to **msBottom**.

FormCapOption

The **FormCapOption**, of type TCaptionOpt, allows you to specify handling for the parent form's title bar (Caption property). The possible settings are:

coOff	Do not handle form's caption
coWin31	Format in Windows 3.x style
coWinNew	Format in "Windows '95" style

The Windows 3.1 style is normally the executable name of the application followed by the currently open document, for example:

MyApplication: C:\Delphi\SomeFile

The new style that should be used with Windows '95 is the name of the currently open document followed by the executable name of the application in brackets, for example:

SomeFile - MyApplication

MenuAfter

See Also

HistoryList MaxHistory Separators

If you wish to implement the History List feature of NOSC, you must tell NOSC where to place it. You do this through the **MenuAfter** property. When you click the setting field for **MenuAfter** in the Object Inspector, you'll be able to display a list of all TMenuItem components currently defined in your form (this implies, of course, that you must already have defined a menu for the form).

The History List will appear directly below the menu item you specify in **MenuAfter**.

You are free to set **MenuAfter** to any available TMenuitem component. it can be a sub-menu item within another, for example; the Save As menu item within the File menu, or it can be a top-level menu item. If you select a top-level menu item, the File History will always appear as the first set of items in that menu; any other menu items you define for that menu will follow the File History.

MaxHistory

See Also

HistoryList MenuAfter

The **MaxHistory** property allows you to specify the maximum number of file names to be stored in the History List. If you do not want to use the File History feature, simply set **MaxHistory** to zero.

MaxHistory defaults to a value of nine. This is generally the most you would want to allow as this allows each file name to be prefixed with an accelerator key using the digits from 1 to 9. This many also fit comfortably in a File menu containing just the standard menu items of New, Open, Save, Save As, and Exit. If your File menu contains more than these standard items, you will likely want to decrease the **MaxHistory** setting.

If you are defining a top-level menu item for the sole purpose of containing a File History, then you can comfortably set **MaxHistory** to more than nine. In this case, only the first nine history menu items will have accelerator keys assigned (digits 1 to 9). All following history menu items will need to be selected without the use of an accelerator (this generally isn't a big concern).

IniFileName

See Also

KeepIniLocal IniFile

In order to implement the History List, and other features, NOSC must store information in an initialization file. The **IniFileName** property defines the file name given to the initialization file. When you first look at this property in the Object Inspector, you will see a default name of "[ExeName].INI". This, of course, is not the actual name NOSC will try to use, it's NOSC's way of telling you that the file will be named according to the executable name of your application. For example, if you compile your project to an executable name of WIDGET.EXE, then NOSC will use an initialization file called WIDGET.INI.

Although not recommended, you are allowed to change this default to any legal file name you wish to use.

You do not need to be concerned with creating the initialization file; NOSC will create and maintain the file automatically through the use of Delphi's standard TIniFile component. You also have access to the TIniFile component at runtime via the IniFile property. This means you are free to use IniFile to store and retrieve any other information you may need for your application in the same initialization file without needing to define and instantiate your own TIniFile component.

IniFile

See Also

KeepIniLocal IniFileName

IniFile is a runtime-only property that gives you access to the application's initialization file. NOSC creates this component in order to save information about the current state of its various features. You are free to access this property and its methods in order to store any other information you need for your application. There is no need to create your own TIniFile component unless you wish to maintain another initialization file with a different name.

NOSC stores all of its information in a single section called **[History]**. Although you can also store information in this section, it is advisable to create your own section names to insure you do not conflict with any key values within the [History] section needed by NOSC..

SaveFormPos

See Also

IniFile

SaveFormPos is a Boolean property. If set True, NOSC will store the Form's current size and position information to the initialization file before closing. Each time the application is run, it will retrieve this information and set the opening size and position of the form accordingly.

