Dave's Development Blog

Software Development using Borland / Codegear / Embarcadero RAD Studio

Chapter 8: Editor Notifiers

By David | August 19, 2011 0 Comment

This time around I'm going to look at editor notifiers so that you can get notifications of when editors have changed and a few other useful things.

The implementation below I'm going to show will only work in Delphi 2005 and above. I will write some thing at the end on how you can work around this for earlier version of Delphi in the same manner that I've done for my Browse And Doc It expert.

This implementation is a little different from the previous ones I've described as its a native IDE interface signified by the N in the name rather than the O, however we declare the notifier class in exactly the same way as before as shown below:

```
  TEditorNotifier = Class(TNotifierObject, INTAEditServicesNotifier)
  Strict Private
  Strict Protected
  Public
    procedure WindowShow(const EditWindow: INTAEditWindow; Show, LoadedFromDesktop:
Boolean);
    procedure WindowNotification(const EditWindow: INTAEditWindow; Operation: TOperation);
    procedure WindowActivated(const EditWindow: INTAEditWindow);
    procedure WindowCommand(const EditWindow: INTAEditWindow; Command, Param: Integer;
var Handled: Boolean);
    procedure EditorViewActivated(const EditWindow: INTAEditWindow; const EditView:
IOTAEditView);
    procedure EditorViewModified(const EditWindow: INTAEditWindow; const EditView:
IOTAEditView);
    procedure DockFormVisibleChanged(const EditWindow: INTAEditWindow; DockForm:
TDockableForm);
    procedure DockFormUpdated(const EditWindow: INTAEditWindow; DockForm: TDockableForm);
    procedure DockFormRefresh(const EditWindow: INTAEditWindow; DockForm: TDockableForm);
    Constructor Create;
    Destructor Destroy; Override;
  End;
```

Now that we've defined the class we need to tell the IDE how to use the notifier. If you haven't already looked at the Chapter 1: Starting an Open Tools API Project in this series you may wish to to understand the following code.

The below code registers the notifier with the IDE returning an integer value which is needed to remove the notifier from the IDE at the end of the session.

```
Function InitWizard(Const BorlandIDEServices : IBorlandIDEServices; RegisterProc :
TWizardRegisterProc; var Terminate: TWizardTerminateProc) : Boolean; StdCall;

Begin
  Application.Handle := Application.MainForm.Handle;
  Result := BorlandIDEServices <> Nil;
  If Result Then
    Begin
      ...
      iEditorIndex := (BorlandIDEServices As IOTAEditorServices).AddNotifier(
        TEditorNotifier.Create);
      ...
    End;
End;
```

Finally the below code removes the notifier from the IDE with the previously obtained integer value:

```
  ...
Initialization
  ...
Finalization
  ...
  (BorlandIDEServices As IOTAEditorServices).RemoveNotifier(iEditorIndex);
  ...
End.
```

There are a lot of methods to this notifier that are fired by the IDE for different situations. I'm going to list each methods and list below it the situation in which the IDE fires the method and what the parameters that are provide contain.

EditorViewActivated

```
procedure TEditorNotifier.EditorViewActivated(const EditWindow: INTAEditWindow; const
EditView: IOTAEditView);
```

This method is fired each time a tab is changed in the editor whether that's through opening and closing files or simply changing tabs to view a different file. The `EditWindow` parameter provides access to the editor window. This is usually the first docked editor window unless you've opened a new editor window to have a second one visible. The `EditView` parameter provides you with access to the view of the file where you can get information about the cursor positions, the selected block, etc. By drilling down through the `Buffer` property you can get the text associated with the file but that's for another chapter, then next one I think.

EditorViewModified

```
procedure TEditorNotifier.EditorViewModified(const EditWindow: INTAEditWindow; const
EditView: IOTAEditView);
```

This method is fired each time the text of the file is changed whether that is an insertion or a deletion of text. The values returned by the parameters as the same as those for the above `EditorViewActivated` method.

WindowActivated

```
procedure TEditorNotifier.WindowActivated(const EditWindow: INTAEditWindow);
```

Well I've been unable to get this to fire in both a docked layout and a classic undocked layout, so if someone else knows what fires this, please let me know.

WindowCommand

```
procedure TEditorNotifier.WindowCommand(const EditWindow: INTAEditWindow; Command, Param:
Integer; var Handled: Boolean);
```

This method is fired for some keyboard commands but there doesn't seem to be any logic to when it is fired or for what. The `Command` parameter is the command number and in all my tests the `Param` parameter was 0. I've check against keyboard binding and have found that this event is not fired for OTA keyboard binding.

WindowNotification

```
procedure TEditorNotifier.WindowNotification(const EditWindow: INTAEditWindow; Operation:
TOperation);
```

This event is fired for each editor window that is opened or closed. The `EditWindow` parameter is a reference to the specific editor window opening or closing and the `Operation` parameter depicts whether the editor is opening (insert) or closing (remove).

WindowShow

```
procedure TEditorNotifier.WindowShow(const EditWindow: INTAEditWindow; Show,
LoadedFromDesktop: Boolean);
```

This event is fired each time an editor window appears or disappear. The `EditWindow` parameter references the editor changing appearance with the `Show` parameter defining whether it is appearing (`show` = true) or disppearing (`show` = false). The `LoadFromDesktop` parameter defines whether the operation is being caused by a desktop layout being loaded or not.

DockFormRefresh

```
procedure TEditorNotifier.DockFormRefresh(const EditWindow: INTAEditWindow; DockForm:
TDockableForm);
```

This method seems to be fired when the IDE is closing down and the desktop of being save. I’ve not been able to get the event to fire for any other situations. The `EditWindow` is the edit window that the docking operation is be docked to (its a dock site) and `DockForm` is the form that is being docked.

DockFormUpdated

```
procedure TEditorNotifier.DockFormUpdated(const EditWindow: INTAEditWindow; DockForm:
TDockableForm);
```

This event seems to be fired when a dockable form is docked with an Edit Window dock site. The parameters are the same as those for the above `DockFormRefresh`.

DockFormVisibleChanged

```
procedure TEditorNotifier.DockFormVisibleChanged(const EditWindow: INTAEditWindow;
DockForm: TDockableForm);
```

This method seems to be fired when desktops are loaded and not as I thought when dockable forms change their visibility. The parameters are the same as those for the above `DockFormRefresh`.

With just the `EditorViewActivated` and `EditorViewModified` we can understand what editor files are being shown and display information based on that. We can also know when a file has been updated (changed) so that the information can be updated. This is how the Browse and Doc It expert works and can display the explorer view of the code in the files.

For earlier IDEs we have to do something else. What we have to do is set up a timer and look for changes to the active file in the editor (see `ActiveProject` and `ActiveSourceEditor` in Chapter 5: Useful Open Tools Utility Functions) and react to those changes when the project or file change. To detect the modification of the file itself, then we need to monitor the size of the edit buffer each time we check and look for changes. You can see all of this in the source for to the Browse And Doc It expert.

Dave.

Category: Open Tools API Tags: Borland, BorlandIDEServices, CodeGear, Delphi, Embarcadero, Experts, INTAEditServicesNotifier, IOTAEditorServices, IOTAEditView, IOTAWizard, OTA, RAD Studio