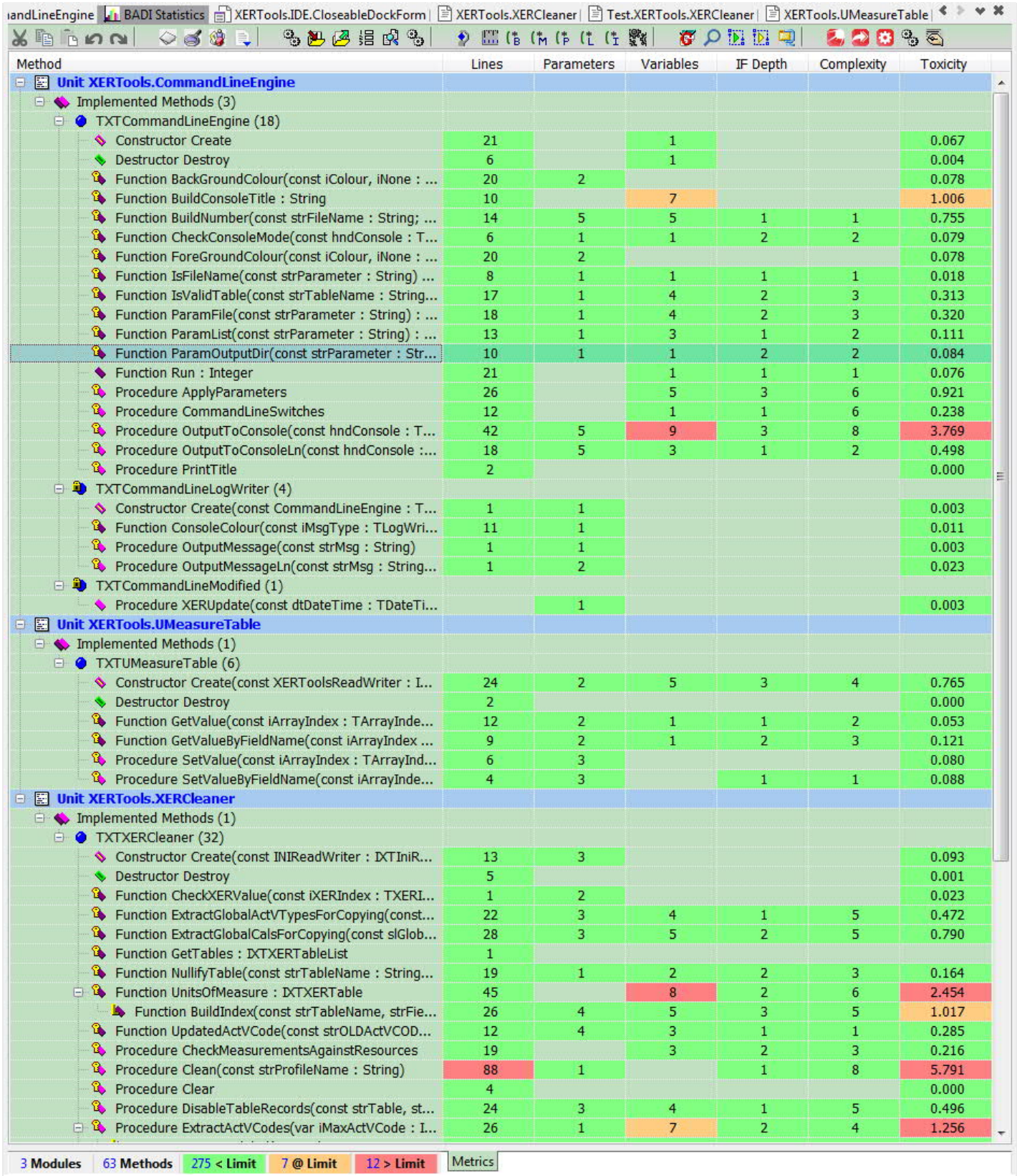
Dave's Development Blog
Software Development using Borland / Codegear / Embarcadero RAD Studio

Editor Views in RAD Studio

By David | June 5, 2018 0 Comment

Overview

Well, this was the blog I tried to write before Xmas before I found it didn't quite work in all circumstances. So here I'll describe how to create a custom editor view in RAD Studio along with editor status bar panels. These editor views are full editor tabs not sub-tabs like the sub-views I described before, therefore they are not associated with a specific module and you have to provide the contents of the editor view via a frame. The example below is from the currently unreleased Browse and Doc It plug and it provides a treeview of metrics for all modules (units, forms, etc) that are in the active project and highlights those that are above the limits set.



