



home articles quick answers discussions features community

help

Search for articles, questions, tips

Articles » Languages » VB.NET » General

Next →

Article

Browse Code

Bugs / Suggestions

Stats

Revisions (2)

Alternatives

Comments (334)



View this article's Workspace

Fork this Workspace



Connect using Git

Add your own alternative version

Share

Add Custom Properties to a PropertyGrid

By **Danilo Corallo**, 22 Aug 2006

★★★★★ 4.94 (195 votes) ▶

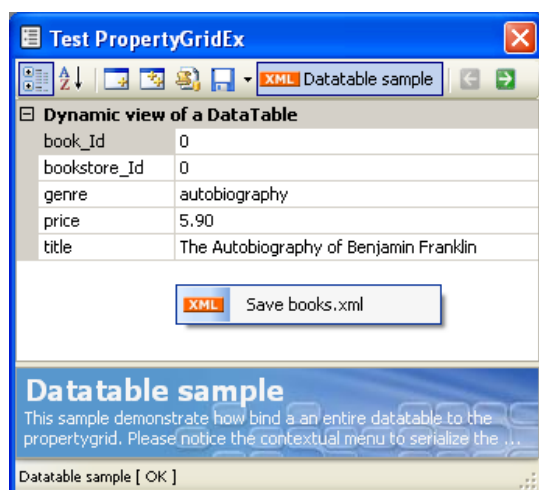
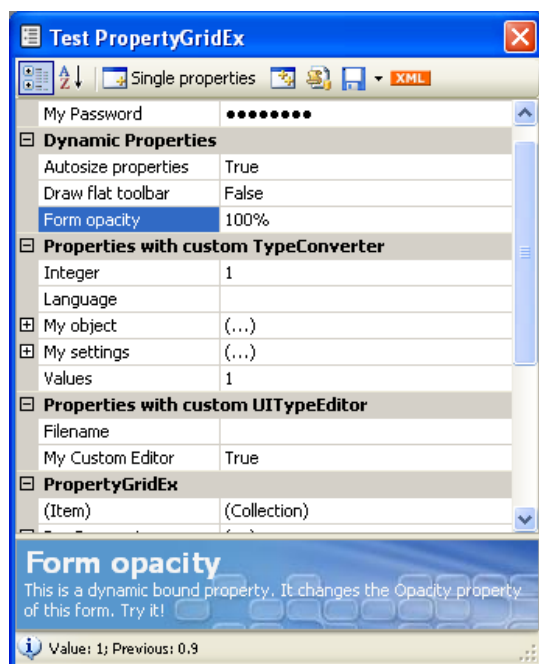
Rate:

Download VB.NET demo project v1.03 - 188 Kb

Download VB.NET source code v1.03 - 45.9 Kb

Download C# version (source code and demo project) v1.02 - 170 Kb

Download VB.NET 2003 version (source code only) v0.98 - 31.4 Kb



About Article

Extend a PropertyGrid with an Item collection; easy customization of properties with custom editor, custom converter and databinding.

Type **Article**

Licence **Apache**

First Posted **31 Mar 2006**

Views **673,876**

Downloads **13,933**

Bookmarked **405 times**

.NET1.1 .NET2.0
VS.NET2003 VS2005 C# VI
+



Top News

The Secret Messages Inside Chinese URLs

Get the Insider News free each morning.

Related Videos



Related Articles

Add (Remove) Items to (from) PropertyGrid at Runtime

Customized display of collection data in a PropertyGrid

Win32 SDK PropertyGrid Made

Introduction

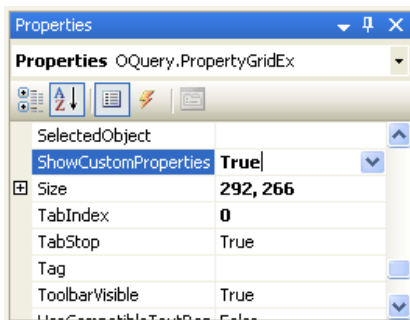
For my last project, I looked at the **PropertyGrid** control that has a very nice look and feel. Immediately, I noticed how difficult it was to use it. Looking over the Internet, I found lots of people wishing for something easier to use, like the **TreeView** or the **ListView**. So I started developing an extended version of the **PropertyGrid** that includes an **Item** collection.