AutoOpen, AutoNew

See Also:

OnOpen OnNew Open and Save Dialogs UntitledString

If you set **AutoOpen** to True, NOSC will check the current file name as loaded from the initialization file. If it is the name of an existing file, NOSC will issue an immediate OnOpen event. If this name does not represent an existing file, and the **AutoNew** property is not set, the user will first be presented with the Open Dialog. If the user selects a file from the **Open Dialog**, NOSC will then issue the **OnOpen** event, else the application will display initially with **[Untitled]** as the file name. If AutoNew is also set and there is no valid file to be auto-opened, NOSC will instead issue an immediate **OnNew** event.

If **AutoNew** is set and **AutoOpen** is not, NOSC will immediately issue an **OnNew** event when the application starts.

KeepIniLocal

See Also

IniFile IniFileName

KeepIniLocal is a Boolean property that allows you to control where NOSC stores your application's initialization (INI) file.

Setting **KeepIniLocal** to True causes NOSC to store the initialization file in the same directory as the application's executable module. Setting it to False causes the initialization file to be placed in the Windows directory.

UntitledString

The **UntitledString** property lets you define what string is to be displayed in place of a file name when there is currently no existing file open or when the user creates a new file. The default is **[Untitled]**, but you can set this to anything you like.

MenuNew

See Also:

MenuOpen MenuSave MenuSaveAs Events

If you have a menu item to allow the user to create new files (normally called New), assign this menu item to the **MenuNew** property in Object Inspector. This allows NOSC to handle OnClick events for this menu item.

MenuOpen

See Also:

MenuNew MenuSave MenuSaveAs Events

If you have a menu item to allow the user to open files (normally called Open), assign this menu item to the **MenuOpen** property in Object Inspector. This allows NOSC to handle OnClick events for this menu item.

MenuSave

See Also:

MenuNew MenuOpen MenuSaveAs Events

If you have a menu item to allow the user to save changes to files (normally called Save), assign this menu item to the **MenuSave** property in Object Inspector. This allows NOSC to handle OnClick events for this menu item.

MenuSaveAs

See Also:

MenuNew MenuOpen MenuSave Events

If you have a menu item to allow the user to save files under a different name (normally called Save As), assign this menu item to the **MenuSaveAs** property in Object Inspector. This allows NOSC to handle OnClick events for this menu item.

FileName

The **FileName** property contains the name of the file currently open, or, if current file has not yet been given a name, it will be equal to the UntitledString property.

When you receive an OnOpen event, **FileName** is the file to be opened.

When you receive an OnSave, OnSaveAs, or OnSaveQuery event, **FileName** is the name to use for saving the file.

osDefaultExt

See Also:
Open and Save Dialogs _____ Properties

The **osDefaultExt** property represents the **DefaultExt** property of both the **TOpenDialog** and **TSaveDialog** components. Set this property as you would for either of those components.

osFilter

See Also:

Open and Save Dialogs Properties

The **osFilter** property represents the **Filter** property of both the **TOpenDialog** and **TSaveDialog** components. Set this property as you would for either of those components.

osFilterIndex

See Also:

Open and Save Dialogs _____ Properties

The **osFilterIndex** property represents the **FilterIndex** property of both the **TOpenDialog** and **TSaveDialog** components. Set this property as you would for either of those components.

osOpenOptions

See Also:

Open and Save Dialogs Properties osSaveOptions

The **osOpenOptions** property represents the **Options** property of the **TOpenDialog** component. Set this property as you would it.

NOSC initializes this property with the options most commonly needed for a typical application and there is a good chance you will not need to alter them. When you place your own TOpenDialog component, all options default to all False, NOSC defaults the following options to True:

ofHideReadOnly ofPathMustExist ofFileMustExist ofShareAware ofNoReadOnlyReturn

osSaveOptions

See Also:

Open and Save Dialogs Properties osOpenOptions

The **osSaveOptions** property represents the **Options** property of the **TSaveDialog** component. Set this property as you would it.