Method	Lines	Parameters	Variables	IF Depth	Complexity	Toxicity
Unit XERTools.CommandLineEngine						
Implemented Methods (3)						
TXTCommandLineEngine (18)						
Constructor Create	21		1			0.067
Destructor Destroy	6		1			0.004
Function BackGroundColour(const iColour, iNone : ...	20	2				0.078
Function BuildConsoleTitle : String	10		7			1.006
Function BuildNumber(const strFileName : String; ...	14	5	5	1	1	0.755
Function CheckConsoleMode(const hndConsole : T...	6	1	1	2	2	0.079
Function ForeGroundColour(const iColour, iNone : ...	20	2				0.078
Function IsFileName(const strParameter : String) ...	8	1	1	1	1	0.018
Function IsValidTable(const strTableName : String...	17	1	4	2	3	0.313
Function ParamFile(const strParameter : String) : ...	18	1	4	2	3	0.320
Function ParamList(const strParameter : String) : ...	13	1	3	1	2	0.111
Function ParamOutputDir(const strParameter : Str...	10	1	1	2	2	0.084
Function Run : Integer	21		1	1	1	0.076
Procedure ApplyParameters	26		5	3	6	0.921
Procedure CommandLineSwitches	12		1	1	6	0.238
Procedure OutputToConsole(const hndConsole : T...	42	5	9	3	8	3.769
Procedure OutputToConsoleLn(const hndConsole :...	18	5	3	1	2	0.498
Procedure PrintTitle	2					0.000
TXTCommandLineLogWriter (4)						
Constructor Create(const CommandLineEngine : T...	1	1				0.003
Function ConsoleColour(const iMsgType : TLogWri...	11	1				0.011
Procedure OutputMessage(const strMsg : String)	1	1				0.003
Procedure OutputMessageLn(const strMsg : String...	1	2				0.023
TXTCommandLineModified (1)						
Procedure XERUpdate(const dtDateTime : TDateTi...		1				0.003
Unit XERTools.UMeasureTable						
Implemented Methods (1)						
TXTUMeasureTable (6)						
Constructor Create(const XERToolsReadWriter : I...	24	2	5	3	4	0.765
Destructor Destroy	2					0.000
Function GetValue(const iArrayIndex : TArrayInde...	12	2	1	1	2	0.053
Function GetValueByFieldName(const iArrayIndex ...	9	2	1	2	3	0.121
Procedure SetValue(const iArrayIndex : TArrayInd...	6	3				0.080
Procedure SetValueByFieldName(const iArrayInde...	4	3		1	1	0.088
Unit XERTools.XERCleaner						
Implemented Methods (1)						
TXTXERCleaner (32)						
Constructor Create(const INIReadWriter : IXTIniR...	13	3				0.093
Destructor Destroy	5					0.001
Function CheckXERValue(const iXERIndex : TXERI...	1	2				0.023
Function ExtractGlobalActVTypesForCopying(const...	22	3	4	1	5	0.472
Function ExtractGlobalCalsForCopying(const slGlob...	28	3	5	2	5	0.790
Function GetTables : IXTXERTableList	1					
Function NullifyTable(const strTableName : String...	19	1	2	2	3	0.164
Function UnitsOfMeasure : IXTXERTable	45		8	2	6	2.454
Function BuildIndex(const strTableName, strFie...	26	4	5	3	5	1.017
Function UpdatedActVCode(const strOLDActVCOD...	12	4	3	1	1	0.285
Procedure CheckMeasurementsAgainstResources	19		3	2	3	0.216
Procedure Clean(const strProfileName : String)	88	1		1	8	5.791
Procedure Clear	4					0.000
Procedure DisableTableRecords(const strTable, st...	24	3	4	1	5	0.496
Procedure ExtractActVCodes(var iMaxActVCode : I...	26	1	7	2	4	1.256

I'm only going to describe the class you need to create for the editor view and how to tell the IDE about it. The frame that is used will only be referred to as we look at some of the methods.