Using the Code

The control is made of three basic classes:

- Class **CustomProperty**, that represents a single property.
- Class **CustomPropertyDescriptor**, that is used internally by the **PropertyGrid** to retrieve property attributes.
- Class **CustomPropertyCollection**, that represents a collection of **CustomProperties**.

In the end, we have the control **PropertyGridEx** that inherits from **PropertyGrid** and exposes an **Item** collection. To use the component, add it to the Toolbox and then to your form. Compared to the classic **PropertyGrid**, we have a new property to set, called **ShowCustomProperties**.



The property must be set to **True**, or the component will have the same behavior as the classic **PropertyGrid**.

The Item Collection

We are now ready to fill our custom properties.

```
With Properties
.Item.Clear()
.Item.Add("My String", "My Value", False, _
    "Simple type", "This is a string", True)
.Item.Add("My Integer", 100, False, "Simple type", _
    "This is an integer", True)
.Item.Add("My Double", 10.4, False, "Simple type", _
    "This is a double", True)
.Item.Add("My Font", New Font("Arial", 9), False, _
    "Classes", "This is a font class", True)
.Item.Add("My Color", New Color(), False, _
    "Classes", "This is a color class", True)
.Item.Add("My Point", New Point(10, 10), False, _
    "Classes", "This is point class", True)
.Refresh()
End With
```

The **CustomProperty** class exposes several properties that you can use to personalize your **PropertyGrid**:

- Name**, a string representing the property name;
- Value**, an object representing the value of the property;
- IsReadOnly**, a boolean that indicates if the property is editable or not;
- Category**, a string that represents the category in which the property is shown;
- Description**, a string that represents the description of the property, shown at the bottom of the component;
- Visible**, a boolean that indicates if the property is shown or not;

Finally, remember to **Refresh** the **PropertyGrid** after any modification to the collection.

Easy

Application Configuration
Editor using the PropertyGrid

Type Wrapper Classes and a
PropertyGrid

Using PropertyGrid Part-I

Bending the .NET PropertyGrid
to Your Will

Renderer for PropertyGrid

Filtering properties in a
PropertyGrid

Ordering Items in the Property
Grid

Extending the PropertyGrid with
a new PropertyTab

Making a PropertyGrid
ReadOnly

Dynamic Properties for
PropertyGrid

A C# 2008 Advanced
Customizable PropertyGrid
Control

Changing the look and feel of
the propertygrid

Gain Control Over
PropertyGrid's Description Area

Adding a File Open dialog to a
PropertyGrid control

Exploring the .NET PropertyGrid
in depth

Native WPF 4 PropertyGrid

Dynamic Type Description
Framework for PropertyGrid

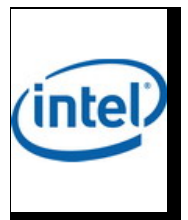
Related Research



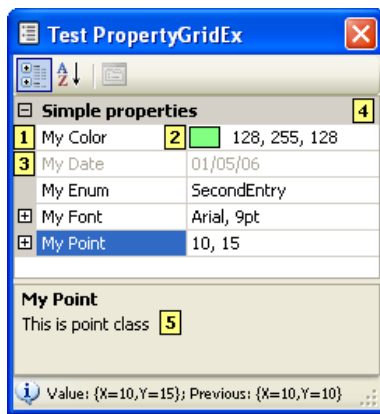
Fine-Tuning the Engines of SMB
Growth: 4 strategies for growing
your business



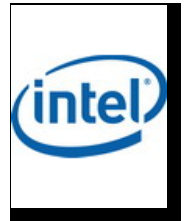
Protecting Your Business Data:
Five Do's and Don'ts



Custom API Management for
the Enterprise: Learn how to
build a successful API strategy
[Webinar]



- 1 `CustomProperty.Name`
- 2 `CustomProperty.Value`
- 3 `CustomProperty.IsReadOnly`
- 4 `CustomProperty.Category`
- 5 `CustomProperty.Description`



Insider Secrets on API Security
From Experts at Securosis
[Webinar]

