Mole v4.2 - Visualizer With Property Editing

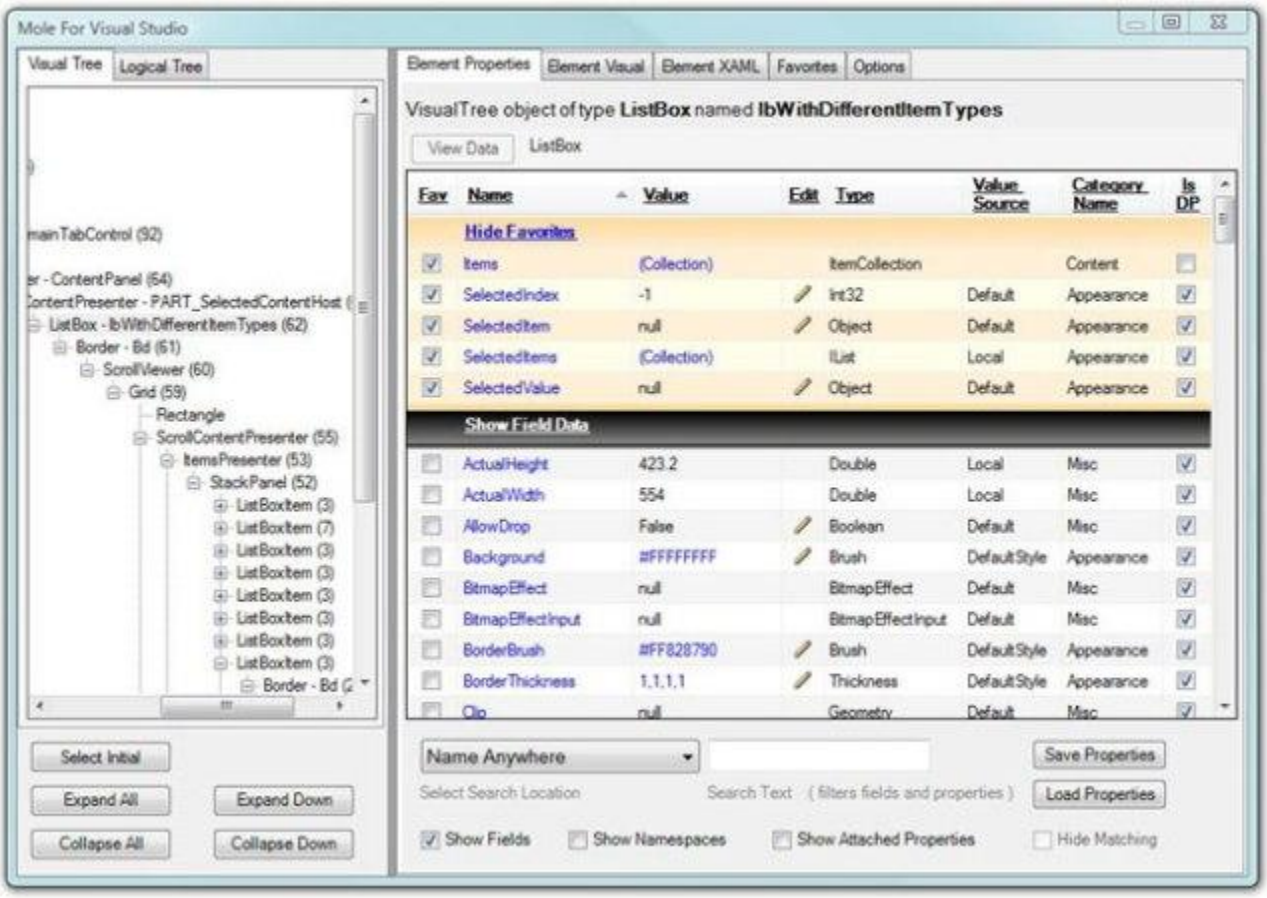


Table Of Contents

Mole v4.2 Update New Features!

- Mole v4.2 now supports using Mole with WorkFlow (WF) applications. A new test bench project for WF was added to the Source and Test Benches solution. For full details, please see WF Project Example section below.
- **"The Rock Star Hack of 2008!" -** We at Team Mole have been trying for a long time to figure out a generic way for developers to launch Mole. Josh "Rock Star" Smith has finally breached this stone wall with his awesome hack. Mole has been modified so that Visual Studio can reconginze `System.WeakReference` as a supported `Type` that Mole can visualize. For full details please see the Launch Mole using a WeakReference Hack section below.
- Suggestion: If you subscribe to Karl's blog feed, you will be notified as new Mole videos are added along with new version notifications of Mole.

Introduction

Mole was authored by Karl Shifflett, Josh Smith and Andrew Smith who make up Team Mole. The core development process took the team several weeks. Most of the additional enhancements were implemented by Andrew and Karl, with a lot of testing and feedback provided by Josh. You can visit Team Mole's Home Page here. This article was written by Karl.

Mole has been tested on **WPF, WCF, WF, WinForms and ASP.NET** projects on Vista, XP Pro, x32, x64, VS2005 and VS2008, C# and VB.NET. We sincerely hope all .NET developers like and use Mole. Currently Mole does not support Silverlight but we will consider writing a visualizer, or similar tool, shortly after Silverlight 2.0 is released to the public.

This article makes no assumptions about your prior exposure to the Mole project, or visualizers in general.

What is Mole?

Mole is a Visual Studio visualizer. Visualizers have been part of Visual Studio since version 2005. During debugging sessions, visualizers allow developers to view objects and data using a customized interface. Visual Studio ships with several simple but useful visualizers. Many developers have posted visualizers for .NET classes.

Mole was designed to not only allow the developer to view objects or data, but to also allow the developer to drill into properties of those objects and then edit them. Mole allows unlimited drilling into objects and sub-objects. When Mole finds an `IEnumerable` object, the data can be viewed in a `DataGridView` or in the properties grid. Mole easily handles collections that contain multiple types of data. Mole also allows the developer to view non-public fields of all these same objects. You can learn a lot about the .NET Framework by drilling around your application's data.

Depending on the type of object you are visualizing you can view properties, fields, `IEnumerable` collection data, an image of the data/control, and run-time XAML.

Mole v4 allows editing of properties. Please see the Editing section below for full details.

Quick History

When a great friend and co-author Josh Smith wrote Woodstock for WPF on 13 Nov 2007, I got pretty excited about how Visual Studio visualizers that target WPF can really assist developers. I started making a list of features I wanted and began working on Mole.

Mole was my first visualizer and first real WinForms program I've created with .NET 2.0. I spend most of my time with WPF, WCF, ASP.NET, SQL Server and Windows Services. So starting from scratch would prove to be challenging at times.

Mole grew alongside Woodstock as it was being developed and refined. Josh and I worked hard to develop fast performing visualizers. We both tried some crazy things during this process of refinement, some of these I wouldn't post on a billboard or even admit to a priest during confession.

To give you a comparison of what we were up against from a data loading perspective, consider the Text Visualizer which ships with Visual Studio. It allows you to view long strings in a multiline `TextBox`. Text Visualizer opens quickly and displays the string.

Mole displays a tree of the UI's elements and over a hundred associated properties in a `DataGridView` for any selected element in the `TreeView` in just about one second! Let's face it, the visualizer needs to load fast or it won't get used.

What you'll see in this article is a highly refined visualizer which runs lightning fast. When running Mole in stress tests with absurdly large WPF visual trees (over 10,000 elements) it was able to open in less than a second. Bearing in mind that most sane developers would never create visual trees, WinForms or ASP.NET pages with element counts of that size, it is

safe to say that Mole is fast enough to be usable by the most impatient developers out there. The largest visual tree loaded in Mole that I know about is 125,000 items! Josh was just stress testing Mole.

On 26 November 2007 Mole v1, a Visualizer for WPF, was released. After Mole v1 was published, Josh, Andrew and I teamed up and have continued making significant upgrades to Mole v1.

Due to the tremendous response from the WPF community, we radically expanded Mole's capabilities. On 6 Dec 2007, we released Mole v2. On 11 Dec 2007, we made further enhancements and released Mole v2.2 Black Ops. However, these tools only targeted WPF developers.

Karl just couldn't put Mole down until extending Mole to all Visual Studio developers. So on 14 Dec 2007, we released Mole v3 that allowed all Visual Studio project types to be inspected by Mole.

After Mole v3, Andrew really wanted to add editing capabilities to Mole. So Andrew and I jumped off that cliff to bring property Editing to Mole. This was not as easy task.

Mole v4 adds editing of properties to Mole's already awesome feature set. Please see the Editing section for a full explanation of this new and powerful feature.

This project has been a wild adventure to say the least. In just over a month, we have gone from never using a visualizer to authoring this product.

For those that are interested, our development team employs a very specialized formal development methodology that is described towards the end of this article. When Josh and I started on our beginner visualizers a month ago, we never envisioned where Mole would take us. We have been working almost non-stop since we started. All three of us work regular jobs and work on Mole when we can.

One of the side effects of just coding our brains out like this, almost without a rudder, is that Mole has a few areas that could use some refactoring. We are aware of this. We are considering a new product next year that can use this one as a prototype and one that will be designed before we start coding. This was a ready-fire-aim operation. We had a lot of fun with Mole!

We are three regular guys, two C#, one VB.NET. Yes we are all bilingual, but have our preference of expression. We live in three different states and two of us have never met or spoken on the phone! Additionally, right in the middle of this Josh had to move and got a new job! Meanwhile, Karl would fire off new features like the Fourth of July. Andrew has been like a sniper, who assassinates the really nasty bugs and then disappears into the thickets until needed again.

Installation Requirements

You must install the .NET 3.0 Framework on your computer before installing Mole.

Standalone Installation

If you are developing on **Vista** and are running with **Elevated Security** enabled, you must install Mole in the following directory:

- *VS Install path**Common7**Packages**Debugger**Visualizers*

All others download and unzip the above package. Copy the file in the Release package to either:

- *My Documents**Visual Studio 2005**Visualizers {VS2005}*
- *My Documents**Visual Studio 2008**Visualizers {VS2008}*

ASP.NET Developers Please Read This

When using Mole with **ASP.NET** projects that utilize IIS as the web server, as opposed to the Visual Studio's built in web server, you **MUST** give the account that the web site is running under, `Read` and `Read & Execute` permissions to the *Visualizers* directory.

If you do not do this, you will get an exception when attempting to load the visualizer in a debugging session. You would get this exception for any visualizer and not just Mole. This is because the ASPNET account has very few permissions on your computer. Adding these permissions prevents this exception.

Source Code Installation

We have provided source and test bench programs for both VS2005 and VS2008. The only difference between the two versions of the visualizer is the reference to *Microsoft.VisualStudio.DebuggerVisualizers.DLL*. For VS2005 the file version is 8.0.0.0 and for VS2008 the file version is 9.0.0.0.

Copy the projects to your *Projects* directory for either VS 2005 or VS 2008. Don't forget, after compiling, you must copy the binary to the appropriate directory.

The source code has two solutions. One solution has the visualizer project and also WPF, WinForm and ASP.NET test-bench programs that Josh and I wrote. The other solution has the WCF test bench program. I used a Chapter One sample from Michele Leroux Bustamante's book I've read, "Learning WCF". I strongly recommend "Learning WCF" to anyone interested in checking out WCF.

