TMS TInspectorBar DEVELOPERS GUIDE

July 2019

Web: https://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Availability

TMS TInspectorBar is a VCL component for Win32 & Win64 application development and is available for Delphi 7, 2007, 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle,10.1 Berlin C++Builder 2007, 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin (Prof/Enterprise/Architect)

List of included components

- TINIInspectorBar
- TInspectorBar
- TDBInspectorBar
- TRTTIInspectorBar
- TAEInspectorEditLink
- TColComboInspectorEditLink
- TMemoInspectorEditLink
- TAdvMoneyEditInspectorEditLink
- TAdvSpinEditInspectorEditLink

Online references

TMS software website:
http://www.tmssoftware.com

TMS TInspectorBar page:
http://www.tmssoftware.com/site/inspbars.asp

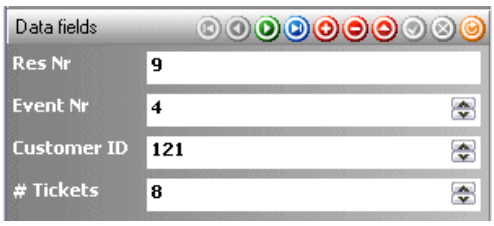
Purchase a license

TMS TInspectorBar is available in the following bundles:

- TMS List Controls Pack: http://www.tmssoftware.com/site/listcontrols.asp
- TMS Component Pack: http://www.tmssoftware.com/site/tmspack.asp
- TMS Component Studio: http://www.tmssoftware.com/site/studio.asp
- TMS VCL Subscription: http://www.tmssoftware.com/site/vdsub.asp

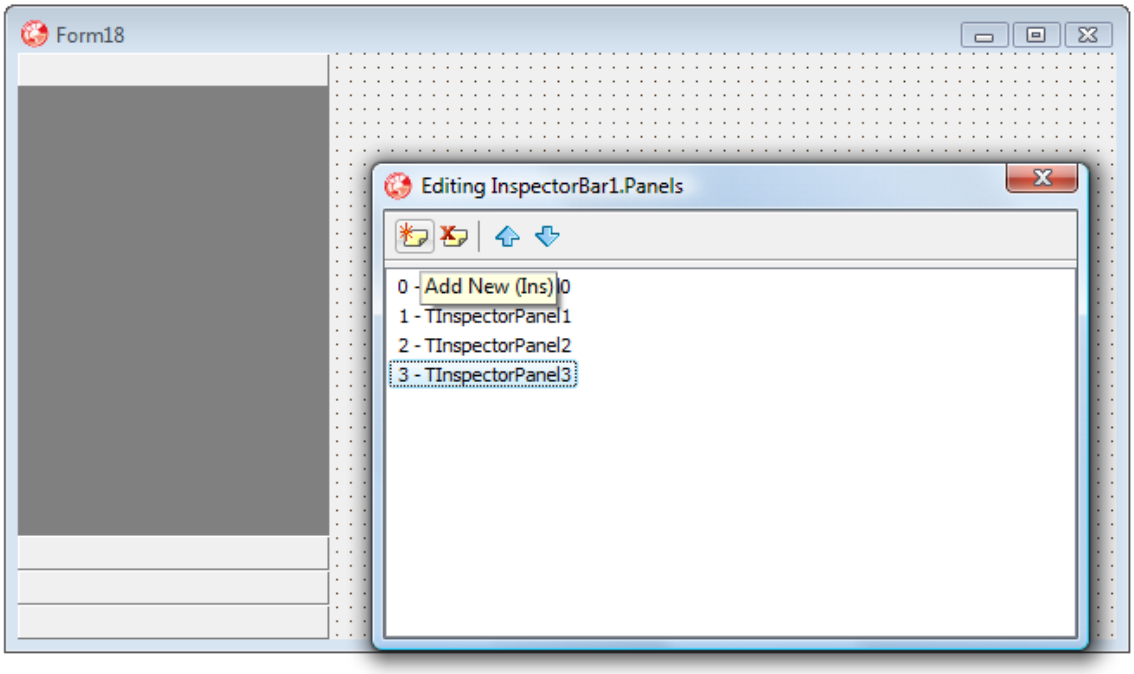
Main features

- TInspectorbar, TINIInspectorbar, TRTTIInspectorbar, TDBInspectorbar
- Standard Outlook large and small icon size panels
- Can accept controls in InspectorPanels
- Build in support for checkboxes, edits, comboboxes, spin editors, datetime pickers, font editing, color picker, password edit ..
- Custom inplace editors support
- Visual Studio .NET toolbox style
- Optional build-in DBNavigator in DBInspectorBar
- Background textures, gradients, images in items
- Various shading styles for panel captions, including XP button style & bitmap effects
- Various hover effects
- Full visual styles support on Windows XP
- GlyFX DBNavigator glyphs
- Compatible with TMS TAdvFormStyler / TAdvAppStyler