NOSC initializes this property with the options most commonly needed for a typical application and there is a good chance you will not need to alter them. When you place your own TSaveDialog component, all options default to all False, NOSC defaults the following options to True:

ofOverwritePrompt ofHideReadOnly ofPathMustExist ofFileMustExist ofCreatePrompt ofShareAware

osOpenTitle

See Also:

Open and Save Dialogs _____Properties osSaveTitle

The **osOpenTitle** property represents the **Title** property of the **TOpenDialog** component. Set this property as you would it.

osSaveTitle

See Also:

Open and Save Dialogs Properties osOpenTitle

The **osSaveTitle** property represents the **Title** property of the **TSaveDialog** component. Set this property as you would it.

osOpenHelpCtx

See Also:

Open and Save Dialogs Properties osSaveHelpCtx

The **osOpenHelpCtx** property represents the **HelpContext** property of the **TOpenDialog** component. Set this property as you would it.

osSaveHelpCtx

See Also:

Open and Save Dialogs Properties osOpenHelpCtx

The **osSaveHelpCtx** property represents the **HelpContext** property of the **TSaveDialog** component. Set this property as you would it.

OnNew

See Also:

OnOpen OnSave OnSaveAs OnSaveQuery OnCloseQuery
Open and Save Dialogs

The **OnNew** event is issued to you by NOSC when the user selects the menu item named (or corresponding to) New.

Before issuing the **OnNew** event, NOSC first issues an OnSaveQuery event. This allows you to check for unsaved changes and whether the user wants to save them. If they are to be saved, NOSC then checks to see if the current file has a valid name. If not, it will display the Save Dialog. Once there is a valid name, NOSC will issue an OnSave event so you can perform the actual save. NOSC will then issue an **OnNew** event so you can perform the appropriate preparations.

When you receive an **OnNew** event, simply perform whatever is required to prepare for a new file.

Example:

```
procedure TForm1.NOSC1New(Sender: TObject; var Proceed: Boolean);
begin
   Memo1.Clear;
   Memo1.Modified := False;
end;
```

The **Proceed** parameter defaults to **True** on the assumption you will honor the user's New menu selection. If, for any reason, you cannot honor the request, you can set the Proceed parameter to False in order to inform NOSC that the currently open file is to remain current. **For the user's benefit, you should disable or gray the New menu item if you know you are not going to honor that selection.**

OnOpen

See Also:

OnNew OnSave OnSaveAs OnSaveQuery OnCloseQuery
Open and Save Dialogs

The **OnOpen** event is issued to you by NOSC when the user selects the menu item named (or corresponding to) Open.

Before issuing the **OnOpen** event, NOSC first issues an OnSaveQuery event. This allows you to check for unsaved changes and whether the user wants to save them. If they are to be saved, NOSC then checks to see if the current file has a valid name. If not, it will display the Save Dialog. Once there is a valid name, NOSC will issue an OnSave event so you can perform the actual save. NOSC will then issue an **OnOpen** event so you can perform the actual load and display of the file.

When you receive an **OnOpen** event, simply perform whatever is required to open and display selected file. The name of the file to be opened is in the FileName property.

Example:

```
procedure TForm1.NOSC1Open(Sender: TObject; var Proceed: Boolean);
begin
  Memo1.Lines.LoadFromFile(NOSC1.FileName);
  Memo1.Modified := False;
end;
```

The **Proceed** parameter defaults to **True** on the assumption you will honor the user's Open menu selection. If, for any reason, you cannot honor the request, you should set the Proceed parameter to False in order to inform NOSC that the currently open file is to remain current. **For the user's benefit, you should disable or gray the Open menu item if you know you are not going to honor that selection.**

OnSave

See Also:
OnNew OnOpen OnSaveAs OnSaveQuery OnCloseQuery Open and Save Dialogs

The **OnSave** event is issued to you whenever the current file requires saving. If the user selects the Save menu item, then the **OnSave** event is passed to you directly (NOSC will first present the user with the Save Dialog if the current file has not yet been given a name). For all other menu selections, the **OnSave** event is always preceded by the OnSaveQuery event.