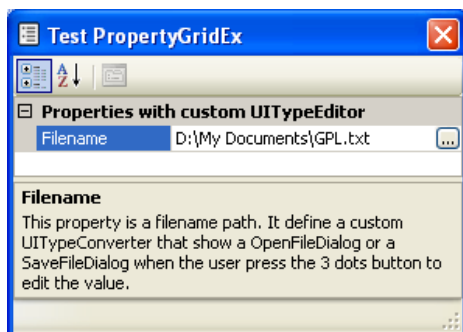
The Filename Editor

It is also possible to create a `CustomProperty` that shows a dialog to modify a filename. The trick is achieved by the class `UIFilenameEditor` that inherits from `System.Drawing.Design.UITypeEditor`. You can create a property, with the well known `FileDialog` as a Type Editor, in the following way:

[Collapse](#) | [Copy Code](#)

```
With Properties
    .Clear
    .Item.Add("Filename", "", False, "Misc", "", True)
    .Item(.Item.Count - 1).UseFileNameEditor = True
    .Item(.Item.Count - 1).FileNameDialogType = _
        = FileDialogType.LoadFileDialog
    .Item(.Item.Count - 1).FileNameFilter = _
        "Text files (*.txt)|*.txt|All files (*.*)|*.*"
    .Refresh()
End With
```

Following is the resulting `PropertyGrid`. Please notice the Ellipsis symbol beside the value of the property, to access the `UIFilenameEditor`.



Custom Choices Type Converter

It's a very commonly requested feature to have a list of values from which the user can choose, while editing a property. To implement a dropdown list, do the following:

[Collapse](#) | [Copy Code](#)

```
Dim Languages As String() = New String() {"English", _
                                           "Italian", _
                                           "Spanish", _
                                           "Dutch"}

With Properties
    .Clear
    .Item.Add("Language", "", False, "Misc", "")
    .Item(.Item.Count - 1).Choices = _
        = New CustomChoices(Languages, True)
    .Refresh()
End With
```

You can use arrays of `String`, `Integer`, `Double` and `Object` to initialize the `Choices` property. Following is the resulting `PropertyGrid` initialized with an array of `Strings`.



Enumerations

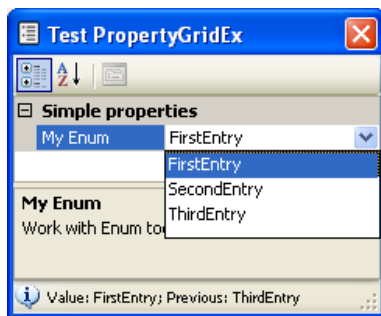
Most of the times, developers have enumerations declared in their code. The component automatically shows a dropdown list with the values from the **Enum**.

[Collapse](#) | [Copy Code](#)

```
Public Enum MyEnum
    FirstEntry
    SecondEntry
    ThirdEntry
End Enum

With Properties
    .Clear
    .Item.Add("Enum", MyEnum.FirstEntry, False, "Misc")
    .Refresh()
End With
```

Following is the resulting **PropertyGrid**.



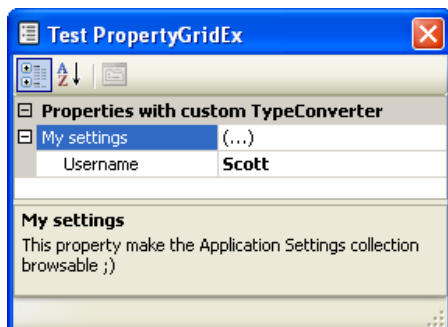
Expandable Object Converter

Using this **TypeConverter**, it is possible to expand nested properties. Here is an example that shows the application settings.

[Collapse](#) | [Copy Code](#)

```
With Properties
    .Clear
    .Item.Add("My settings", My.MySettings.Default, False, "Misc")
    .Item(.Item.Count - 1).IsBrowsable = True
    .Item(.Item.Count - 1).BrowsableLabelStyle = _
        BrowsableTypeConverter.LabelStyle.lsEllipsis
    .Refresh()
End With
```

Following is the resulting **PropertyGrid**.



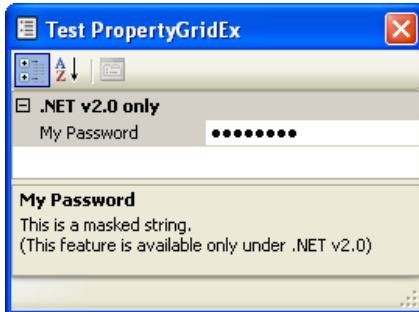
Password fields

Using masked fields, it is possible to hide the value of a property. The result is produced by the attribute **PasswordPropertyTextAttribute**, new to the .NET Framework v2.0. Here is an example on how to use it.