Definition

Below is the definition of the class which implements a number of Open Tools API interfaces. The definition at

first might appear a little more complicated than it needs to be and that is because I have embedded some other classes in this class which manage information. I will go through those where they apply.

```
Type
  TBADIModuleMetricsEditorView = Class(TInterfacedObject, INTACustomEditorView,
INTACustomEditorView150, INTACustomEditorViewStatusPanel)
  Strict Private
    Type
      TBADIFileInfoManager = Class
        ...
      End;
      TBADIMetricStatusPanel = (mspModules, mspMethods, mspLinesOfCode, mspUnderLimit,
mspAtLimit, mspOverLimit);
      TBADIFrameManager = Class
      Strict Private
        Type
          TBADIFrameManagerRecord  = Record
            FEditWindowName : String;
            FFrameReference : TframeBADIModuleMetricsEditorView;
          End;
      Strict Private
        FFrames : TList;
      Strict Protected
        Function  GetFrame(Const strEditWindowName : String) :
TframeBADIModuleMetricsEditorView;
        Function  Find(Const strEditWindowName :String) : Integer;
      Public
        Constructor Create;
        Destructor Destroy; Override;
        Procedure Add(Const strEditWindowName : String; Const AFrame :
TframeBADIModuleMetricsEditorView);
        Property Frame[Const strEditWindowName : String] :
TframeBADIModuleMetricsEditorView Read GetFrame;
      End;
    Class Var
      FEditorViewRef : INTACustomEditorView;
  Strict Private
    FFrameManager     : TBADIFrameManager;
    FFileInfoMgr      : TBADIFileInfoManager;
    FImageIndex       : Integer;
    FViewIdent        : String;
    FModulePanels     : Array[Low(TBADIMetricStatusPanel)..High(TBADIMetricStatusPanel)]
Of TStatusPanel;
    FCount            : Integer;
    FSourceStrings    : TStringList;
```

```
    FSource           : String;
    FFileName         : String;
    FModified         : Boolean;
    FFileDate         : TDateTime;
    FLastRenderedList : TBADIModuleMetrics;
  Strict Protected
    // INTACustomEditorView
    Function  CloneEditorView: INTACustomEditorView;
    Procedure CloseAllCalled(Var ShouldClose: Boolean);
    Procedure DeselectView;
    Function  EditAction(Action: TEditAction): Boolean;
    Procedure FrameCreated(AFrame: TCustomFrame);
    Function  GetCanCloneView: Boolean;
    Function  GetCaption: String;
    Function  GetEditState: TEditState;
    Function  GetEditorWindowCaption: String;
    Function  GetFrameClass: TCustomFrameClass;
    Function  GetViewIdentifier: String;
    Procedure SelectView;
    // INTACustomEditorView150
    Procedure Close(Var Allowed: Boolean);
    Function  GetImageIndex: Integer;
    Function  GetTabHintText: String;
    // INTACustomEditorViewStatusPanel
    Procedure ConfigurePanel(StatusBar: TStatusBar; Panel: TStatusPanel);
    Procedure DrawPanel(StatusBar: TStatusBar; Panel: TStatusPanel; Const Rect: TRect);
    Function  GetStatusPanelCount: Integer;
    // General Methods
    Procedure ParseAndRender;
    Procedure UpdateStatusPanels;
    Procedure ExtractSourceFromModule(Const Module : IOTAModule);
    Procedure ExtractSourceFromFile;
    Procedure LastModifiedDateFromModule(Const Module: IOTAModule);
    Procedure LastModifiedDateFromFile(Const ModuleInfo: IOTAModuleInfo);
    Function  CurrentEditWindow : String;
    Procedure ProcesModule(Const ModuleInfo : IOTAModuleInfo);
    Function  RenderedList : TBADIModuleMetrics;
  Public
    Class Function CreateEditorView: INTACustomEditorView;
    Constructor Create(Const strViewIdentifier : String);
    Destructor Destroy; Override;
  End;
```

Interfaces

The following interfaces are implemented in the above class as follows:

INTACustomEditorView

This interface is the main interface which you need to implement in order to provide a custom editor view. You should note that this is a native interface (the `N` in `INTAXxxx`) and as such it is accessing some of the internals of the specific version of RAD Studio. What this means is that in order to use this interface you need to create specific versions of your plug-in for each RAD Studio IDE as the interface is version specific and will likely crash other versions.

INTACustomEditorView150

This interface extends the above interface by added the ability to provide hints and icons on the editor tabs.

INTACustomEditorViewStatusPanel

This last interface is not required for implementing a custom editor view however I've implemented it to show you how to create editor status panels that are specific to your custom editor view.

Inner Classes

I've embedded a number of types in the class as they are not required outside of the custom editor view. These may end up being pulled out into separate units as they may be useful in other editor views (code checks for instance).

TBADIFileInfoManager

This first class is a wrapper around a simple generic record and is used to store module filenames against their last modified update. The reason I've done this is that when the custom editor view regains focus it checks which modules might be updated and refreshes only those metrics in the treeview. I'm not going to describe the implementation as I'm sure that this is straightforward for everyone.