Getting started

Drop a TInspectorBar on the form and right-click to open the panels editor via the context menu to add panels to the TInspectorbar[1].

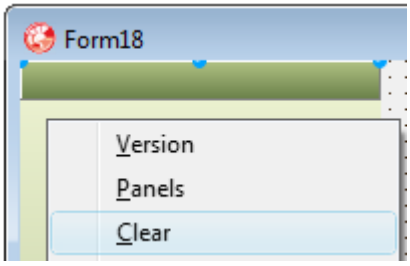


The equivalent code to insert a panel programmatically is:

```
procedure TForm18.FormCreate(Sender: TObject);
begin
  InspectorBar1.Panels.Add;
  InspectorBar1.Panels[0].Caption := 'Dynamic created panel';
end;
```

The method 'clear' is also available from the context menu and removes all existing panels.

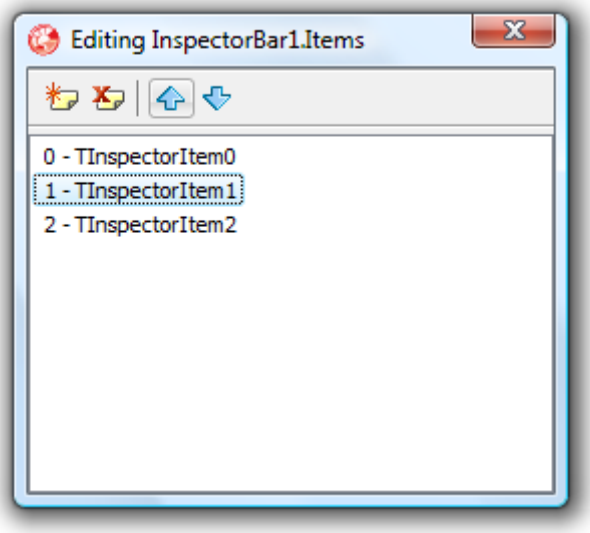
[1] TIP: When using the TMS Component Pack and the context menu Panels to add TInspecorPanels does not appear, this means you have not installed the design time package. Make sure that the package TMSDExx.DPK is effectively installed and active in the IDE and this issue will be solved.



The equivalent code to clear a panel programmatically is:

```
procedure TForm18.FormCreate(Sender: TObject);
begin
InspectorBar1.Panels.Clear;
end;
```

Via the items editor we can add new items, delete an existing item or move a selected item up/down.



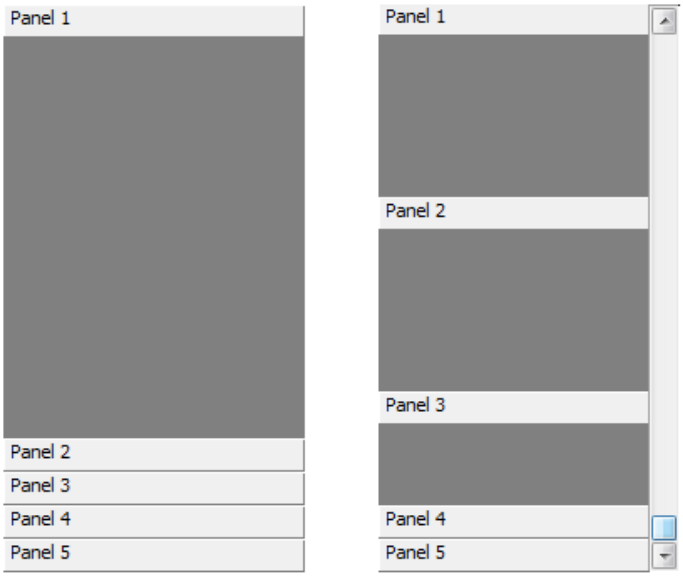
The equivalent code to insert an item programmatically is:

```
procedure TForm18.Button1Click(Sender: TObject);
begin
  inspectorbar1.Panels.Clear;
  inspectorbar1.Panels.Add;
  inspectorbar1.Panels[0].Items.add.Caption := 'item 1';
  inspectorbar1.Panels[0].Items.add.Caption := 'item 2';
end;
```

TInspectorPanel with Items collection

The InspectorPanel has 4 styles. It can be used as navigational control where each item represents a selection. The selection can be shown as a small image with caption text on the right side, a larger image with caption at the bottom and a radiobutton.