ASP.NET Developers Please Read This

When using Mole with **ASP.NET** projects that utilize IIS as the web server, as opposed to the Visual Studio's built in web server, you **MUST** give the account that the web site is running under, `Read` and `Read & Execute` permissions to the *Visualizers* directory.

If you do not do this, you will get an exception when attempting to load the visualizer in a debugging session. You would get this exception for any visualizer and not just Mole. This is because the ASPNET account has very few permissions on your computer. Adding these permissions prevents this exception.

Mole's Feature Set

- Lightning fast performance is realized by using a multi-threaded, lazy-loaded architecture for passing data between processes. We also used custom serialization of our data objects to reduce the number of fields that are packaged and sent over the visualizer's remoting channel.
- View application UI elements in a TreeView control.
- View all properties, dependency properties and attached properties and non-public fields of any element or sub-element in the UI tree.
- Editing of properties in all project types
- Multi-level drilling into objects and child objects, viewing any of their data.
- Breadcrumb type navigation when drilling into child objects.
- For properties that are `IEnumerable`, view all of their data. The collection data is grouped in an expandable and collapsible region.
    - Data can be viewed in the grid with object properties which allows drilling into child object properties.
    - Data can be viewed in a separate grid using the Mole Collection Viewer.
    - View heterogeneous collections of data in one or more grid, where each grid displays the values of objects with the same type.
- Properties that you have defined as your favorites are grouped together and pinned at the top of the grid.
- View an image which represents the element being visualized (whenever possible).
- View run-time XAML in an HTML viewer for an element (whenever possible).
- Persisted application state and reload of settings from a previous Mole session. ('Who ever heard of a Visualizer with saved settings?')
- Search for properties/fields and specific data values in various ways.
- Configure which columns to display.
- Single click selection. No double clicking is required anywhere in Mole.
- Mole is contained in one DLL, so it is very easy to deploy.
- Mole does not use any registry settings.

Mole has VS2005 and VS2008 builds and Mole has been tested on x32 and x64, Vista and XP systems.

Lightning Fast Performance

We were fanatically obsessed with improving the speed of Mole; from how long it takes to initially load the Window, to how long it takes for an element’s properties to appear in the grid. This was a very high priority because most developers simply will not use a debugging tool which slows them down. So it had to be lightning fast. Period. To speed things up we did many things, such as:

- The initial loading of the visual tree is done on a background worker thread while the form is loading.
- The entire application uses lazy-loading of data and data caching to prevent unnecessary performance bottlenecks.
- A `TreeView` subclass, called `MoleTreeView`, performs lazy-loading of its nodes so that we only create the nodes which must exist at any point in time. Josh is the Master recursive programmer!
- The data transfer objects, which are sent via remoting between the debuggee and debugger processes, implement custom serialization to avoid using the default reflection-based algorithm. This also allows us to only serialize the fields which must be persisted at that point in time.

- Every single line of code has been optimized by using .NET coding best practices. Additionally, Andrew Smith really helped out by showing us alternatives to reflection whenever possible.

Mole Visual Studio Visualizer and XBAPs

One big advantage Mole has over some other tools, is that you can use it to debug XBAP applications. Here is what you need to do to enable that:

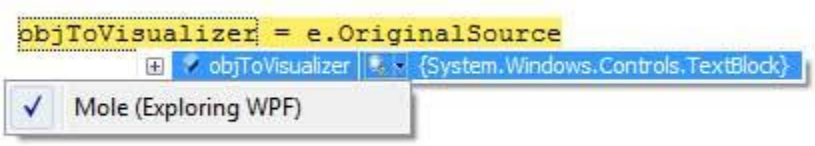
1. Right-click on the XBAP project in Solution Explorer and open the Properties page.
2. Under the Security tab select the "This is a full trust application" RadioButton.
3. Be sure to set it back later on to partial trust in order to properly test deployment.

Thanks to CodeProject member "ivolved" for this tip!

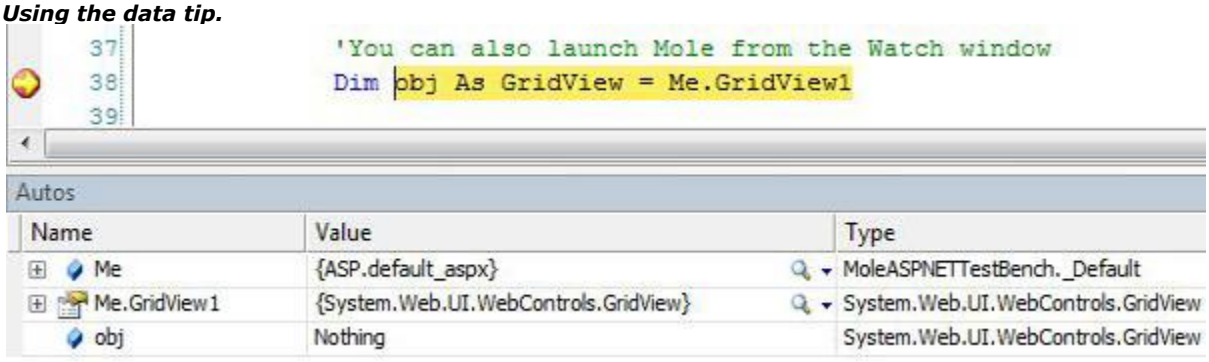
Using Visualizers

Visualizers are very easy to start. When debugging and at a breakpoint, hover your mouse over the object you want to visualize. If a visualizer is installed on your system which can visualize that type of object, the little magnifying glass will appear in the datatip. You can also start a visualizer from the Watch Window.

You can click the magnifying glass to open the default visualizer for that object type. Or you can click the down arrow and select a visualizer to use. The check mark indicates the last visualizer used for the selected object type.

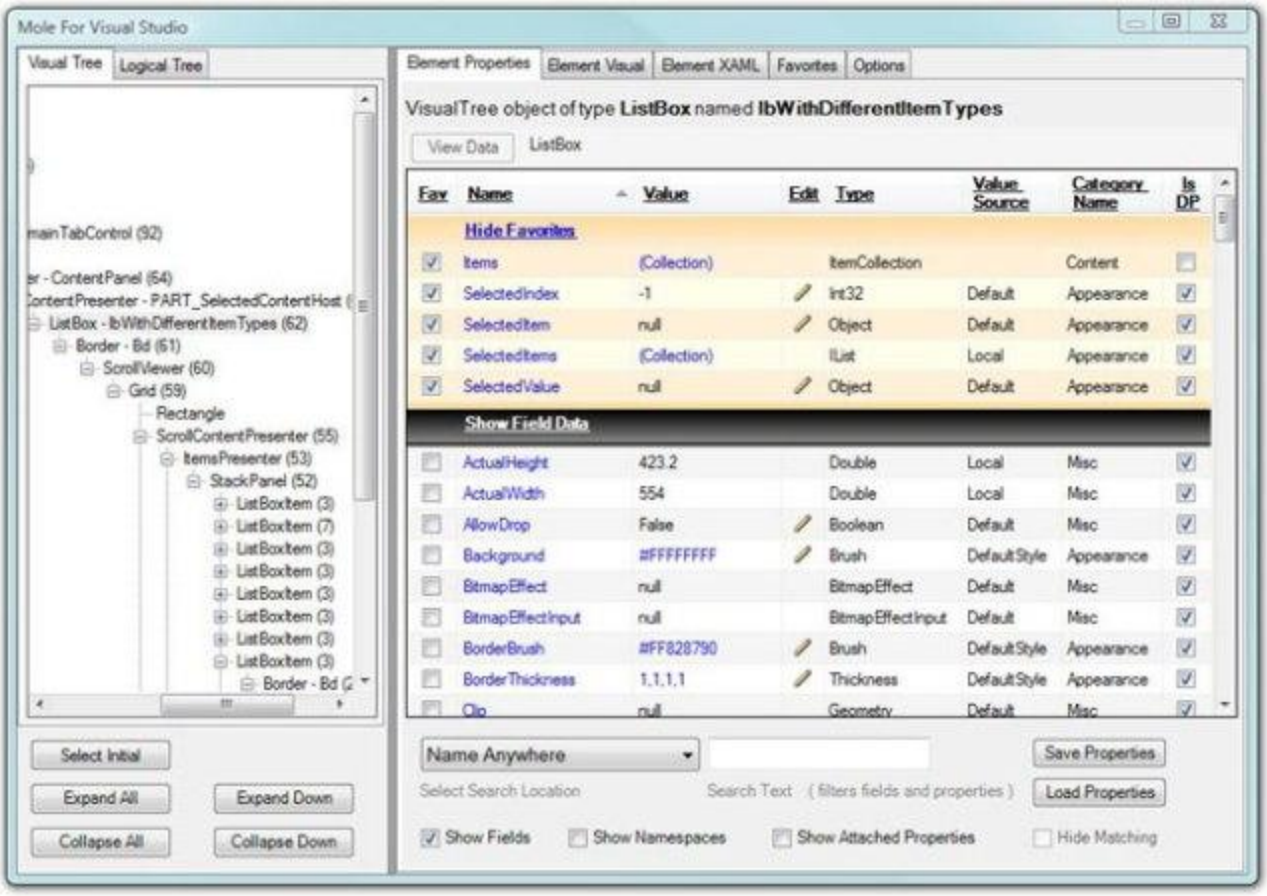


Using the data tip.



Using the Autos Watch Window. Notice the same magnifying glass in the Value column.

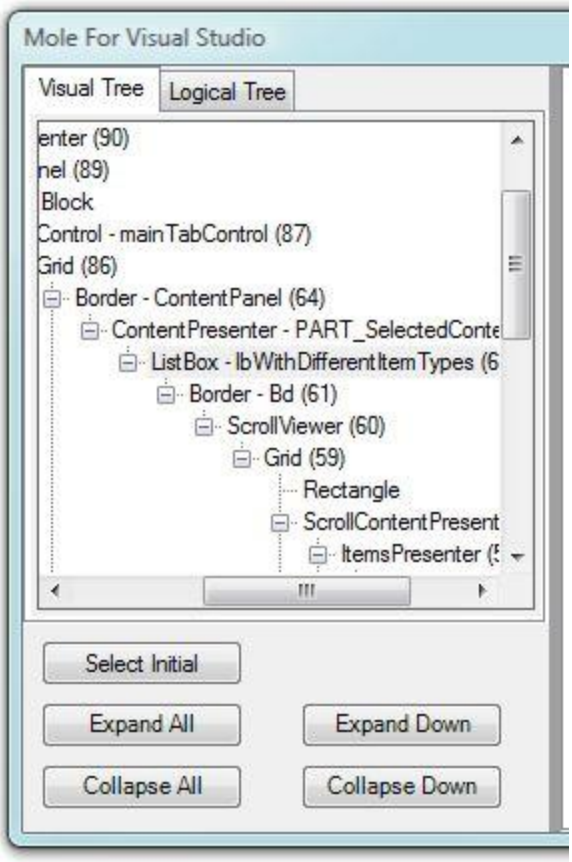
Mole's UI



When Mole opens it looks like the above image. On load, Mole configures and positions itself the same way you closed it during the previous session. Just about everything in a Mole session is persisted and restored without using the registry.

Mole's UI can change depending on which object you have selected in the Visual Studio debugger. For example, a WinForms application does not have a visual tree, logical tree, or XAML. So these features are turned off. The same logic applies to ASP.NET, DataSets, DataTables, and class objects visualized in Mole. Additionally, the "Show Attached Properties" CheckBox, "Value Source" column, and "Is DP" column will only be available in applications that support them.