TBADIMetricStatusPanel

This is an enumerate to define the panels in the statusbar. This just makes the code more readable in terms of what each panel contains.

TBADIFrameManager

This class requires a little more explanation. I found that there is no way in the IDE to track custom editor views in different editor windows. This was the issue that prevented me from writing this blog before. Hopefully you know that you can have more than one editor window open in the IDE. In these cases you need an editor view for each and its keeping track of these that requires this class. I'm not going to go through the implementation as its straight forward however the class keeps track of the frame reference against the editor window name.

Fields

Below is a brief explanation of the fields I've defined in the custom editor view class.

FFrameManager : TBADIFrameManager

This is a reference to an instance of the frame manager described above.

FFileInfoMgr : TBADIFileInfoManager

This is a reference to the file information manager described above.

FImageIndex : Integer

This is the image index of the image we need to add the IDE's image list that will be displayed next to the editor tab.

FViewIdent : String

This is the name of the view which is passed in the class's constructor.

FModulePanels : Array[Low(TBADIMetricStatusPanel)..High(TBADIMetricStatusPanel)] Of TStatusPanel

This field is an array which holds references to each of the statusbar panels we want to maintain with the view.

FCount : Integer

This is a counter which is used in the editor view caption. See `GetCaption` for more details on why this is required.

FSourceStrings : TStringList

This field is used to load the source text from a disk file. Its created once in the constructor and free in the destructor to try and improve performance.

FSource : String

This field is used to hold the module source code for the module that is about to be parsed.

FFileName : String

This field holds the current filename being parsed.

FModified : Boolean

This field holds a boolean denoting whether the current file being parsed was modified.

FFileDate : TDateTime

This field holds the date and time of the current file being parsed

FLastRenderedList : TBADIModuleMetrics

This field holds a list of the metric options being rendered so that if the options are changed the list can be re-rendered even if the code has not changed.

Constants

Below are a few constants that are used within the code.

```
Const
```

```
  strBADIMetricsEditorView = 'BADIMetricsEditorView';
  strBADIMetrics = 'BADI Metrics';
  strUnknown = 'Unknown';
```

Implementation

Next I'll go through the implementation of the methods in the class.

Functions

This method returns an instance of the custom editor view and is passed in the registration of the view so that a view can be created when a desktop is loaded.

```
Function RecreateBADIStatisticEditorView: INTACustomEditorView;

Begin
  Result := TBADIModuleMetricsEditorView.CreateEditorView;
End;
```

Interfaces Methods

INTACustomEditorView

Function CloneEditorView: INTACustomEditorView

This method is called when the IDE wants to clone the view and if the `GetCanClose` method returns true.

You should return a cloned instance of your view if requested. I have not been able to get the IDE to ever call this method.

```
Function TBADIModuleMetricsEditorView.CloneEditorView: INTACustomEditorView;

Var
  EVS : IOTAEditorViewServices;

Begin
  If Supports(BorlandIDEServices, IOTAEditorViewServices, EVS) Then
    EVS.CloseActiveEditorView;
  Result := RecreateBADIStatisticEditorView;
End;
```

Procedure CloseAllCalled(Var ShouldClose: Boolean)

This method is called when all the views in the editor are being requested to close. Return `true` to allow it to close else return `false` for it to persist.

I return `true` here so it can be closed.

```
Procedure TBADIModuleMetricsEditorView.CloseAllCalled(Var ShouldClose: Boolean);
```

```
Begin
  ShouldClose := True;
End;
```

Procedure DeselectView

This method is called when the editor view loses focus.

I don't do anything here but you may want to do some processing here to store any state information for your view.

```
Procedure TBADIModuleMetricsEditorView.DeselectView;

Begin
  // Does nothing
End;
```

Function EditAction(Action: TEditAction): Boolean

This method is called for the given editor action that you have said is supported by the editor view (the return of GetEditState).

I have only implemented copy, so the treeview text is copied to the clipboard if that action is invoked.

```
Function TBADIModuleMetricsEditorView.EditAction(Action: TEditAction): Boolean;

Var
  AFrame: TframeBADIModuleMetricsEditorView;

Begin
  Result := False;
  Case Action Of
    eaCopy:
      Begin
        AFrame := FFrameManager.Frame[CurrentEditWindow];
        If Assigned(AFrame) Then
          AFrame.CopyToClipboard;
        Result := True;
      End;
  End;
End;
```

Procedure FrameCreated(AFrame: TCustomFrame)

This method is called when the frame is first created.