When you receive an **OnSave** event, you simply need to perform the actual save.

Example:

```
procedure TForm1.NOSC1Save(Sender: TObject; var Proceed);
begin
  Memo1.Lines.SaveToFile(NOSC1.FileName);
end;
```

If, for any reason, you are not able to honor the save request at this time, you should set the **Proceed** parameter to False in order to inform NOSC. In this situation, NOSC will stop whatever action was in progress and the current file name will remain current.

OnSaveAs

See Also:

OnNew OnOpen OnSave OnSaveQuery OnCloseQuery
Open and Save Dialogs

The **OnSaveAs** event is only issued to you by NOSC whenever the user selects the menu item named (or corresponding to) Save As.

The main purpose of the **OnSaveAs** event is to simply inform the application that the user has requested this. It allows a chance to make any other preparations that may be necessary for the application. If you know beforehand you are not going to allow this action, you should disable or gray the Save As menu item (or not provide that menu item at all if you will never allow it). If necessary, you can set Proceed to False in order to disallow the request.

Example:

This example stops NOSC from displaying the Save Dialog for the user and cancels further related action.

```
procedure TForm1.NOSC1SaveAs(Sender: TObject; var Proceed: Boolean);
begin
   Proceed := False;
end;
```

OnCloseQuery

See Also:

OnNew OnOpen OnSave OnSaveAs OnSaveQuery Open and Save Dialogs

The **OnCloseQuery** event is issued to you by NOSC when the user attempts to close the application via the Close menu item within the application's control menu, or the <Alt>-<F4> shortcut, or if you call the form's Close method in your code.

NOSC passes this message on to you directly. You only need to respond to this event if there may be circumstances where you want to stop the application from being closed. By not coding a response method for it, or by responding but not setting CanClose to False, NOSC will subsequently issue an **OnSaveQuery** and possibly an **OnSave** event before allowing the application to terminate.

Example:

This example shows how to stop the application from terminating. Subsequent **OnSaveQuery** and **OnSave** events are cancelled.

```
procedure TForm1.NOSC1CloseQuery(Sender: TObject; var CanClose: Boolean);
begin
   CanClose := False;
end;
```

Create

See Also:
Destroy

You should never need (or want) to create a TNOSC component dynamically. A TNOSC component should be placed on your form at design-time. This will cause the Create constructor to be called by the form.

Destroy

See Also:
Create

You will normally never need to use the destroy method yourself. Since you should always want to place a TNOSC component on your form at design-time, your form will automatically call TNOSC.Destroy for you.

Tasks

See Also:

Properties Events

Basic Tasks

To implement the basic functionality of NOSC, assign the New, Open, Save, and Save As menu item components to the **MenuNew**, **MenuOpen**, **MenuSave** and **MenuSaveAs** properties of NOSC in the Object Inspector. On the Events page of Object Inspector, double-click the **OnNew**, **OnOpen**, **OnSave, OnSaveAs** and **OnSaveQuery** events (For more detail on these events, click Events above).

Additional Tasks

To implement the **History List** feature, click on the **MenuAfter** property and assign it to the menu item you wish the **History List** to follow. Typically this will be the File|Save As menu item, but it can be anywhere you like including top-level menu items. Optionally, change the **MaxHistory** property to any number you like (defaults to nine). Optionally set the **Separators** property to indicate which, if any, menu separators you want **NOSC** to use.

You can change control the name of the **TIniFile** component through the **IniFileName** property and you can control where this INI file is stored with the **KeepIniLocal** property.

You can have NOSC save the position and size of your form and automatically use this information each time your application starts by setting the **SaveFormPos** property to True.

You can have NOSC automatically update the form's **Caption** to display the name of the currently open file by setting **FormCaptionOpt** to either **coWin31** or **coWinNew**.

You can have NOSC send an immediate **OnOpen** or **OnNew** event each time the application starts by setting the **AutoOpen** and/or **AutoNew** property to True.