[Collapse](#) | [Copy Code](#)

```
With Properties
    .Clear
    .Item.Add("My Password", "password", False)
    .Item(.Item.Count - 1).IsPassword = True
    .Refresh()
End With
```

Following is the resulting **PropertyGrid**.



Dynamic property binding

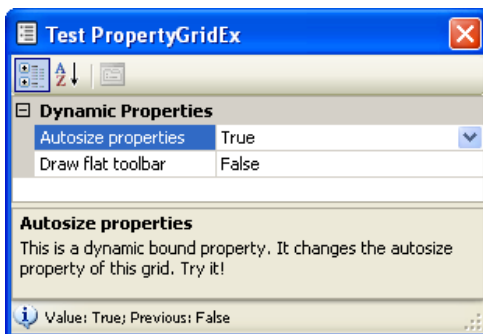
Using **PropertyGridEx**, it is possible to bind object properties to the component. This is done by a method of the collection that accepts a value argument by reference. In this way, it is possible to freely mix reference type and value type properties. The following example binds the **AutoSizeProperties** and the **DrawFlatToolbar** property of the grid.

Any modification to the value of these properties is automatically reflected into the object they belong to.

[Collapse](#) | [Copy Code](#)

```
With Properties
    .Clear
    .Item.Add("AutoSize properties", Properties, "AutoSizeProperties", False, _
        "Dynamic Properties", "This is a dynamic bound property", True)
    .Item.Add("Draw flat toolbar", Properties, "DrawFlatToolbar", False, _
        "Dynamic Properties", "This is a dynamic bound property", True)
    .Refresh()
End With
```

Following is the resulting **PropertyGrid**.



Multiple object properties

One more feature provided with this control is the ability of editing multiple object properties at the same time. This is done by a collection of objects attached to the **SelectedObjects** property of the base component. The behavior is that the grid only displays the properties that are common to all the objects that are in the array. To activate the multiple objects functionality, the property of the grid **ShowCustomPropertiesSet** must be set to **True**.

[Collapse](#) | [Copy Code](#)

```
With Properties
    .ShowCustomPropertiesSet = True
    .ItemSet.Clear()

    .ItemSet.Add()
    .ItemSet(0).Add("My Point", New Point(10, 10), False, "Appearance")
    .ItemSet(0).Add("My Date", New Date(2006, 1, 1), False, "Appearance")
End With
```

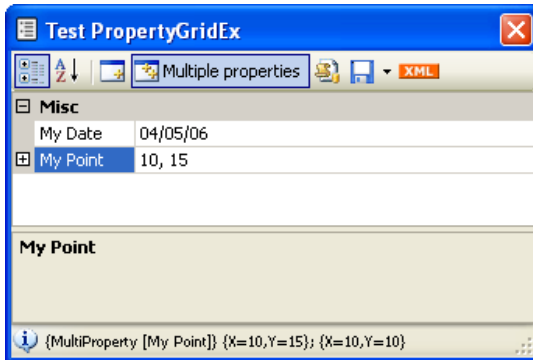
```

.ItemSet.Add()
.ItemSet(1).Add("My Point", New Point(10, 10), False, "Appearance")
.ItemSet(1).Add("My Date", New Date(2007, 1, 1), False, "Appearance")
.ItemSet(1).Add("My Color", New Color(), False, "Appearance")

.Refresh()
End With

```

Following is the resulting **PropertyGrid**.



Databinding of a Property to a Datasource

The databinding of a single property to a data-source is the latest addition to this component. The wrapper created accepts for a **CustomProperty**, three members:

- **Datasource**, the datasource to bind to.
- **DisplayMember** (optional), the field used to bind the list shown in the dropdown control.
- **ValueMember** (optional), the field used to bind the value returned by the **CustomProperty**.

The **CustomProperty** will return the following values:

- **Value**, that represents the value shown as **System.String**.
- **SelectedItem**, that represents the object selected as **System.Object**.
- **SelectedValue**, that represents the value selected as **System.Object**.

The following example creates three properties that bind a **DataTable**, an array of **Objects**, and an array of **Strings**.