In addition to this mode, there is the psProperties style where an item represents a property shown as a name in the left column and a value with optionally an editor for the value in the right column.



When using the InspectorPanel in style psProperties, the left column shows the caption of properties. The caption text is set with the Item's Caption property. The value is shown in the right column and the value can be set in different ways depending on the type:

- InspectorItem.IntValue: integer: value as integer
- InspectorItem.FontValue: TFont: value as font
- InspectorItem.TextValue: string: value as text
- InspectorItem.PictureValue: TPicture: value as picture
- InspectorItem.BoolValue: boolean: value as boolean
- InspectorItem.ColorValue: TCOlor: value as TColor

The width of the left column is set with InspectorPanel.CaptionWidth and the background color of the left column is set with InspectorPanel.CaptionColor. The font for the caption can also be set with a different font. It is controlled by InspectorPanel.CaptionFont. When InspectorPanel.AllowResize is set to true and InspectorPanel.ShowGridLines is also set to true, the left column can be resized at runtime by clicking and dragging the column grid line. The type of the editor is set with InspectorItem.PropertyType.

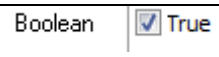
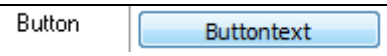
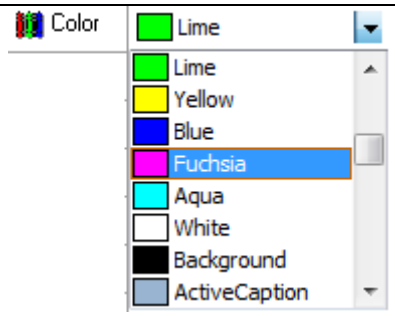
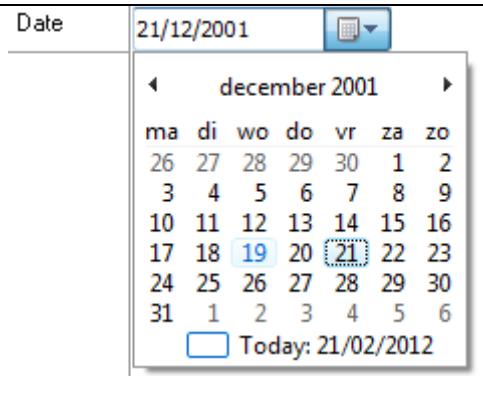
TInspectorItem

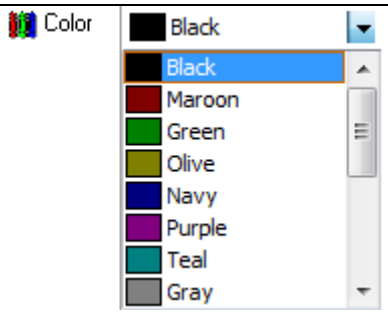
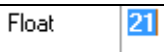
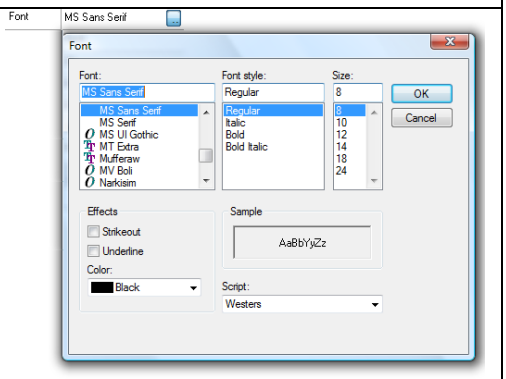
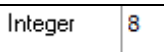
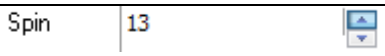
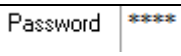
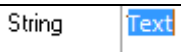
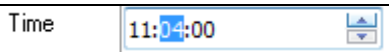
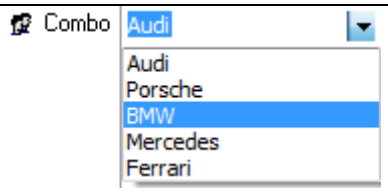
When the InspectorItem is used in a TInspectorPanel with style psButton, psLargeIcon, psSmallIcon, the properties that are used are the caption and several properties that can be used to set the image:

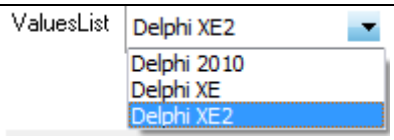
- Caption: sets the caption text
- Bitmap: can be used to set the image to associate with the item
- Icon: can be used to set the icon to associate with the item
- ImageIndex: sets the index of an imagelist image to be used for the item
- URL: optional hyperlink that can be opened when the item is clicked

When the InspectorItem is used in a TInspectorPanel with style psProperties, several additional properties are used. The PropertyType will determine how the property is shown in the right column and what editor will be used to edit it.

The properties types are:

ptBoolean: boolean value	
ptButton: button is shown in the right column for the property	
ptColor: color value	
ptCustom: custom inplace editor can be used to edit the property	
ptDate : date value	

ptFixedColor : non editable color rectangle is shown	
ptFloat: floating point value	
ptFont: font value	
ptInteger: integer value	
ptIntSpin: integer value with spin editor	
ptPassword: text value with password editor	
ptPicture: picture value	
ptPropButton: text value with button to start editor	
ptText: text value	
ptTextButton: text value with editor with embedded button	
ptTime: time value	
ptValues: text value with editable combobox to select value from	

ptValueList: text value with non-editable combobox to select value from	

The inplace editors are accessible via:

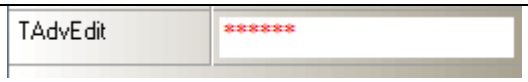
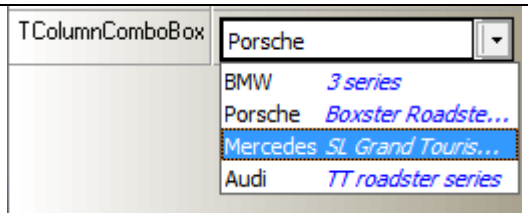
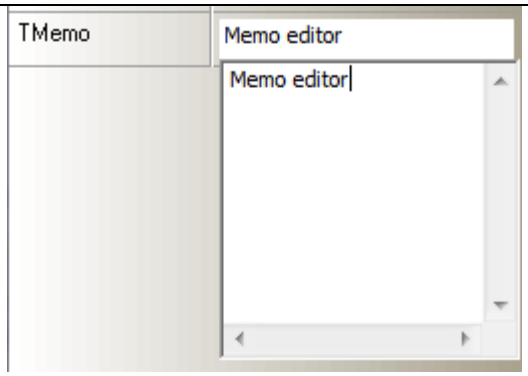
- InspectorBar.Spin: TInspectorSpin: spin edit control
- InspectorBar.Edit: TInspectorEdit: regular edit control
- InspectorBar.EditBtn: TInspectorEditBtn: button with embedded button
- InspectorBar.Combo: TInspectorCombo: combobox control
- InspectorBar.DateTimePicker: TInspectorDateTimePicker: date or time edit control
- InspectorBar.ColorCombo: TInspectorColorCOmbo: color combobox

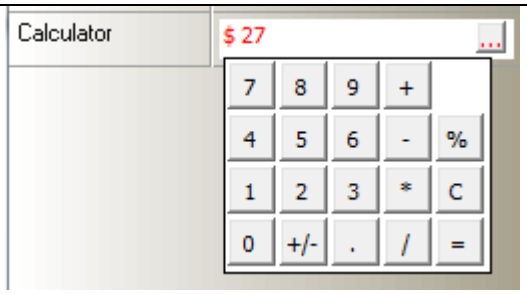
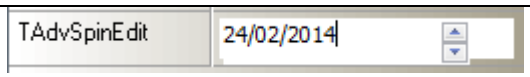
Custom editors

Add an InspectorEditLink on the form and assign the component to Items.Editlink.

(More information on custom editors can be found in the InspectorBarCustomEditors demo that is included in the samples distribution.)