Properties

See Also:

Events Tasks

The following are runtime and read-only only properties:

FileName HistoryList IniFile

NOSC provides the following published properties:

AutoOpen AutoNew FormCapOption IniFileName
KeepIniLocal MaxHistory MenuAfter MenuNew
MenuOpen MenuSave MenuSaveAs SaveFormPos
Separators UntitledString

The following published properties operate directly on the Open and Save Dialogs:

osDefaultExt osFilter osFilterIndex osOpenEditStyle
osSaveEditStyle osOpenOptions osSaveOptions osOpenTitle
osSaveTitle osOpenHelpCtx osSaveHelpCtx

osOpenEditStyle, osSaveEditStyle

See Also

Open and Save Dialogs Properties History List

The **osOpenEditStyle** and **osSaveEditStyle** properties make visible the **FileEditStyle** property of the OpenDialog and SaveDialog components and can be set the same way (fsEdit or fsComboBox).

If set to **fsComboBox,** and you have implemented **NOSC**'s History List feature, the combo box of the corresponding Dialog will automatically be filled with the file names from the History List.

Events

See Also:

Properties Tasks

NOSC events do not translate one to one with the OnClick events of your menu items. When a user clicks one of the linked menu items (eg; the Open menu item), NOSC will generate up to three separate events.

The purpose of breaking a menu event into multiple events is to present you with simple, discrete events that are easily dealt with. Within the context of each of the events, you do not need to worry about any of the related complexities associated with a menu event.

For example, when the user clicks on File|Open, NOSC will first issue an **OnSaveQuery** event. You simply have to tell NOSC whether the current file has changes that need to be saved or not. If there are changes are to be saved, NOSC will check whether the current file has been named and, if not, will display the **Save Dialog** for the user. If a file name is selected here, you will then receive an **OnSave** event in order to save the file changes to disk. The user will then be presented with the **Open Dialog** and, if a file is selected, an **OnOpen** event is issued so you can open and display the selected file.

The result of this process is that each of your NOSC event handlers will be very simple and easy to code. Within each of the events, you do not need to care which menu selection caused the event or what other events are involved.

OnNew OnOpen OnSave
OnSaveAs OnSaveQuery OnCloseQuery

- Attaching Events to NOSC
- Responding to NOSC events

Attaching Events to NOSC

See Also:
Responding to NOSC events

In order to perform, NOSC must be given access to the menu items you design for creating, opening and saving files. These, typically, are the New, Open, Save, and Save As menu items which are, also typically, part of a top-level menu item called File. The actual names do not matter nor does their position in your particular menu structure.

NOSC has four corresponding properties that display in the Object Inspector. These are MenuNew, MenuOpen, MenuSave, and MenuSaveAs. Each of these properties will display all the menu items currently defined in your form making it an easy task to assign the corresponding ones.

Although most applications will contain all four of these menu items, it isn't necessary. For example, if your application doesn't allow new files to be created, then you'll simply leave the MenuNew property unassigned. If your application is a read-only viewer, then you'll only have an Open menu item to assign. The important point is that whichever of these four standard menu items you have in your application, you must assign them to NOSC.

At runtime, NOSC assigns its own event handlers to the OnClick event of these menu items. This allows NOSC to handle these events for you. In turn, NOSC re-issues its own events to you, but in a modified and more controlled form (see Responding to NOSC events).

NOSC also, automatically at runtime, assigns its own event handler to your form's OnCloseQuery event. This is necessary in order to allow NOSC to know when the application is about to close. This event is re-issued to you directly, so you can still use it as you would if youreceived it directly from your form.

Important: Because NOSC assigns itself to the OnClick methods of these particular menu items, you must remember not to try creating event handlers for them in your form. Any code you define for these events will be ignored during execution You must, instead, respond to the events issued by NOSC. This also applies to the OnCloseQuery event.

Responding to NOSC events

See Also:
Attaching Events to NOSC

Responding to NOSC events is much simpler than responding to the corresponding "raw" events. This is the main purpose of NOSC; to do as much of the work as possible for you before passing the events on to you. You are still responsible for the actual management of the files handled in your application as NOSC has no knowledge of what these may be.

