Dave's Development Blog

Software Development using Borland / Codegear / Embarcadero RAD Studio

Chapter 11: Writing editor code

By David | September 1, 2011 0 Comment

In this chapter I'm going to extend the ideas from Chapter 10: Reading editor code and make the code insert a comment block above the method declaration and move the cursor to the start of the description in the comment.

To do this I've modified the **SelectMethod** as below:

```
Procedure SelectMethod;

Var
  slItems: TStringList;
  SE: IOTASourceEditor;
  CP: TOTAEditPos;
  iIndex: Integer;

Begin
  slItems := TStringList.Create;
  Try
    GetMethods(slItems);
    iIndex := TfrmItemSelectionForm.Execute(slItems, 'Select Method');
    If iIndex > -1 Then
      Begin
        CP := InsertComment(slItems, iIndex);
        SE := ActiveSourceEditor;
        If SE <> Nil Then
          Begin
            SE.EditViews[0].CursorPos := CP;
            SE.EditViews[0].Center(CP.Line, CP.Col);
          End;
      End;
  Finally
    slItems.Free;
  End;
End;
```

Note that the **InsertComment** method (detailed in a minute) returns a new cursor position which we then update. The code that got the cursor position from our user-defined record has been moved into the start of the `InsertComment` method as below:

```
Function InsertComment(slItems : TStringList; iIndex : Integer) : TOTAEditPos;

Var
  recItemPos : TItemPosition;
  SE: IOTASourceEditor;
  Writer: IOTAEditWriter;
  i: Integer;
  iIndent: Integer;
  iPosition: Integer;
  CharPos : TOTACharPos;

Begin
  recItemPos.Data := slItems.Objects[iIndex];
  Result.Line := recItemPos.Line;
  Result.Col := 1;
  SE := ActiveSourceEditor;
  If SE <> Nil Then
    Begin
      Writer := SE.CreateUndoableWriter;
      Try
        iIndent := 0;
        For i := 1 To Length(slItems[iIndex]) Do
          If slItems[iIndex][i] = #32 Then
            Inc(iIndent)
          Else
            Break;
        CharPos.Line := Result.Line;
        CharPos.CharIndex := Result.Col - 1;
        iPosition := SE.EditViews[0].CharPosToPos(CharPos);
        Writer.CopyTo(iPosition);
        OutputText(Writer, iIndent, '(**'#13#10);
        OutputText(Writer, iIndent, #13#10);
        OutputText(Writer, iIndent, '  Description.'#13#10);
        OutputText(Writer, iIndent, #13#10);
        OutputText(Writer, iIndent, '  @precon  '#13#10);
        OutputText(Writer, iIndent, '  @postcon '#13#10);
        OutputText(Writer, iIndent, #13#10);
        OutputText(Writer, iIndent, '**)'#13#10);
        Inc(Result.Line, 2);
        Inc(Result.Col, iIndent + 2);
      Finally
        Writer := Nil;
      End;
    End;
End;
```

In this method the following happens:

1. Update the `Result` of the function with the current method cursor position;
2. Next get a reference to the active source editor as described in the previous chapter;
3. If the source editor is valid obtain a undoable writer interface. The IDE supports both an undoable interface and one without that can't be undone. I prefer to allow the user to undo any changes so always use the `CreateUndoableWriter` method.
4. Next we count the number of space at the start of the method line to see if we need to indent the comment (tabs are not supported in this example);
5. Then we need to convert the cursor position to a position index into the writer buffer using the `EditViews.CharPosToPos` method. One thing to note is that the `CharPosToPos` function uses a `TOTACharPos` record not a `TOTAEditPos` record, so we need to convert. In converting the char index in the `TOTACharPos` record is 0 based NOT 1 based therefore we need to subtract 1;
6. Once we have the position in the writer buffer of the cursor position we use the `CopyTo` method of the writer to move to that position;
7. We then output the lines of the comment using a new utility function OutputText which is described below;
8. Finally we update the `Result` cursor position for the new position in the inserted comment text.

The reason for refactoring the `OutputText` functionality out is that this is different for Delphi 2009 (Unicode) and above so casting is required to stop compiler warnings appearing and it also makes the code cleaner.

```
Procedure OutputText(Writer : IOTAEditWriter; iIndent : Integer; strText : String);

Begin
  {$IFNDEF D2009}
  Writer.Insert(PAnsiChar(StringOfChar(#32, iIndent) + strText));
  {$ELSE}
  Writer.Insert(PAnsiChar(AnsiString(StringOfChar(#32, iIndent) + strText)));
  {$ENDIF}
End;
```

There is a caveat to working with the `IOTAEditWriter` interface in that it is a move forward only buffer – you can not move backwards in the buffer. This causes a problem with multiple inserts as generally to maintain the cursor position you would want to move backwards through the text so you don't have to update all you cursor positions. The way I get round this is to perform the insert at each location (going backwards) with new instances of the `IOTAEditWriter` interface.

Here are the files to accompany this (OTA Chapter 11).

Dave.

Category: Open Tools API Tags: Borland, BorlandIDEServices, CodeGear, Delphi, Experts, IOTAEditReader, IOTAEditWriter, IOTASourceEditor, OTA, RAD Studio, TOTACharPos, TOTAEditPos

Iconic One Theme | Powered by Wordpress