Mole's Gardens



The `TabControl` on the left of the UI contains one or two `TreeView` controls depending on the type of object you are visualizing. These `TreeViews` are Mole's "gardens". One contains the tree of UI elements, the other the logical tree of a WPF application being debugged. The UI element tree `TreeView` initially selects the object you selected in Visual Studio. If you select a single object that is not part of a UI element tree, that object could be the only object displayed in the tree view on the left. This does not limit the amount of drilling and discovery available to the developer using the properties grid.

The `TreeNode` (actually `MoleTreeNode`) text is formatted as follows:

Element Type - Element Name (Count Of Descendants)

Some objects were not assigned names in the program so they do not display here and some nodes do not have any child nodes associated with them.

Elements are displayed in one of four colors. White is an unselected element. Light Gray is the selected element when the `TreeView` has lost focus. Blue is the selected element when the `TreeView` has focus. Green is the initially selected object you moused over in Visual Studio.

The dark gray 4 pixel line separating the two `TabControls` is a `GridSplitter`. You may size the two regions. The size of the two panels is persisted when the size is changed so that the panels will be displayed the same when Mole is reopened.

Select Initial button will select the object that you initially selected in Visual Studio.

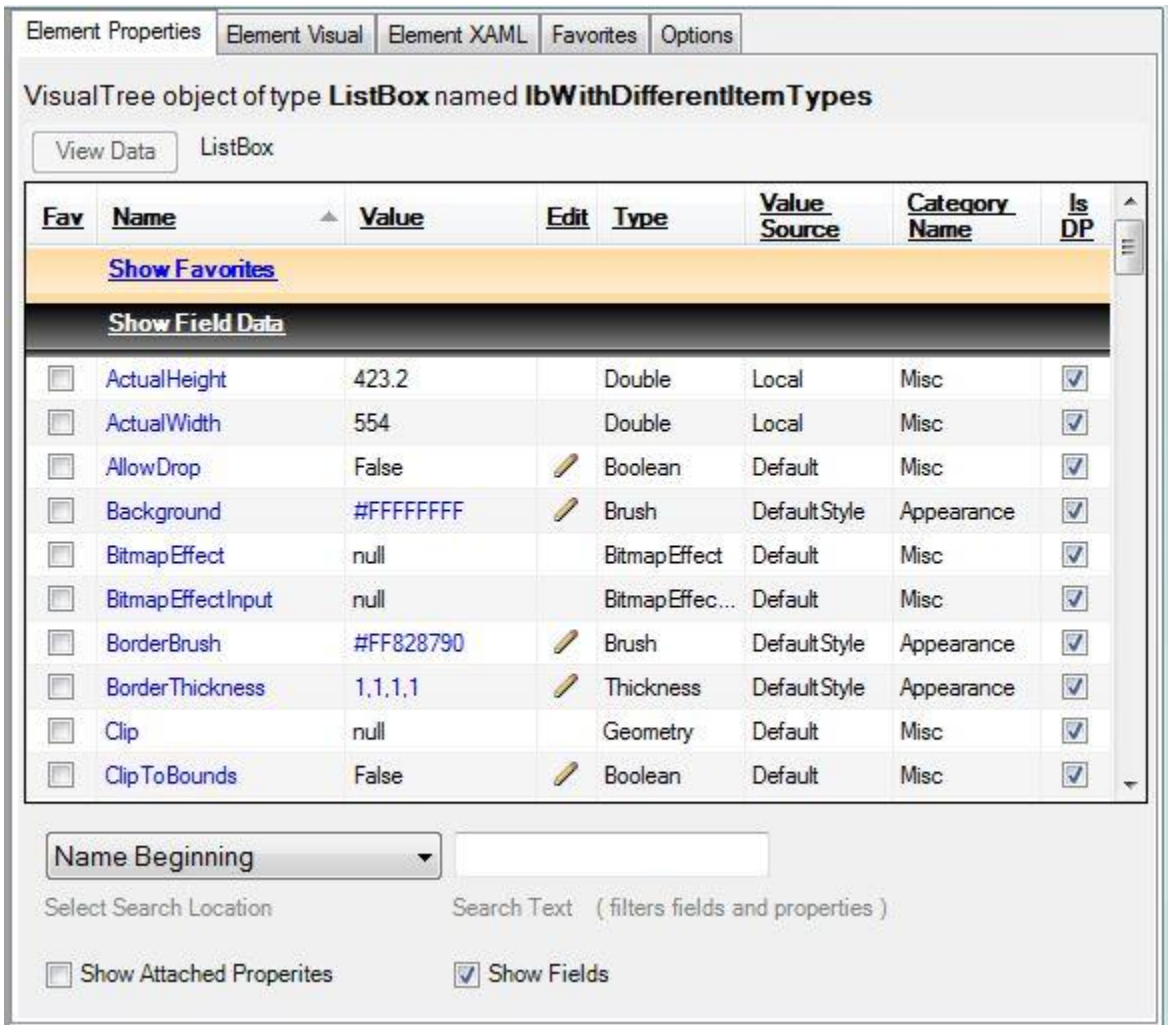
Expand All button will expand all `TreeNodes` in the entire `TreeView`.

Collapse All button will collapse all `TreeNodes` in the entire `TreeView`

Expand Down button will expand all `TreeNodes` that are children of the currently selected `TreeNode`.

Collapse Down button will collapse all `TreeNodes` that are children of the currently selected `TreeNode`.

Element Properties - The Moloscope



Mole's Moloscope employs Max-O-Drilling technology, allowing the visualizer to drill back into the managed heap of another process to extract properties and data structures associated with the element being browsed from either the UI elements tree or logical tree. It is interesting to note that visualizers execute in a separate process from the data they are

visualizing. All data must be piped through remoting, which is mostly handled by Visual Studio. The Max-O-Drilling technology we came up with is responsible for locating and maintaining references to objects on the managed heap that are being drilled into by the user.

The label below the Element Properties TabPage heading is formatted as follows:

[Drilling into] [Object Type] [Object Name]

If you are drilling, the "Drilling into" text will be displayed. Some nodes were not assigned names in the program being debugged, so it won't display here.

Properties Grid

The Element Properties grid displays each of the selected item's properties. Columns with underlined header text can be sorted by clicking on the column header. Like many features in Mole, the selected column sort is persisted between Mole debugging sessions.

The Favorites column (Fav) indicates if this property is one of your favorites for the selected type. You may select or deselect favorites by clicking the leftmost `CheckBox` in the row.

The property names are hyperlinks. When clicked, they open Google.com using your default browser and set the query string to search for the property name and selected item type.

The property values are hyperlinks if the value can be drilled into.

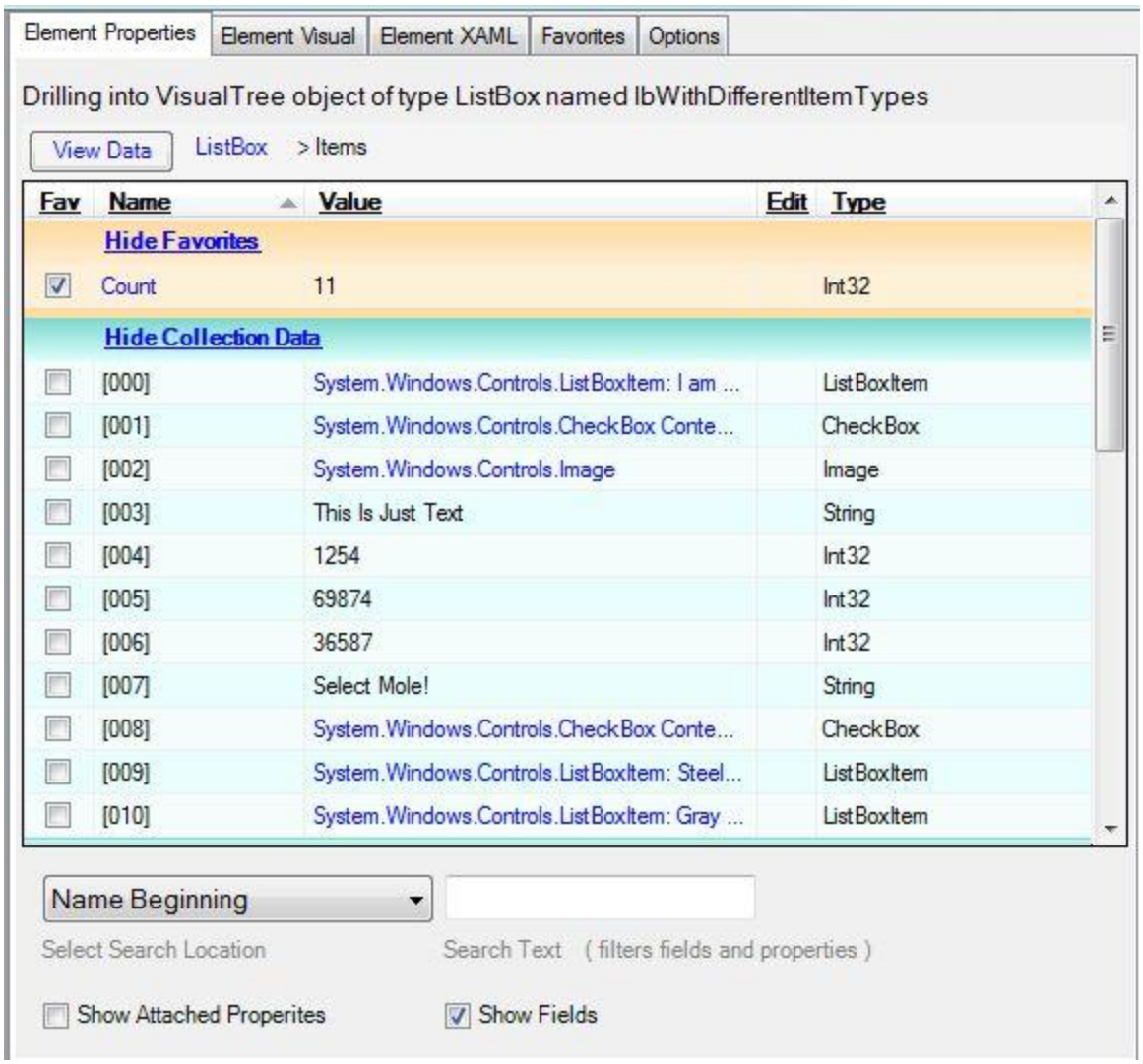
If the property can be edited, the pencil icon will be displayed. Please see the Editing section below for full details.

Value Source identifies the source for the property value. Only WPF dependency properties have a value source. If you are not familiar with the concept of value sources, please read up on the topic at MSDN BaseValueSource Enumeration.

The Category Name column is the name of the category assigned to it on the property declaration with the `Sysem.ComponentModel.Category` attribute.

The rightmost column, Is DP, identifies whether this property is a dependency property or not. This column is removed when debugging applications that do not have dependency properties, like WinForms and ASP.NET.

The below images will walk you through the drilling process:



I have used the Options tab to hide several columns. The developer has selected the node in the `TreeView` which represents a `ListBox` object (not pictured here). After selecting the `ListBox` node, the developer drilled into the `Items` property of that `ListBox` object. Notice how the breadcrumb trails just above the grid documents where you are in the drilling process.