The method stores a reference to the frame so that a module metrics frame can be rendered

```
Procedure TBADIModuleMetricsEditorView.FrameCreated(AFrame: TCustomFrame);
```

```
Const
  strTEditWindow = 'TEditWindow';

Var
  ES : INTAEditorServices;
  C : TWinControl;
  strEditWindowName : String;

Begin
  FFileInfoMgr.Clear;
  If Supports(BorlandIDEServices, INTAEditorServices, ES) Then
    Begin
      strEditWindowName := strUnknown;
      C := AFrame;
      While Assigned(C) Do
        Begin
          If C.ClassName = strTEditWindow Then
            Begin
              strEditWindowName := C.Name;
              Break;
            End;
          C := C.Parent;
        End;
      FFrameManager.Add(strEditWindowName, AFrame As TframeBADIModuleMetricsEditorView);
    End;
End;
```

Function GetCanCloneView: Boolean

This is a getter method for the `CanCloseView` property.

Returns `false` as this editor view should not be cloned (think singleton view).

```
Function TBADIModuleMetricsEditorView.GetCanCloneView: Boolean;

Begin
  Result := False;
End;
```

Function GetCaption: String

This is a getter method for the `Caption` property.

The method returns the caption for the editor view. It is also used as the editor sub view tab description. I found that this occurred on separate calls so by looking at the even or odd calls you can name the editor sub-view tab differently than the editor tab.

```
Function TBADIModuleMetricsEditorView.GetCaption: String;

ResourceString
  strMetrics = 'Metrics';

Const
  iDivisor = 2;

Begin
  Inc(FCount);
  If FCount Mod iDivisor = 0 Then
    Result := strMetrics
  Else
    Result := strBADIMetrics;
End;
```

Function GetEditState: TEditState

This is a getter method for the `EditState` property.

This method is called to tell the IDE what editor states can be invoked on the data in the view (cut, copy, paste, etc). I only want to be able to copy the treeview text.

```
Function TBADIModuleMetricsEditorView.GetEditState: TEditState;

Begin
  Result := [esCanCopy];
End;
```

Function GetEditorWindowCaption: String

This is a getter method for the `EditorWindowCaption` property.

Returns the text to be displayed in the Editor Window (you can only see this when the editor is floating).

```
Function TBADIModuleMetricsEditorView.GetEditorWindowCaption: String;

Begin
  Result := strBADIMetrics;
End;
```

Function GetFrameClass: TCustomFrameClass

This is a getter method for the `FrameClass` property.

The method returns the frame class that the IDE should create when creating the editor view (you don't create this yourself).

```
Function TBADIModuleMetricsEditorView.GetFrameClass: TCustomFrameClass;
```

```
Begin
  Result := TframeBADIModuleMetricsEditorView;
End;
```

Function GetViewIdentifier: String

This is a getter method for the `ViewIdentifer` property.

This returns a unique identifier for this view (must be unique within the IDE – think singleton instance).

```
Function TBADIModuleMetricsEditorView.GetViewIdentifier: String;

Begin
  Result := Format('%s.%s', [strBADIMetricsEditorView, FViewIdent]);
End;
```

Procedure SelectView;

This method is called when the editor view is selected, either when it's created or when it regains focus.

This method renders the module metrics in the frame.

```
Procedure TBADIModuleMetricsEditorView.SelectView;

ResourceString
  strParsingProjectModules = 'Parsing project modules';
  strPleaseWait = 'Please wait...';
  strParsing = 'Parsing: %s...';

Const
  setModuleTypesToParse = [omtForm, omtDataModule, omtProjUnit, omtUnit];

Var
  P: IOTAProject;
  iModule: Integer;
  frmProgress : TfrmProgress;
  ModuleInfo: IOTAModuleInfo;
  AFrame: TframeBADIModuleMetricsEditorView;

Begin
  P := ActiveProject;
  If Assigned(P) Then
    Begin
      If FLastRenderedList <> RenderedList Then
        FFileInfoMgr.Clear;
      FLastRenderedList := RenderedList;
```

```
      frmProgress := TfrmProgress.Create(Application.MainForm);
      Try
        frmProgress.Init(P.GetModuleCount, strParsingProjectModules, strPleaseWait);
        For iModule := 0 To P.GetModuleCount - 1 Do
          Begin
            ModuleInfo := P.GetModule(iModule);
            If ModuleInfo.ModuleType In setModuleTypesToParse Then
              Begin
                ProcesModule(ModuleInfo);
                frmProgress.UpdateProgress(Succ(iModule), Format(strParsing,
[ExtractFileName(FFileName)]));
              End
          End;
        AFrame := FFrameManager.Frame[CurrentEditWindow];
        If Assigned(AFrame) Then
          AFrame.FocusResults;
      Finally
        frmProgress.Free;
      End;
      UpdateStatusPanels;
    End;
End;
```

INTACustomEditorView150

Procedure Close(Var Allowed: Boolean)

This method is called when this view tab in the editor is being requested to close. Return `true` to allow it to close else return `false` for it to persist.