Current available custom editors are:

TAEInspectorEditLink	
TColComboInspectorEditLink	
TMemoInspectorEditLink	

TAdvMoneyEditInspectorEditLink	
TAdvSpinEditInspectorEditLink	

Events

Following events are available in the TInspectorBar:

- OnButtonClick
- OnCustomEditButton
- OnCustomEditDraw
- OnEditAutoAdvance
- OnEditBtnClick
- OnEditCheckChange
- OnEditColorChange
- OnEditComboChange
- OnEditDblClick
- OnEditSpinChange
- OnEditSpinDown
- OnEditSpinUp
- OnEditStart
- OnEditStop
- OnEditUpdate
- OnFileDrop
- OnGetValueList
- OnHelpAnchorClick
- OnItemAnchorClick
- OnItemClick
- OnItemClose
- OnItemDblClick
- OnItemDraw
- OnItemOpen
- OnItemRightClick

- OnItemValue
- OnPanelCaptionClick
- OnPanelCaptionRightClick
- OnPanelClose
- OnPanelDraw
- OnPanelOpen
- OnPanelOpened
- OnStartLabelEdit
- OnStopLabelEdit
- OnURLClick
- OnURLDrop

Styles

Metro style

To make it easier and faster, TInspectorBar has built-in presets for Office colors and can also set the TInspectorBar in Metro style.

To select a Metro style, following properties are available:

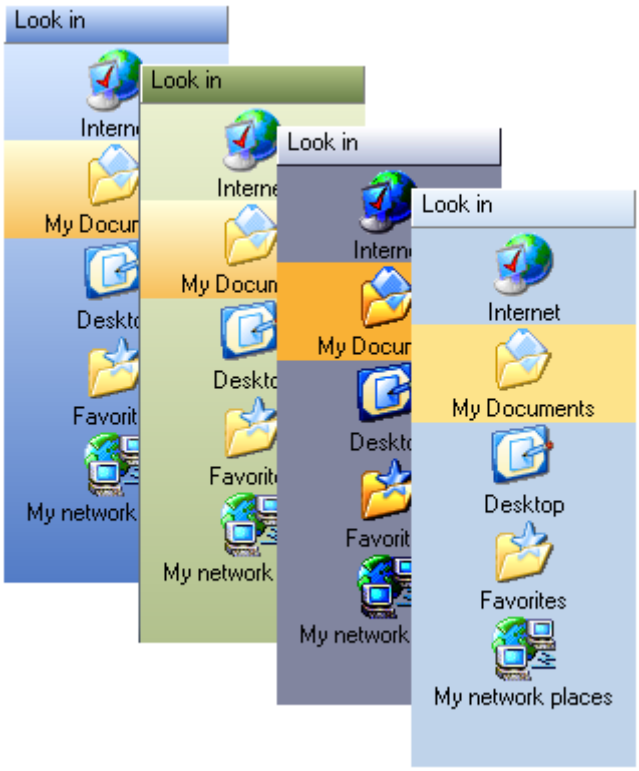
Metro: Boolean : when true, the Metro style is selected
MetroColor: TColor : sets the Metro accent color
MetroTextColor: TColor : sets the Metro text color
MetroStyle: TMetroStyle (msLight, msDark) : selects between a light (white background) or dark (black background) Metro style.

The Metro style can also be set in code. The TInspectorBar component implements the ITMSTones interface (see TMS Metro Controls guide for more background on this interface and Metro style in general). As such, the method InspectorBar.SetColorTones(ATones: TColorTones) can be called. To set the default Metro tones, add the unit AdvStyleIF and the code:

```
procedure TForm18.FormCreate(Sender: TObject);
begin
  InspectorBar1.SetColorTones(DefaultMetroTones);
end;
```

TAdvFormStyler, TAdvAppStyler

TInspectorBar is compatible with TAdvFormStyler, TAdvAppStyler.



A TAdvFormStyler is supposed to be dropped on a form and it will control the style of all TMS components on the form that implement the ITMSStyle interface. TInspectorBar implements this interface. A TAdvFormStyler will only affect components on the form itself. For application-wide appearance control, in addition to a TAdvFormStyler on a form, a TAdvAppStyler component can be dropped on a datamodule and is connected to the TAdvFormStyler components on the forms. By setting then a single property in TAdvAppStyler on the datamodule, the complete application appearance can change, both at design-time but also dynamically at run-time.