Notice that the Value column for some of the items is a hyperlink. This is how you can tell when you can drill into an object. Just click on these hyperlinks and you will conduct a drilling operation.

When drilling into a property that is `IEnumerable`, the actual data is displayed in a Collection Data region. This region, like the Favorites region can be collapsed and expanded. The [000], [001] is the index for the data in the `IEnumerable` collection.

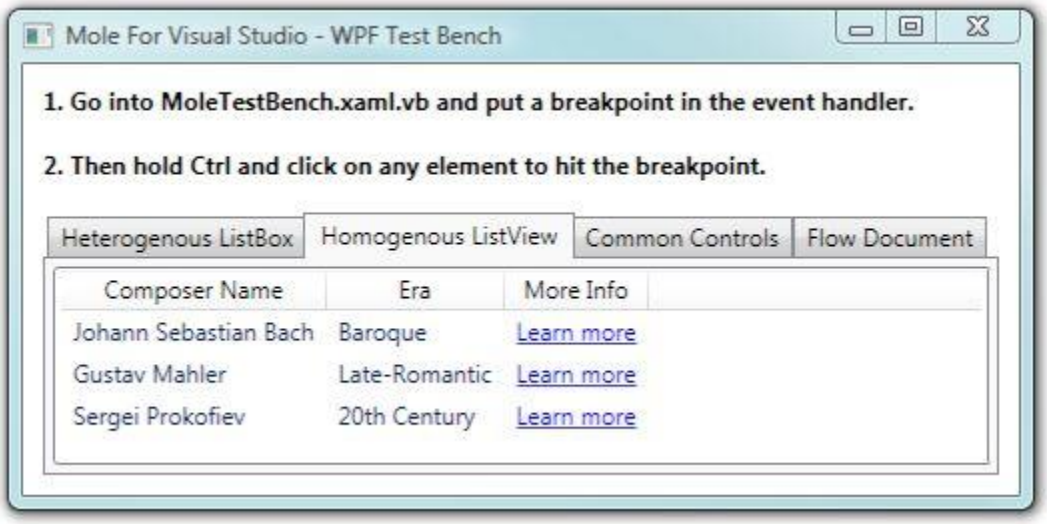
To truly appreciate what is going on here is a mind expanding experience. The visualizer UI does not have any references to any data in the managed heap in the other process. Also after drilling down, the UI element tree or logical tree we are viewing has no knowledge of

what is being drilled into. The data I'm referring to is also located in another processes memory space and not Mole's!

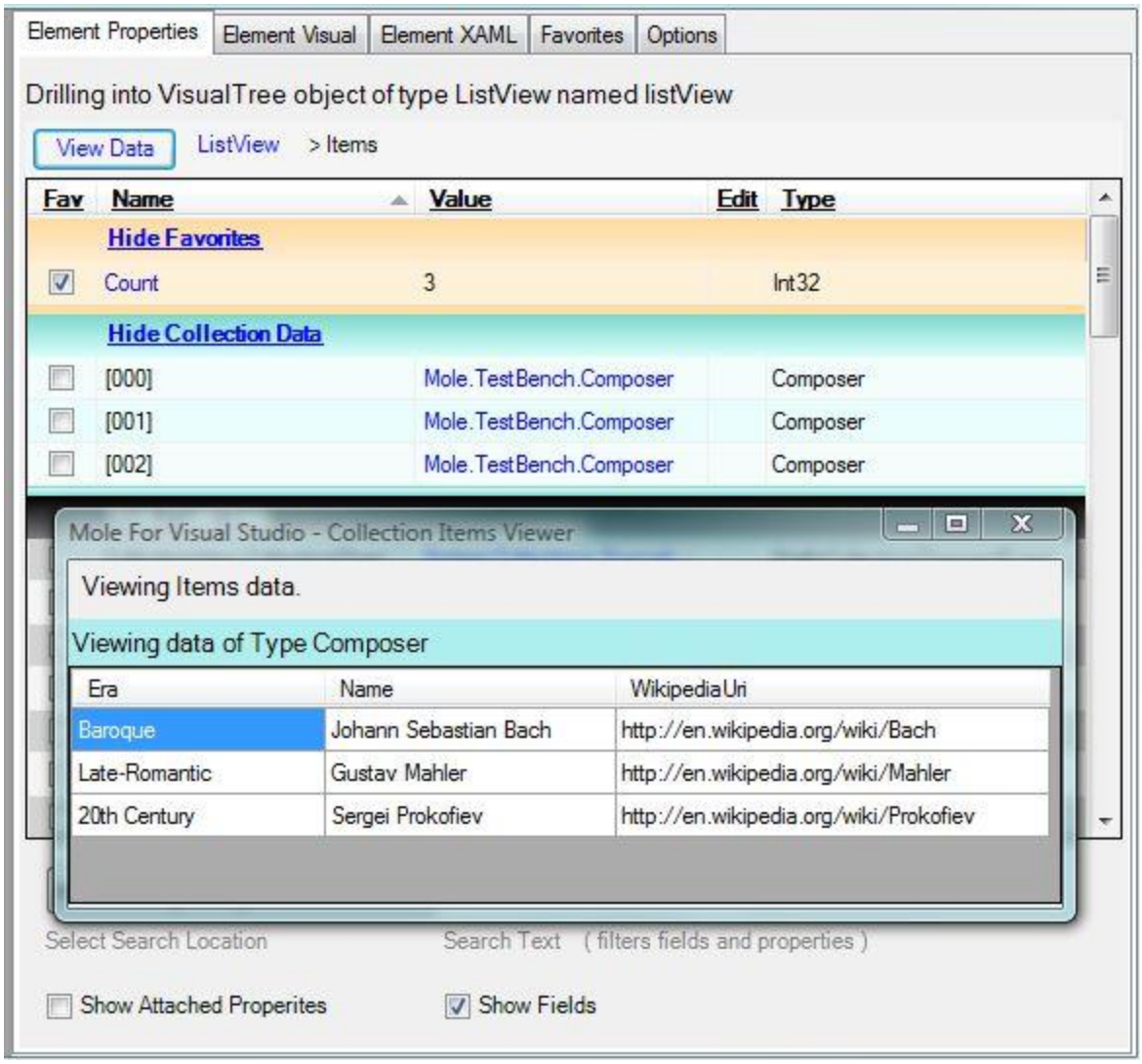
Max-O-Drilling technology enables this to happen. It took the better part of a day for me to architect and code this feature. Most of this feature resides in two places. The `MoleCrumbs` control and a dictionary which maintains the state of a drilling operation. Another point to further expand your mind, is that the Moloscope is just a bunch of strings and one integer, yet, Mole is able to drill into the depths of the .NET Framework.

After my first pass, the feature was operating at 98%. Not good. Josh banged on the Moloscope with a 50lbs hammer and found a few cracks. We were not happy with some false readings that when the user drilled into them, the data was not there. You have to remember that drilling into objects is a two phase process. The first phase is when a list of properties are being returned back to Mole's UI from the data source. It's during this pass, that we must determine if a property can be drilled into, without actually drilling into it. That would simply take too long and slow Mole down. We all spent a good bit of time trying to find a solution. Bottom line, Andrew figured out the best method and that is the method Mole uses. You can view Andrew's solution by searching for the `IsDrillableTest` method in *MoleVisualizerObjectSource.vb*. This method has two overloads.

There is a really funny story of how this drilling business all got started and is representative of how Mole was developed. One day Josh and I will have to post it somewhere.



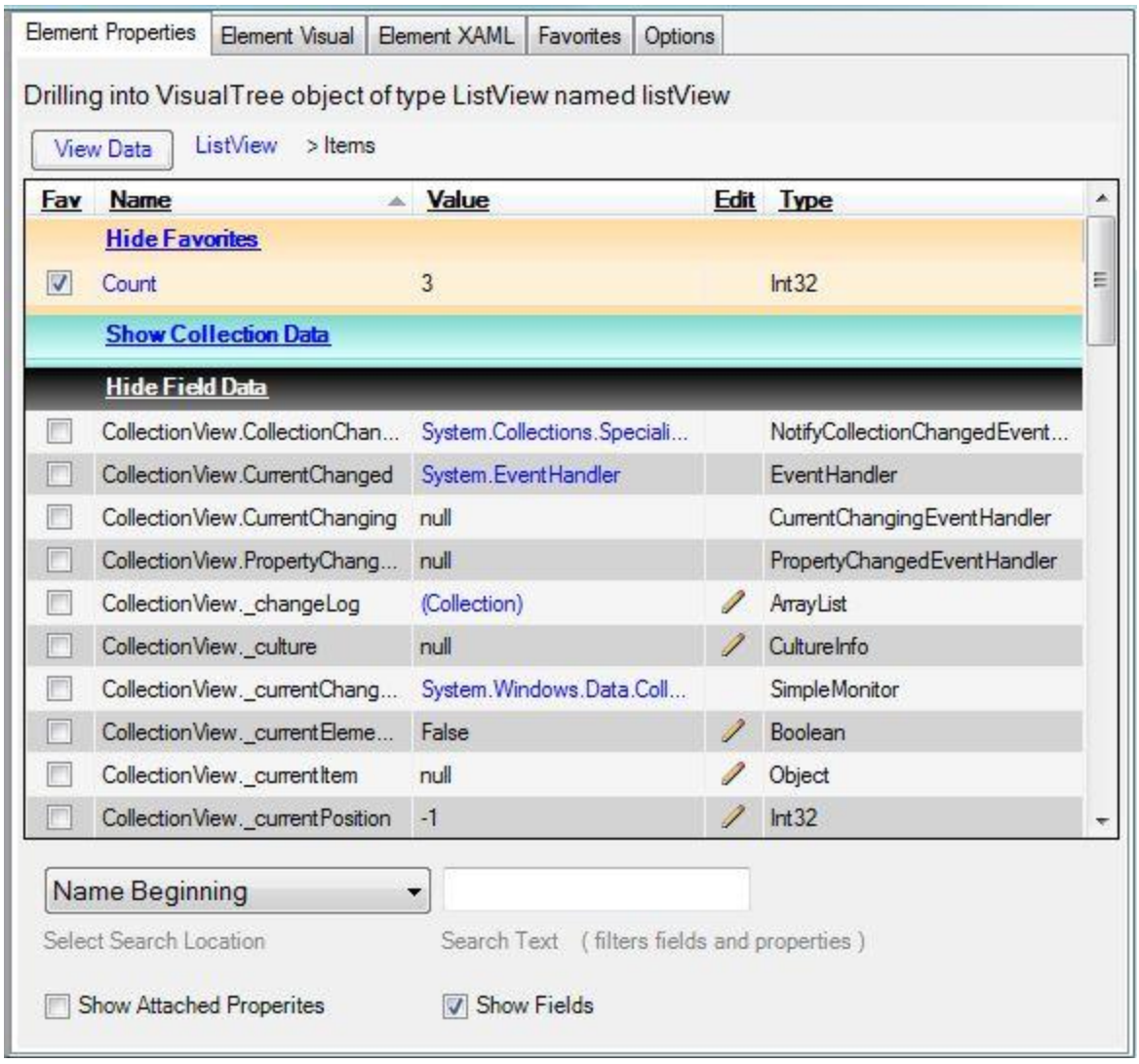
This image is from the WPF Test Bench program Josh wrote for Mole. This is an application and not Mole. Displayed is a collection of the `Composer` class.