I return `true` here so it can be closed.

```
Procedure TBADIModuleMetricsEditorView.Close(Var Allowed: Boolean);

Begin
  Allowed := True;
End;
```

Function GetImageIndex: Integer

This is a getter method for the `ImageIndex` property.

Returns the image index of the image in the editor image list for this editor view.

```
Function TBADIModuleMetricsEditorView.GetImageIndex: Integer;

Begin
  Result := FImageIndex;
```

```
End;
```

Function GetTabHintText: String

This is a getter method for the `TabHintText` property.

Returns the text to be displayed when the mouse is hovered over the editor tab.

```
Function TBADIModuleMetricsEditorView.GetTabHintText: String;

Begin
  Result := strBADIMetrics;
End;
```

INTACustomEditorViewStatusPanel

Procedure ConfigurePanel(StatusBar: TStatusBar; Panel: TStatusPanel)

This method is called when each editor status panel is created.

References to the panels are stored for later use and each panel is configured. Note: I found a bug here regarding the style of the panel.

```
Procedure TBADIModuleMetricsEditorView.ConfigurePanel(StatusBar: TStatusBar; Panel:
TStatusPanel);

Const
  iPanelWidth = 80;

Begin
  FModulePanels[TBADIMetricStatusPanel(Panel.Index)] := Panel;
  FModulePanels[TBADIMetricStatusPanel(Panel.Index)].Alignment := taCenter;
  FModulePanels[TBADIMetricStatusPanel(Panel.Index)].Width := iPanelWidth;
  // Problems with first panel if you do not explicitly set this
  FModulePanels[TBADIMetricStatusPanel(Panel.Index)].Style := psOwnerDraw; // psText;
End;
```

Procedure DrawPanel(StatusBar: TStatusBar; Panel: TStatusPanel; Const Rect: TRect)

This method is called for each status panel if it is set to owner draw.

Each panel is drawn with a blue number and black bold text (more to demonstrate what you can do then actually needing this).

```
Procedure TBADIModuleMetricsEditorView.DrawPanel(StatusBar: TStatusBar; Panel:
TStatusPanel; Const Rect: TRect);

  Procedure DrawBackground(Const strNum : String; Const StyleServices :
TCustomStyleServices);
```

```
Var
  iColour : TColor;

Begin
  If TBADIMetricStatusPanel(Panel.Index) In [mspModules..mspLinesOfCode] Then
    Begin
      iColour := clBtnFace;
      If Assigned(StyleServices) Then
        iColour := StyleServices.GetSystemColor(clBtnFace);
    End Else
      iColour := iLightGreen;
  If strNum <> '' Then
    Case TBADIMetricStatusPanel(Panel.Index) Of
      mspAtLimit:
        If StrToInt(strNum) > 0 Then
          iColour := iLightAmber;
      mspOverLimit:
        If StrToInt(strNum) > 0 Then
          iColour := iLightRed;
    End;
  StatusBar.Canvas.Brush.Color := iColour;
  StatusBar.Canvas.FillRect(Rect);
End;

Function CalcWidth(Const strNum, strSpace, strText : String) : Integer;

Begin
  StatusBar.Canvas.Font.Style := [];
  Result := StatusBar.Canvas.TextWidth(strNum);
  Inc(Result, StatusBar.Canvas.TextWidth(strSpace));
  StatusBar.Canvas.Font.Style := [fsBold];
  Inc(Result, StatusBar.Canvas.TextWidth(strText));
End;

Procedure DrawText(Var strNum, strSpace, strText : String; Const iWidth : Integer;
  Const StyleServices : TCustomStyleServices);

Const
  iDivisor = 2;

Var
  R : TRect;

Begin
```

```
    R := Rect;
    Inc(R.Left, (R.Right - R.Left - iWidth) Div iDivisor);
    Inc(R.Top);
    StatusBar.Canvas.Font.Color := clBlue; //: @todo Fix when the IDE is fixed.
    StatusBar.Canvas.Font.Style := [];
    StatusBar.Canvas.TextRect(R, strNum, [tfLeft, tfVerticalCenter]);
    Inc(R.Left, StatusBar.Canvas.TextWidth(strNum));
    StatusBar.Canvas.TextRect(R, strSpace, [tfLeft, tfVerticalCenter]);
    StatusBar.Canvas.Font.Color := clWindowText;
    If Assigned(StyleServices) Then
      StatusBar.Canvas.Font.Color := StyleServices.GetSystemColor(clWindowText);
    StatusBar.Canvas.Font.Style := [fsBold];
    Inc(R.Left, StatusBar.Canvas.TextWidth(strSpace));
    StatusBar.Canvas.TextRect(R, strText, [tfLeft, tfVerticalCenter]);
  End;

Var
  strNum, strSpace, strText : String;
  iPos : Integer;
  StyleServices : TCustomStyleServices;
  {$IFDEF DXE102}
  ITS : IOTAIDEThemingServices;
  {$ENDIF}

Begin
  StyleServices := Nil;
  {$IFDEF DXE102}
  If Supports(BorlandIDEServices, IOTAIDEThemingServices, ITS) Then
    If ITS.IDEThemingEnabled Then
      StyleServices := ITS.StyleServices;
  {$ENDIF}
  // Split text by first space
  iPos := Pos(#32, Panel.Text);
  strNum := Copy(Panel.Text, 1, Pred(iPos));
  strSpace := #32;
  strText := Copy(Panel.Text, Succ(iPos), Length(Panel.Text) - iPos);
  DrawBackground(strNum, StyleServices);
  DrawText(strNum, strSpace, strText, CalcWidth(strNum, strSpace, strText),
StyleServices);
End;
```