Please see the article: http://www.tmssoftware.com/site/atbdev3.asp that explains how you can use the TAdvFormStyler /TAdvAppStyler.

Current available Office styles in TInspectorBar are:

tsOffice2003Blue : Office 2003 style on a blue XP theme
tsOffice2003Silver : Office 2003 style on a silver XP theme
tsOffice2003Olive : Office 2003 style on an olive XP theme
tsOffice2003Classic : Office 2003 style on a non themed XP or pre XP operating system
tsOffice2007Luna : Office 2007 Luna style
tsOffice2007Obsidian : Office 2007 Obsidian style
tsOffice2007Silver : Office 2007 Silver style
tsWindowsXP : Windows XP / Office XP style
tsWhidbey : Visual Studio 2005 style

tsCustom : unforced style
tsWindowsVista : Windows Vista style
tsWindows7 : Windows 7 style
tsWindows8 : Windows 8 style
tsWindows10 : Windows 10 style
tsTerminal : reduced color set for use with terminal servers
tsOffice2010Blue : Office 2010 Blue style
tsOffice2010Silver : Office 2010 Silver style
tsOffice2010Black : Office 2010 Black style
tsOffice2013White : Office 2013 White style
tsOffice2013LightGray : Office 2013 Light Gray style
tsOffice2013Gray : Office 2013 Gray style
tsOffice2016White : Office 2016 White style
tsOffice2016Gray : Office 2016 Gray style
tsOffice2016Black : Office 2016 Black style

Tips and FAQ

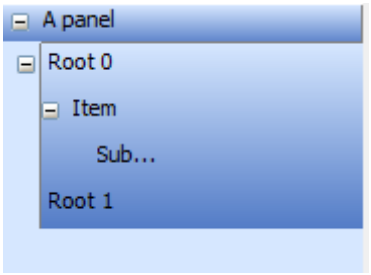
Adding a tree structure in the InspectorBar

A tree structure can be added in InspectorBar items by using the item's Level property. By default, all items have Level = 0, meaning that all items are treated as root items. When Level is set to a value different from zero, this means that the item becomes the child item of another item before this item with a smaller Level property value. In this sample code snippet, a panel is programmatically created and a tree structure of a root item, a child item and a child of this child item is inserted.

Following code:

```
begin
  InspectorBar1.Panels.Add;
  InspectorBar1.PanelCaption.SideDisplay := true;
  InspectorBar1.PanelCaption.SideWidth := 20;
  InspectorBar1.PanelCaption.OpenCloseGraphic := ocgCross;
  InspectorBar1.Mode := imMultiPanelActive;
  InspectorBar1.Panels[0].Caption := 'A panel';
  InspectorBar1.Panels[0].ItemHeight := 23;
  InspectorBar1.Panels[0].Items.Add;
  InspectorBar1.Panels[0].Items.Add;
  InspectorBar1.Panels[0].Items.Add;
  InspectorBar1.Panels[0].Items.Add;
  InspectorBar1.Panels[0].Style := psProperties;
  InspectorBar1.Panels[0].Items[0].Level := 0;
  InspectorBar1.Panels[0].Items[0].Caption := 'Root 0';
  InspectorBar1.Panels[0].Items[1].Level := 1;
  InspectorBar1.Panels[0].Items[1].Caption := 'Item';
  InspectorBar1.Panels[0].Items[2].Level := 2;
  InspectorBar1.Panels[0].Items[2].Caption := 'SubItem';
  InspectorBar1.Panels[0].Items[3].Level := 0;
  InspectorBar1.Panels[0].Items[3].Caption := 'Root 1';
end;
```

Results in:



Programmatically adding a property panel and item and starting editor

This code adds a new panel in property style and adds on regular editable item:

```
var
  pnl: TInspectorPanel;
  inspi: TInspectorItem;
begin
  pnl := inspectorbar1.Panels.Add;
  pnl.Style := psProperties;
  inspi := pnl.Items.Add;
  inspi.PropertyType := ptText;
  inspi.Caption := 'Edit';
end;
```

This item can be programmatically set into edit mode with the code:

```
begin
  inspectorbar1.SetFocus;
  inspectorbar1.StartEdit(inspectorbar1.Panels[0].Items[0]);
end;
```

How to programmatically open / close a panel

You can programmatically open / close a panel in imMultiPanelActive mode by toggling the Panel.Open: Boolean property.