This image shows how Mole drilled into the `ListBox` items collection and displays the same data in a grid inside Mole.

I really wish the Visual Studio debugger could display data inside collections with just one click without having to type so much in the debugger to view collection data. Maybe Visual Studio 2010, please?

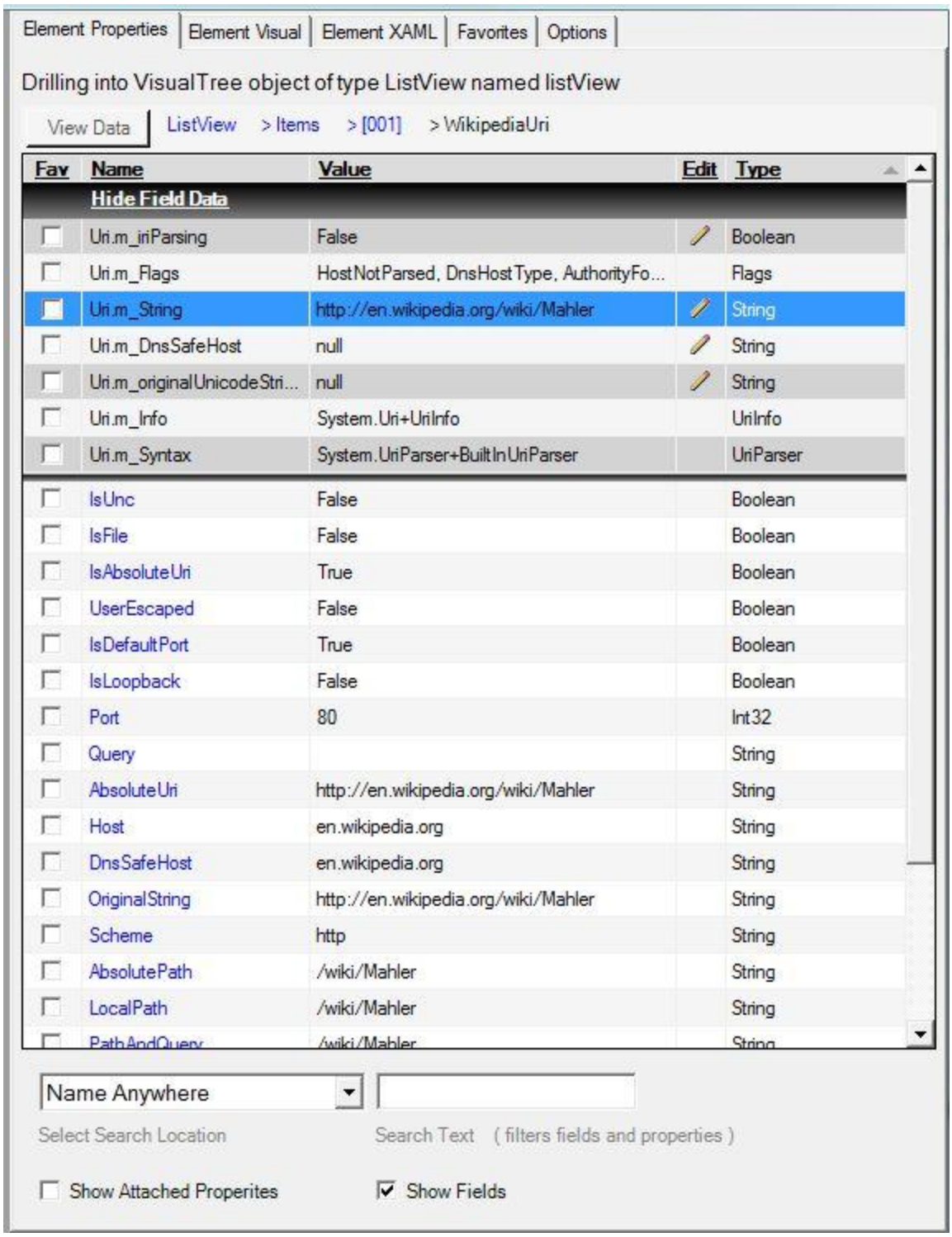
This image shows the collection data in multiple data tables. Since the collection contains data that is of different `Types`, a separate `DataGridView` is created for each `Type` of data.



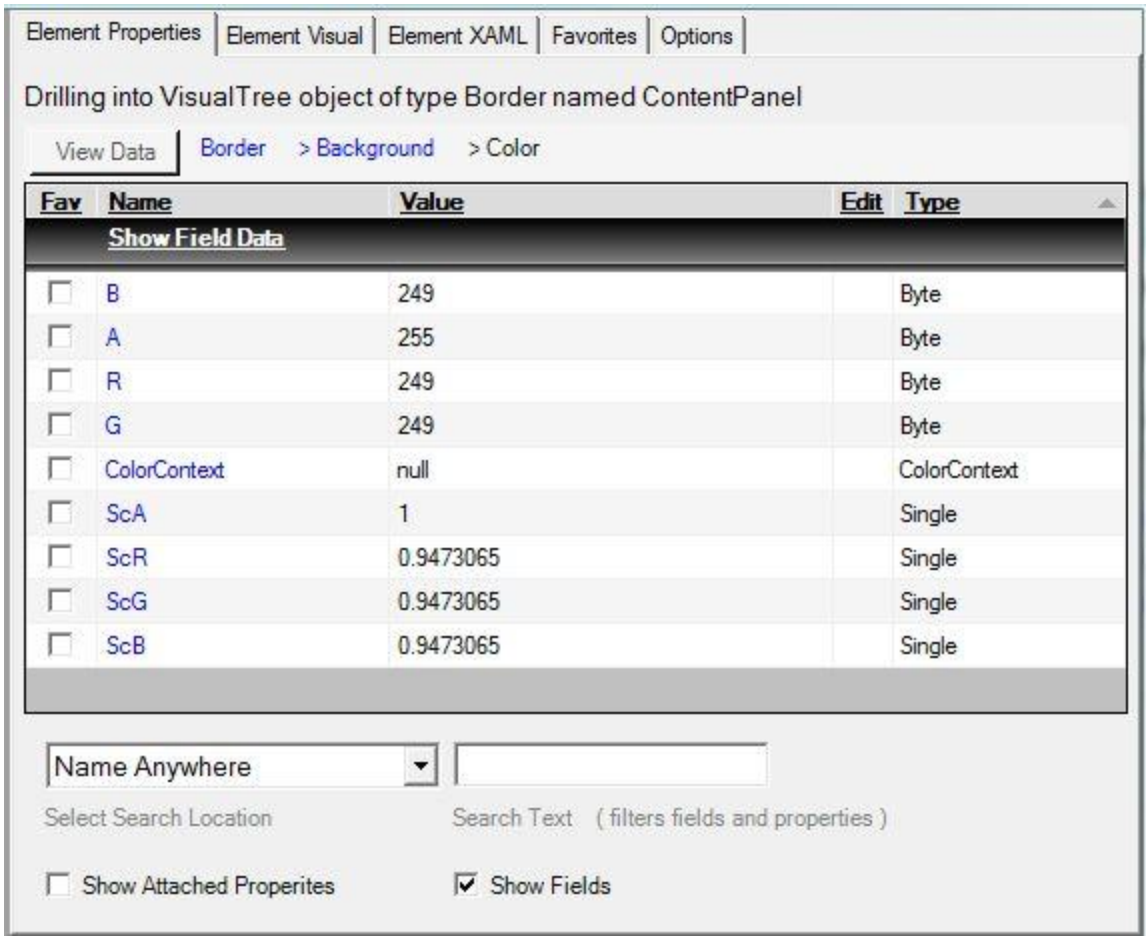
This image shows what the screen will look like after collapsing the Collection Data region. The standard WinForms `DataGrid` does not support grouping and collapsing regions so we added the feature. Dive into Mole's code and use this feature in your applications.

Mole Can Help You Learn Or Teach .NET

Mole really shines in the next two images! .NET instructors can use this tool in the classroom to show your students how everything fits together in .NET.



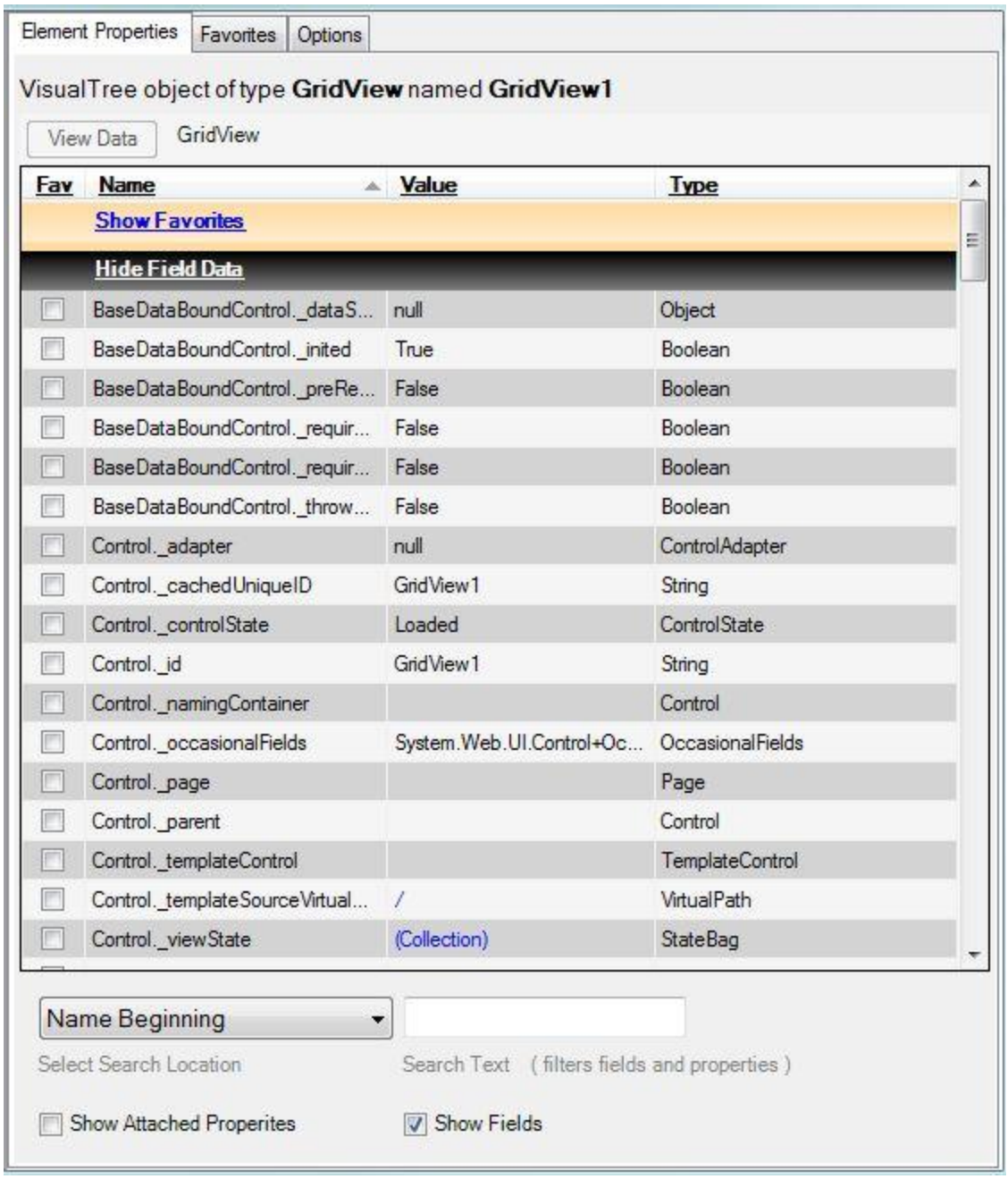
Here the developer has drilled 4 layers deep to a control that exposes `URI` properties. You can see all the `URI` properties in context. The property is not just some abstract name, but a developer can see how the `URI` properties relate to each other. Notice that Mole v4 allows you to edit private members of the URI property directly.