Function GetStatusPanelCount: Integer;

This is a getter method for the `StatusPanelCount` property.

Returns the number of status panels to create for the editor view.

```
Function TBADIModuleMetricsEditorView.GetStatusPanelCount: Integer;

Begin
  Result := Ord(High(TBADIMetricStatusPanel)) - Ord(Low(TBADIMetricStatusPanel)) + 1;
End;
```

General Methods

Constructor

This is the constructor for the TBADIModuleMetrics class.

This create a number of the classes for managing information and adds an image to the editor image list to be displayed against this editor view.

```
Constructor TBADIModuleMetricsEditorView.Create(Const strViewIdentifier : String);

Const
  strBADIMetricsImage = 'BADIMetricsImage';

Var
  EVS : INTAEditorViewServices;
  ImageList : TImageList;
  BM: TBitmap;

Begin
  Inherited Create;
  FFrameManager := TBADIFrameManager.Create;
  FFileInfoMgr := TBADIFileInfoManager.Create;
  FSourceStrings := TStringList.Create;
  FViewIdent := strViewIdentifier;
  FCount := 0;
  If Supports(BorlandIDEServices, INTAEditorViewServices, EVS) Then
    Begin
      ImageList := TImageList.Create(Nil);
      Try
        BM := TBitMap.Create;
        Try
          BM.LoadFromResourceName(HInstance, strBADIMetricsImage);
          ImageList.AddMasked(BM, clLime);
          FImageIndex := EVS.AddImages(ImageList, strBADIMetricsEditorView);
        Finally
          BM.Free;
        End;
      Finally
        ImageList.Free;
```

```
      End;
    End;
End;
```

Destructor

This is the destructor for the TBADIModuleMetrics class.

It frees the memory used by the module's management classes.

```
Destructor TBADIModuleMetricsEditorView.Destroy;

Begin
  FSourceStrings.Free;
  FFileInfoMgr.Free;
  FFrameManager.Free;
  Inherited Destroy;
End;
```

Class Constructor

This is a class method to create a singleton instance of this editor view.