[Collapse](#) | [Copy Code](#)

```

With Properties
.Item.Clear()
.Item.Add("Datatable", "", False, "Databinding", _
        "This is a UITypeEditor that implement a listbox", True)

.Item(.Item.Count - 1).ValueMember = "book_Id"
.Item(.Item.Count - 1).DisplayMember = "title"
.Item(.Item.Count - 1).Datasource = LookupTable

.Item.Add("Array of objects", ListValues(2).Text, False, "Databinding", _
        "This is a UITypeEditor that implement a listbox", True)

.Item(.Item.Count - 1).ValueMember = "Value"
.Item(.Item.Count - 1).DisplayMember = "Text"
.Item(.Item.Count - 1).Datasource = ListValues

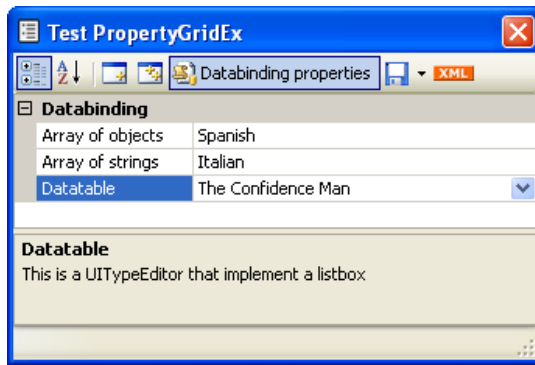
.Item.Add("Array of strings", Languages(1), False, "Databinding", _
        "This is a UITypeEditor that implement a listbox", True)

.Item(.Item.Count - 1).Datasource = Languages
.Refresh()
End With

```

Please notice that the result is very different from the one achieved by using the "Custom Choices" functionality, that uses a **TypeConverter**. The databinding feature uses a **UITypeConverter** that implements a **ListBox** control.

Following is the resulting **PropertyGrid**.



Custom Event Editor

Using **PropertyGridEx**, it is possible to bind the 3-dots button used in Modal editors with a custom event handler. This is done by a method of the **CustomProperty** that accepts a **Delegate** as argument. The following is an example:

[Collapse](#) | [Copy Code](#)

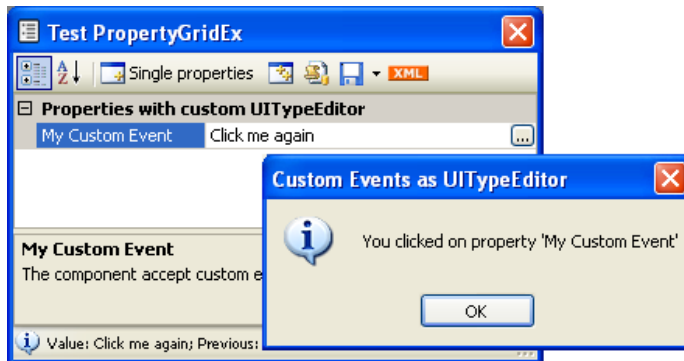
```
With Properties
    .Clear
    .Item.Add("My Custom Event", "(Click me)", False, "Misc", _
        "The component accept custom event handler.", True)
    .Item(.Item.Count - 1).OnClick = AddressOf Me.CustomEventItem_OnClick
    .Refresh()
End With
```

The event will be handled by this function:

[Collapse](#) | [Copy Code](#)

```
Private Function CustomEventItem_OnClick(ByVal sender As Object, _
    ByVal e As EventArgs) As Object
    MsgBox("You clicked on property '" & sender.CustomProperty.Name & "'", _
        MsgBoxStyle.Information, _
        "Custom Events as UITypeEditor")
    Return "(Click me again)"
End Function
```

Please notice that the result value of the function will be the new value of the property. Following is the resulting **PropertyGrid**.



Add some style to the control

Looking on Internet and on CodeProject, I found a lot of articles regarding this component. Unfortunately, I had to merge everything together to have an easier **PropertyGrid** to use. In this section, you'll find small features I've found and added in this project.

- **AutoSizeProperties** - Move automatically the splitter to better fit all the properties shown.
- **MoveSplitterTo** - Move the splitter as indicated by the user in the parameter.
- **DocComment** - Expose the comments area as a control.
- **DrawFlatToolbar** - Draw a flat toolbar or a VS like toolbar.