In the image above, the developer has drilled three layers deep. You can see a `Color` object broken down into its constituent pieces.

I've played around with this a lot and it's so cool to be able to drill around the .NET Framework classes, and your own classes, too!

Mole Black Ops



In the above image, an ASP.NET `GridView` has been selected in the UI element `TreeView` control. The breadcrumb points this out. We can see the Favorites section has been collapsed. Below the Favorites section is the new Fields (Black Ops) section. I wanted to call the region "Black Ops" but Josh talk me out of it. He is right, we needed to use standard naming for our UI. We have had downloads from countries all over the world, so keeping the lingo off the application can help our fellow developers translate and use Mole. However, like almost anything in Mole you can customize your version to look and say what you want. Heck, it could even play sounds or music for you!

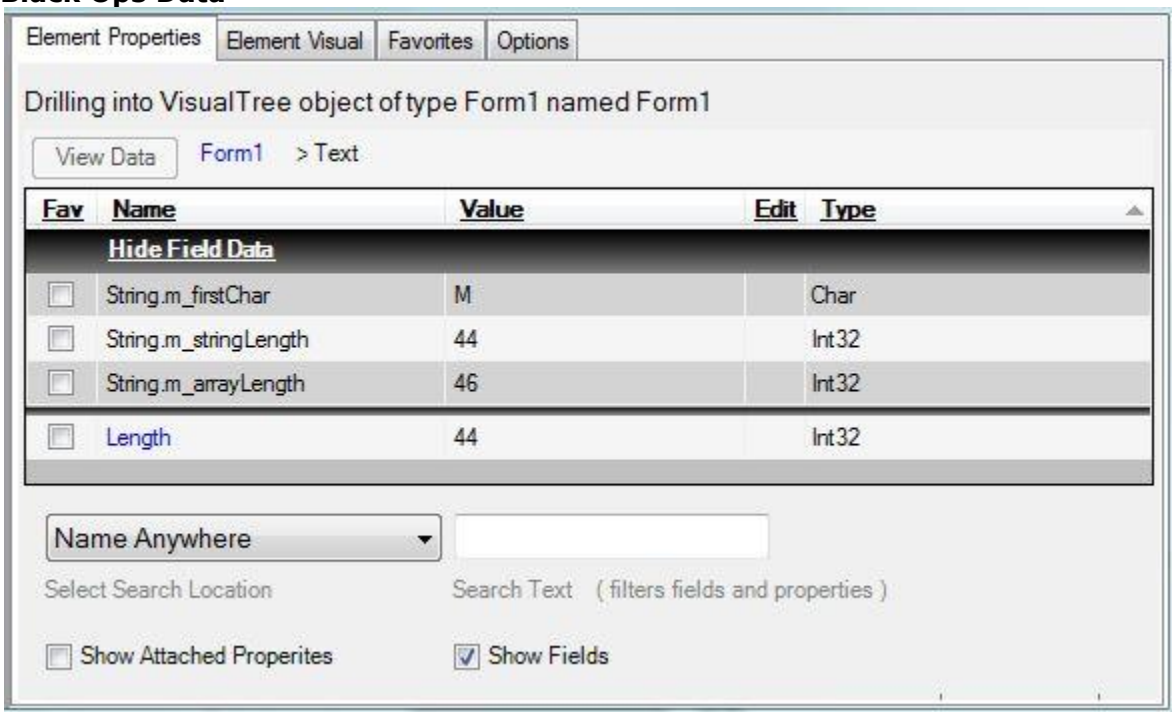
A BIG tip for all the ASP.NET developers. You can use Mole to discover the funky names of your child controls that ASP.NET assigns them. Just debug your ASP.NET page in Mole and look at the `ID`, `_id`, `Name` and `_name` properties. No more guessing.

Fields (Black Ops) Section

- Region supports collapse/expand.
- Field items may be assigned to the Favorites section. If assigned to Favorites, that field will be displayed along with your favorite properties.
- Field items have their Category Name set to ({field attribute} Field). {field attribute} is from the `FieldInfo.Attributes` that you can read about in this MSDN FieldInfo.Attributes article.
- Field items are all non-public fields and non-public properties.
- Field items support drilling, just like public properties.
- Google search has been disabled for all Field items since the search would return strange results.

Below the Search TextBox, there is a new "Show Fields" `CheckBox` which allows the Fields section to not be displayed if desired. This setting is persisted between Mole sessions.

Black Ops Data



Some would ask, why show private or protected members? Allow me to list several reasons. (be aware that you can easily turn this feature off by un-checking the above Show Fields `CheckBox`.)

In the above image, I have selected a `String` from the logical tree. Notice the members of the `String` class. Mole Black Ops has revealed some internal information that I never knew about before I started drilling around using Black Ops. I spent about an hour just drilling around and checking out WPF's classes using this feature.

The real reason I wanted this feature was to be able to view private and protected members that I have defined in my own classes. Now when I'm using Mole, I can view and drill into the module level private and protected members of classes I have written. This provides a more in-depth view of my own data.

Property Editing

Background

With the release of Mole v4 comes the most requested feature, Property Editing. Andrew really wanted to incorporate editing into Mole, so he and Karl dove off the cliff and into the property editing abyss. This change impacted many areas of Mole and required much testing by the entire team. When used correctly, this feature can be a great asset to WPF and WinForm developers. We will give some usage examples below.

Visualizers allow the replacement of the object being visualized. Code Project MVP, Rama Krishna Vavilala brings this feature out in his XAML Debugger Visualizer for WPF article. He also explains the issues with replacing objects being visualized.

Since the default object replacement capabilities of visualizers are limited to the original object being visualized, we were not able to use this technique. Instead Mole edits the values in the heap directly.

As you may have already noticed in the above screen shots, not every property can be edited. Andrew wrote this explanation up for us to understand how editing in Mole works.

The editing functionality makes use of .NET's TypeConverter infrastructure. The type converter is used to convert the entered text into the appropriate type for the property/field. Therefore, only properties and fields whose associated `TypeConverter` can be used to convert a value from a string can be edited. The only exceptions to this are properties/fields of type `Object`. In that case, it is assumed that the property/field may accept a value of type `string` and no conversion is done.

In addition, the properties/fields of certain objects are not allowed to be edited:

- Objects that derived from `Freezable` whose `IsFrozen` is `true`.
- `ValueTypes` - This is necessary since the property/field that is referencing the value type is referencing a copy of the value type and not the same copy of the value that would be edited.
- Any type that has a public `Boolean` property named `IsSealed` (e.g. `Style`, `SetterBase`, `FrameworkTemplate`, etc.) where that property is `true`.

We have provided a good number of type editors (see below) that assist developers in editing properties. In order to provide the most flexibility and actually deliver this version of Mole, we allow the editing of objects that probably should not be edited. In order to exclude every class that "should not" be edited would take a lot more programming time, cause Mole to run slower and in the end we would see very little payback for the effort. You must also consider that Mole allows editing with every Visual Studio project type, so that "list" of non-editable types would be HUGE.

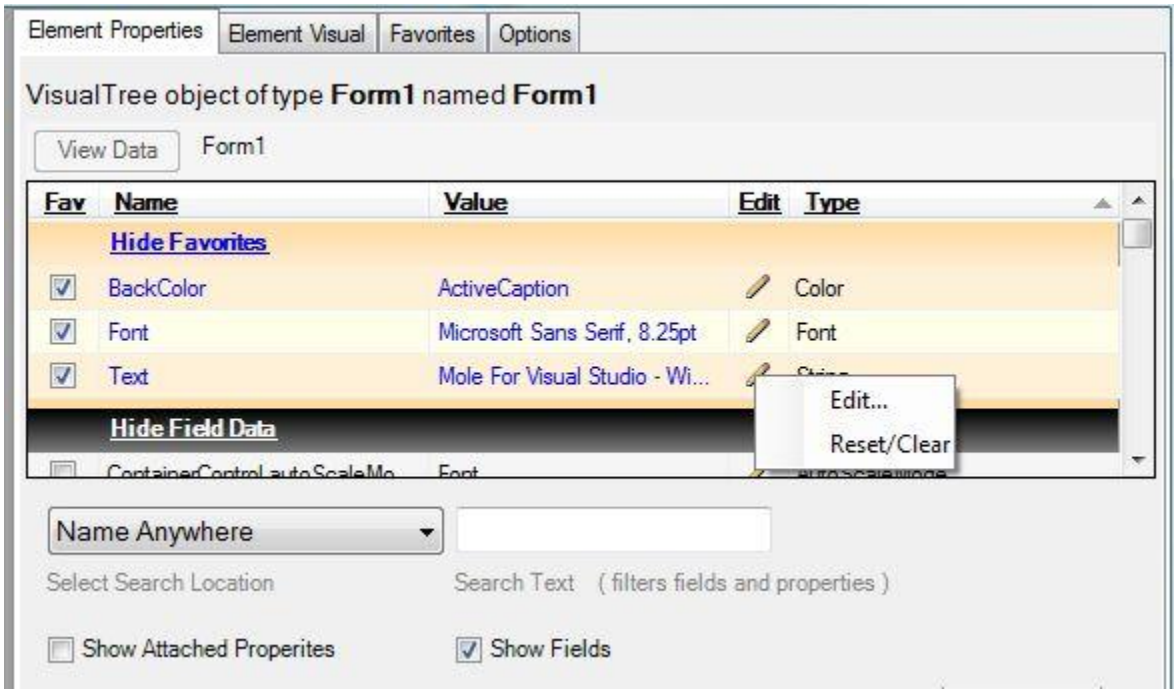
So, please note that a certain amount of common sense is required when editing. The original purpose of the editing feature was to allow developers to adjust their UI at run-time and see the impact of those changes. But like everything in Mole, this feature took on a life of its own and even allows the editing of private members if they meet the above requirements.