It create the editor view if it does not already exist else it returned the existing instance reference.

```
Class Function TBADIModuleMetricsEditorView.CreateEditorView : INTACustomEditorView;

Var
  EVS : IOTAEditorViewServices;

Begin
  Result := Nil;
  If Supports(BorlandIDEServices, IOTAEditorViewServices, EVS) Then
    Begin
      If Not Assigned(FEditorViewRef) Then
        FEditorViewRef := TBADIModuleMetricsEditorView.Create('');
      Result := FEditorViewRef;
      EVS.ShowEditorView(Result);
    End;
End;
```

CurrentEditWindow

This method returns the name of the current top level editor window.

```
Function TBADIModuleMetricsEditorView.CurrentEditWindow: String;

Var
```

```
  ES : INTAEditorServices;

Begin
  Result := strUnknown;
  If Supports(BorlandIDEServices, INTAEditorServices, ES) Then
    Result := ES.TopEditWindow.Form.Name;
End;
```

UpdateStatusPanels

This method updates the status panels with the information from the frame, i.e. statistics on the metrics.

```
Procedure TBADIModuleMetricsEditorView.UpdateStatusPanels;

ResourceString
  strModules = '%d Modules';
  strMethods = '%d Methods';
  strLinesOfCode = '%d Lines';
  strUnderLimit = '%d < Limit';
  strAtLimit = '%d @ Limit';
  strOverLimit = '%d > Limit';

Var
  AFrame: TframeBADIModuleMetricsEditorView;

Begin
  AFrame := FFrameManager.Frame[CurrentEditWindow];
  If Assigned(AFrame) Then
    Begin
      FModulePanels[mspModules].Text := Format(strModules, [AFrame.ModuleCount]);
      FModulePanels[mspMethods].Text := Format(strMethods, [AFrame.MethodCount]);
      FModulePanels[mspLinesOfCode].Text := Format(strLinesOfCode, [AFrame.LinesOfCode]);
      FModulePanels[mspUnderLimit].Text := Format(strUnderLimit, [AFrame.UnderLimit]);
      FModulePanels[mspAtLimit].Text := Format(strAtLimit, [AFrame.AtLimit]);
      FModulePanels[mspOverLimit].Text := Format(strOverLimit, [AFrame.OverLimit]);
    End;
End;
```

ProcessModule

This last general method has been added as it presents an interesting issue I, to date, have not had to tackle, and that is getting the source code (for parsing) for all the modules in a project. You might think that's easy I'll just open each module with `OpenModule()` from the `IOTAModuleInfo` interface (which you can get from the `IOTAModule` interface) however if you do this you will get the IDE to open every module in the project into memory (not necessarily as an editor tab) and this wouldn't be a good idea for large projects.

So what I've done here is see if the IDE has a module open with `IOTAModuleServices.FindModule()` and if

so get the source code from ther editor else I get the code from the disk file.

```
Procedure TBADIModuleMetricsEditorView.ProcesModule(Const ModuleInfo : IOTAModuleInfo);

Var
  Module: IOTAModule;

Begin
  FModified := False;
  Module := (BorlandIDEServices As IOTAModuleServices).FindModule(ModuleInfo.FileName);
  If Assigned(Module) Then
    LastModifiedDateFromModule(Module)
  Else
    LastModifiedDateFromFile(ModuleInfo);
  If FFileInfoMgr.ShouldUpdate(FFileName, FFileDate) Then
    Begin
      If Assigned(Module) Then
        ExtractSourceFromModule(Module)
      Else
        ExtractSourceFromFile;
        ParseAndRender;
    End;
End;
```

IDE Registration

This method is called from the main wizard's constructor to register this custom editor view.

Registering the View

```
Procedure RegisterMetricsEditorView;

Var
  EVS : IOTAEditorViewServices;

Begin
  If Supports(BorlandIDEServices, IOTAEditorViewServices, EVS) Then
    EVS.RegisterEditorView(strBADIMetricsEditorView, RecreateBADIStatisticEditorView);
End;
```

Unregistering the View

This method is called from the main wizard's destructor to unregister this custom editor view.

```
Procedure UnregisterMetricsEditorView;

Var
```

```
  EVS : IOTAEditorViewServices;

Begin
  If Supports(BorlandIDEServices, IOTAEditorViewServices, EVS) Then
    EVS.UnregisterEditorView(strBADIMetricsEditorView);
End;
```

Final Thoughts

Although I'm not in a position to provide you with a working example I've include the code for this module below. On the Browse and Doc It web page you can download a beta test which includes this functionality (for XE3 to Tokyo only).

BADI.Module.Metrics.pas

Related posts:

1. RAD Studio Custom Editor Sub-views (15.8)
2. Chapter 10: Reading editor code (8.8)
3. Chapter 8: Editor Notifiers (6.9)
4. Chapter 11: Writing editor code (6.6)
5. Chapter 5: Useful Open Tools Utility Functions (5.6)

Category: Miscellaneous Tags: INTACustomEditorView, INTACustomEditorView150, INTACustomEditorViewStatusPanel, IOTAModuleServices

Iconic One Theme | Powered by Wordpress