There are six NOSC events. These correspond to the OnClick methods for the four menu items you assign to NOSC; the form's OnCloseQuery event; and one unique NOSC event (see Attaching Events to NOSC).

The six events are: OnNew, OnOpen, OnSave, OnSaveAs, OnSaveQuery, and OnCloseQuery.

The key to remember when responding to these events is that you no longer need to be concerned with any other aspect of your program except that specific event. For example, when you respond to the OnSave event, NOSC has already ensured that a valid file name has been selected, so you need only save the file. When you respond to an OnNew event, you do not need to be concerned with whether changes to the current file have been saved yet, you will have already received an OnSave event for the current file

All of the events have two parameters. The **OnCloseQuery** event is identical to your form's OnCloseQuery event. The other five events all contain two parameters: the first is the standard Sender parameter. If you need to see where this event originated from, you can check Sender. For example, if the user selected New, in the **OnSaveQuery** event you will find that Sender is your TMenuItem component corresponding to the New menu item; the second parameter is called **Proceed**. In all cases, the **Proceed** parameter defaults to True. If necessary, you can set this to False in order to modify subsequent NOSC behavior for the current chain of events. Details of what this effect is depends on the particular event.

Open and Save Dialogs

NOSC automatically instantiates and manages a TOpenDialog and TSaveDialog component. Many of the NOSC properties you can set in the Object Inspector are specifically for these components and are passed on directly to their corresponding properties. In some cases, you will see corresponding properties for each of the dialogs (such as osOpenTitle and osSaveTitle), in other cases, there is one NOSC property that is used for both dialogs (such as osDefaultExt).

You never have to worry about executing these dialogs. NOSC executes them when necessary. As part of managing these dialogs for you, NOSC always ensures they reflect the currently open file name and directory. For example; if the user opens a text file in the Windows directory, the next time the Open or Save dialog appears, it will be initialized to start in the Windows directory and will display the current file name.

The following NOSC properties relate directly to the Open and Save Dialogs:

osDefaultExt osFilter osFilterIndex osOpenOptions
osSaveOptions osOpenTitle osSaveTitle osOpenHelpCtx
osSaveHelpCtx osOpenEditStyle osSaveEditStyle

HistoryList

See Also:

MaxHistory MenuAfter IniFile IniFileName

The **HistoryList** property is runtime and read-only. It is of type TStringList and holds the most recent opened file names up to the maximum specified in the MaxHistory property. NOSC maintains this list and uses it as the source for building the history menu. It is written to and loaded from the application's initialization (INI) file.

You can easily retrieve the file names from HistoryList using the same methods as any TStringList object.

OnSaveQuery

See Also:
OnNew OnOpen OnSave OnSaveAs OnCloseQuery Open and Save Dialogs

Whenever the user attempts to create or open a file, or close the application, NOSC will issue an **OnSaveQuery** event. The purpose of this event is for you to let NOSC know whether the file is to be saved or not. The FileName property will contain the name of the currently open file, or the UntitledString if the file hasn't yet been named. Typically, you will check whether the file has unsaved changes and, if so, will inform the user, allowing the user to decide whether to save or not.

The **Proceed** parameter defaults to True, meaning you will subsequently receive an **OnSave** event so you can perform the actual save. If the current file is untitled, the user will be presented with the Save Dialog before you are issued the **OnSave** event.

Setting the Proceed parameter to False tells NOSC that no save is necessary and will carry on with the user's request.

Example:

This example checks for modifications to the text file. If there are any, the Proceed parameter is set to the result of asking the user, otherwise it is set to False.

```
procedure TForm1.NOSC1SaveQuery(Sender: TObject; var Proceed: Boolean);
begin
  if Memo1.Modified then
    Proceed := MessageDlg(NOSC1.FileName + ' has changed. Save?',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes
  else
    Proceed := False;
end;
```