Editing Features

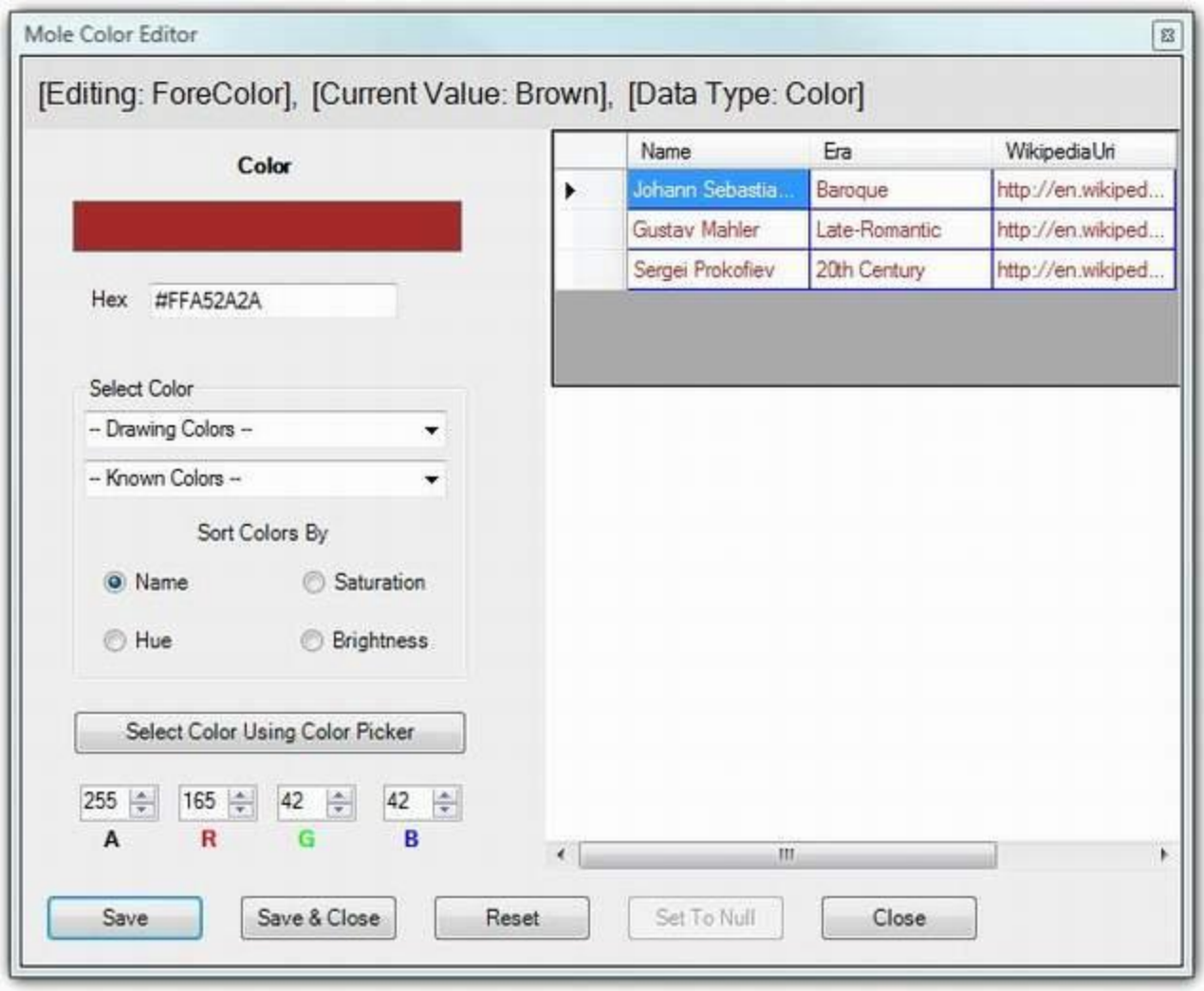
- Eight specialized type editors are provided.
- A real time image is provided next to the editor so that developers can view the effect of their changes without closing the editor.
- If the property value can be set to null (nothing), the editors support this. This includes the generic `Nullable` types.
- If the property value can be reset, the editors support this.
- All edits are logged and when Mole is closed, the edit log is copied to the Clipboard so the developer has a record of what was changed. (Note: Mole does not touch your source code.)
- All editor dialogs save their size and location.

How To Edit

There are two ways to initiate property editing. First you can click the pencil icon and the appropriate Mole Editor dialog will open. Second you can right click the pencil and select "Edit..." or "Reset/Clear." Note the "Reset/Clear" is not always available. This feature is controlled by the `PropertyDescriptor.CanResetValue` method and you can read up on this method here.



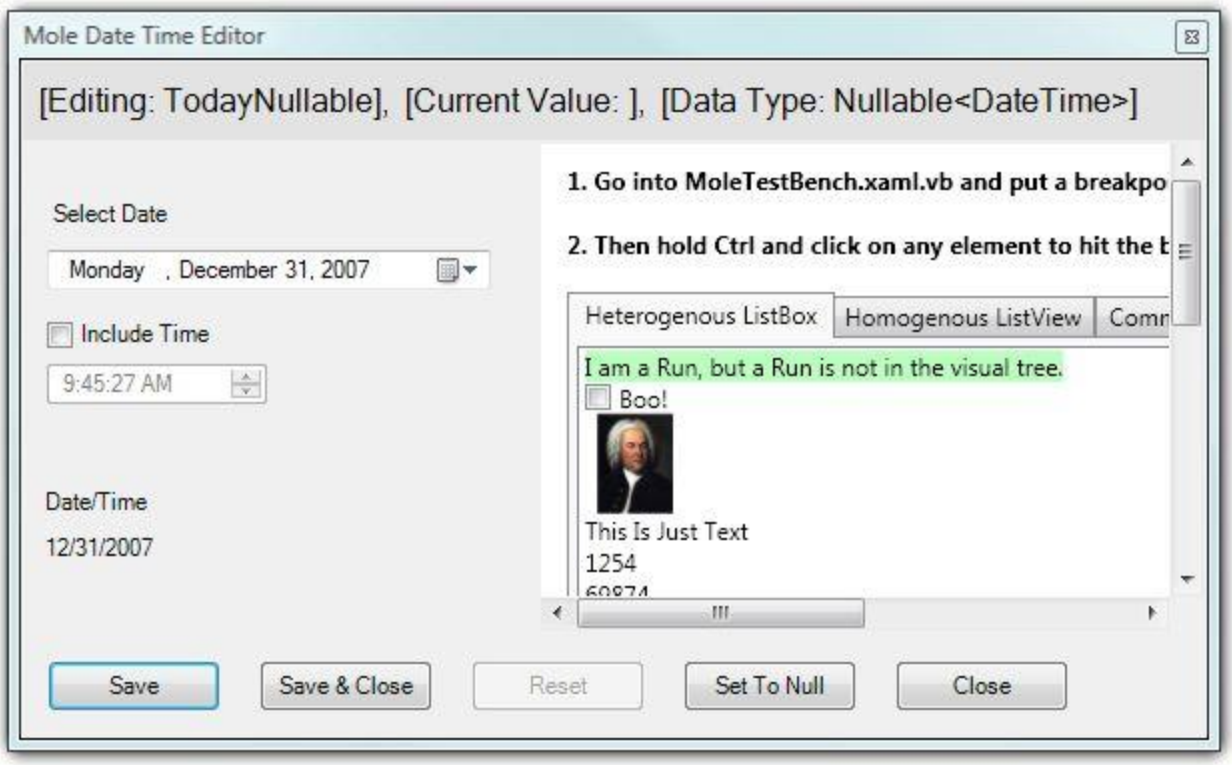
Color Editor



When I first opened the editor, the font text color was black. You can see how the color was changed and then applied to the project being visualized and the color updated without having to close the editor. This is accomplished by selecting a color and pressing the "Save" button.

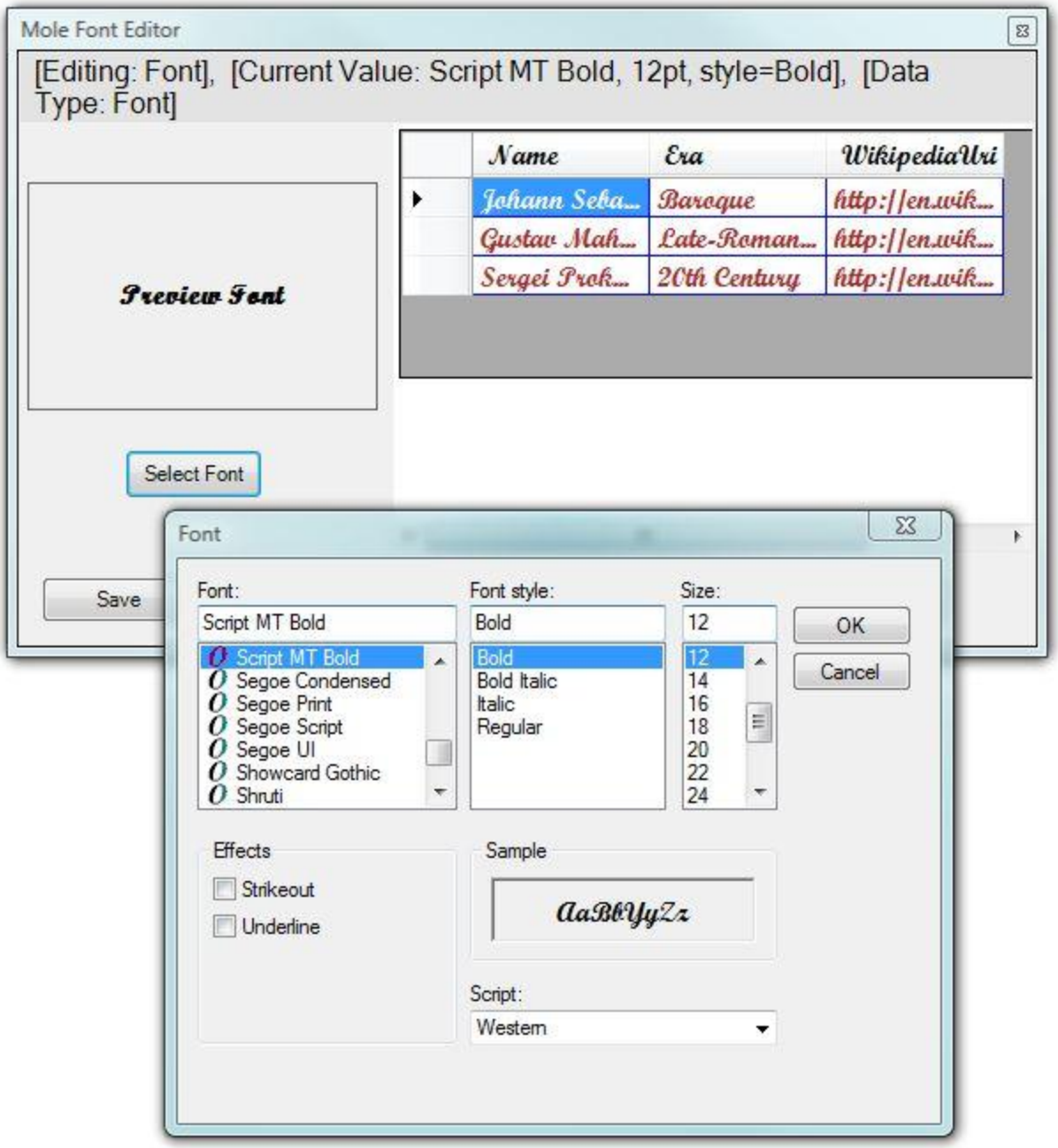
Colors can be selected and/or set in many different ways using this editor. Also, note the color sorting below the `ComboBoxes`. This allows the values in the `ComboBoxes` to be sorted four different ways to make selection just a little easier.