Please notice that the last property switches the drawing of the toolbar within a Professional Renderer with custom colors and a System Renderer.

Final Notes

I hope that you find this article useful. If you found this article stupid, annoying, incorrect, etc., express this fact by rating the article as you see fit.

Credits

Thanks to Pascal Higelin for providing the Property Binding feature and the C# version.
Thanks to Suresh Kavan, for the idea and the test case for the data-binding feature.

References

- [Getting the Most out of the PropertyGrid Control](#)
- [Add Item to PropertyGrid at Runtime](#)
- [Bending the .NET PropertyGrid to Your Will](#)
- [Handy Type Editors: Customizable Filename Editor](#)
- [Handy Type Editors: Universal Dropdown Editor](#)

History

- 22nd August 2006
 - Added XML Editor sample.
- 31st May 2006
 - Added custom event editor.
 - Added DocComment interface.
- 5th May 2006
 - Added serialization.
 - Added property databinding.
 - Article updated.
- 28th April 2006
 - Added C# version.
 - Added dynamic property binding.
 - Added multiple objects feature.
- 10th April 2006
 - Added password property and improved ToolStrip look.
 - Added expandable object converter.
 - Added custom choices type converter.
 - Added file name editor.
- 31st March 2006 – First submission.

License

This article, along with any associated source code and files, is licensed under [The Apache License, Version 2.0](#)

About the Author



Danilo Corallo

Web Developer
Italy 

I am 29 years old and I've been working with C++, Visual Basic .NET, C# and ASP.NET. I have a large experience in Industrial Automation solutions, but I've worked also as Web developer and DBA. I like to share knowledge and projects with other people.

[Article Top](#)

Comments and Discussions



Search this forum

Go

☐ Profile popups Spacing

Relaxed

 Noise

Medium

 Layout




Normal













 Per page










25

Update

First Prev Next

 PropertyGridEx.CustomProperty.Choices broken? 	 The Mighty Atom	29-Nov-13 13:25
 Here is another one... 	 Mizan Rahman	20-Sep-13 6:54
 Re: Here is another one... 	 Danilo Corallo	20-Sep-13 8:16
 Need: display Name with blank for the property name 	 down8899	19-Aug-13 4:04
 My vote of 5 [modified] 	 BillWoodruff	7-Aug-13 9:37
 My vote of 5 	 Member 10091804	6-Jun-13 3:50
 My vote of 5 	 Jibsta-Man	24-Jan-13 6:57
 nice code,thank you! 	 dpuser	7-Jan-13 1:51
 My vote of Five 	 CNeumann2	14-Dec-12 6:33
 extended property grid 	 Neil Haughton	12-Nov-12 9:43
 Custom Properties Not Updating 	 PQSIK	18-Oct-12 10:59
 Excellent piece of work 	 Daniel.Brylak	10-Jul-12 5:44
 My vote of 5 	 radicalfish	15-May-12 3:46
 My vote of 5 	 ProEnggSoft	1-Mar-12 11:16
 Very Nice 	 Mike Hankey	25-Jan-12 20:07
 Show selected items, comma seperated 	 Member 3813338	1-Aug-11 4:35
 DocComment Height 	 Shaka913	4-Apr-11 6:38
 Child Nodes 	 malcomm	14-Dec-10 21:23
 Issues with saving color properties 	 malcomm	13-Dec-10 20:14
 Ordering categories in non-alphabetical order? 	 Brazilian.Engineer	6-Oct-10 11:45
 Ability to control sort order of Categories in PropertyGridEx ??? 	 Brazilian.Engineer	1-Oct-10 20:30

 Enable Enter or Tab Key to move to the next item? 	 BrianGoodheim	26-Sep-10 15:49
 is it possible to disable auto-sort? 	 mryux2005	2-Sep-10 2:16
 Re: is it possible to disable auto-sort? 	 dusan.fedorcak	7-Sep-10 8:53
 multi selection drop down box 	 Member 2281771	24-Aug-10 15:53
Last Visit: 12-May-14 8:59 Last Update: 13-May-14 22:18 Refresh 1 2 3 4 5 6 7 8 9 10 11 Next »		

 General
  News
  Suggestion
  Question
  Bug
  Answer
  Joke
  Rant
  Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.