Date Time Editor



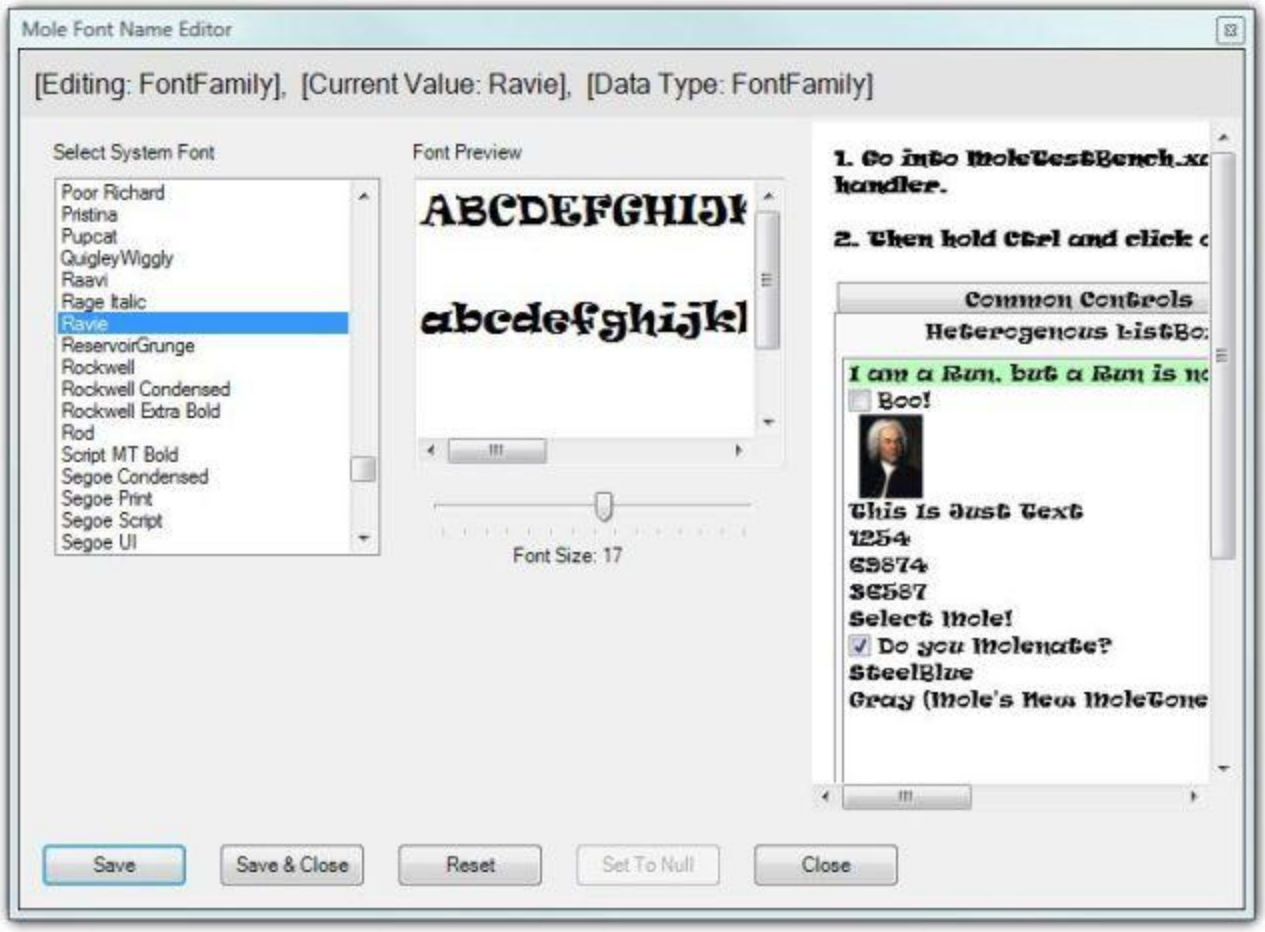
Here the property being edited is a `Nullable(Of DateTime)`. The editor also allows the setting of time if desired.

Font Editor (WinForm)



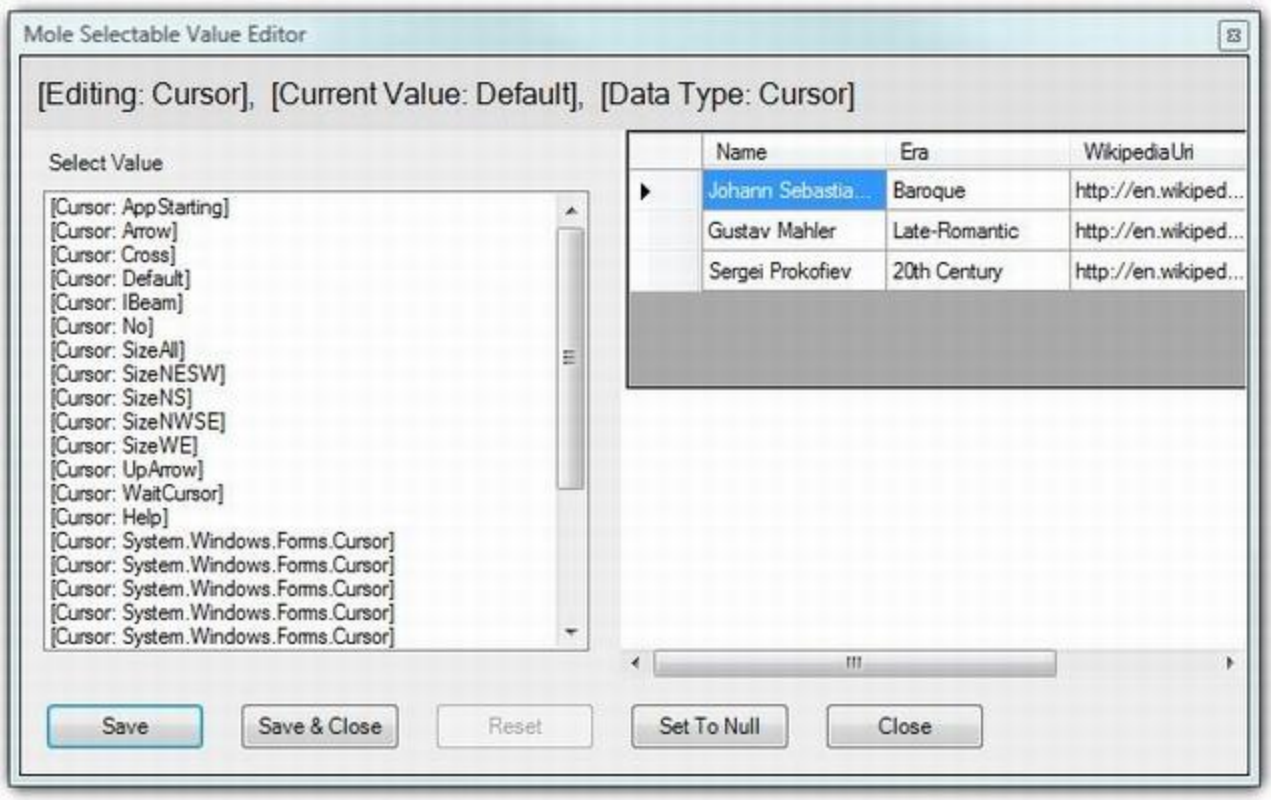
You can see how editing a font value is applied to the project without having to close the editor. After closing the font dialog, the "Preview Font" label is updated. Clicking the Save button actually applies the change to the project object.

Font Name Editor (WPF)



Again, you can see how a font family is selected and set with the changes being applied without having to close the editor. The Font Size track bar is used to adjust the size of the preview only and not the actual font size. WPF breaks all the pieces of a font into separate properties which is different from WinForms. Also notice how font inheritance changes the entire WPF application.

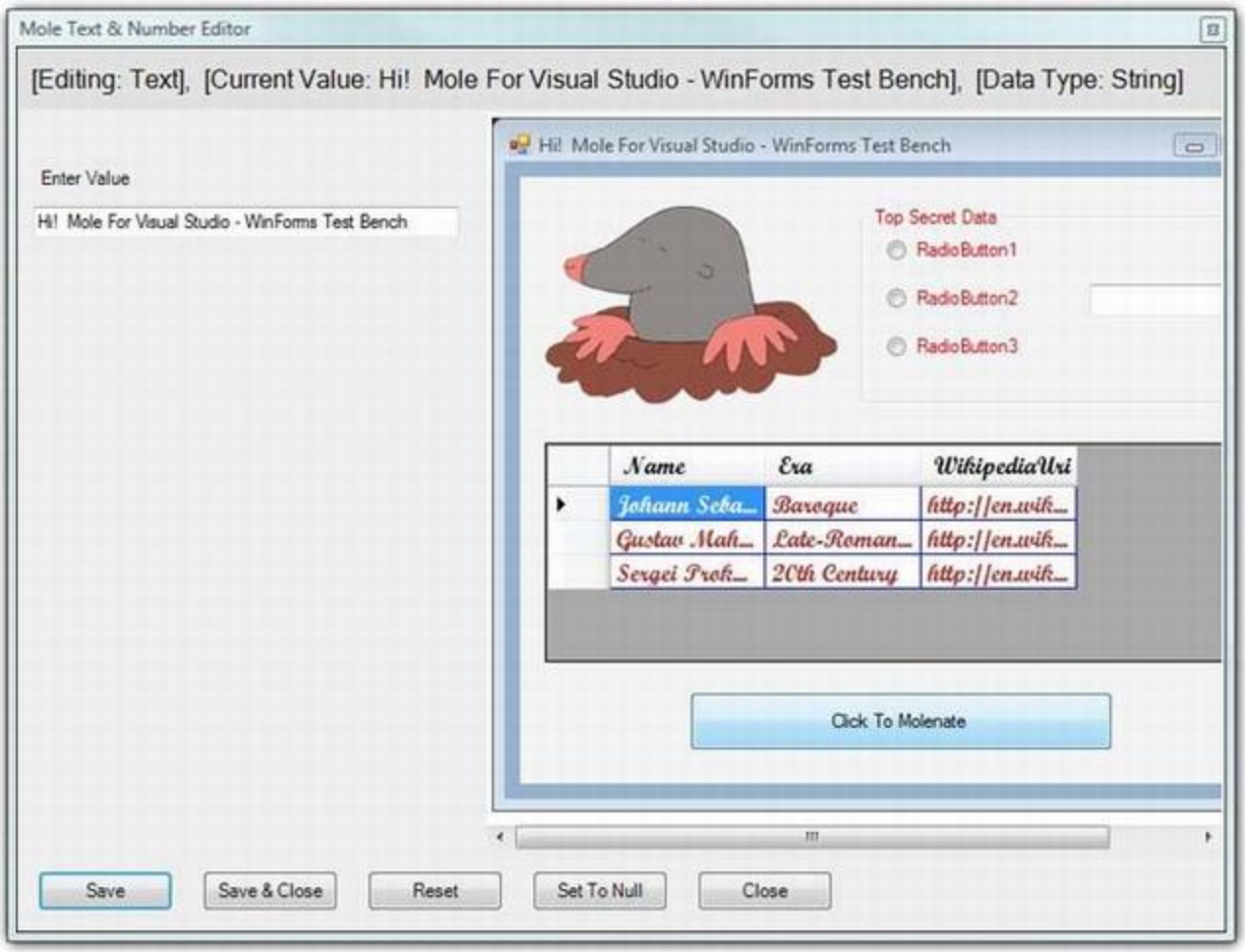
Selectable Value Editor (Enum's, lists, etc.)



Here we can see the mouse cursor property being edited in the Selectable Value Editor.

All enums are edited in this fashion except enums that have the flags attribute. If the enum has a flags attribute and the property value has more than one value assigned to it, Mole won't edit it. We excluded enums with multiple values because the effort would not be worth the payback.

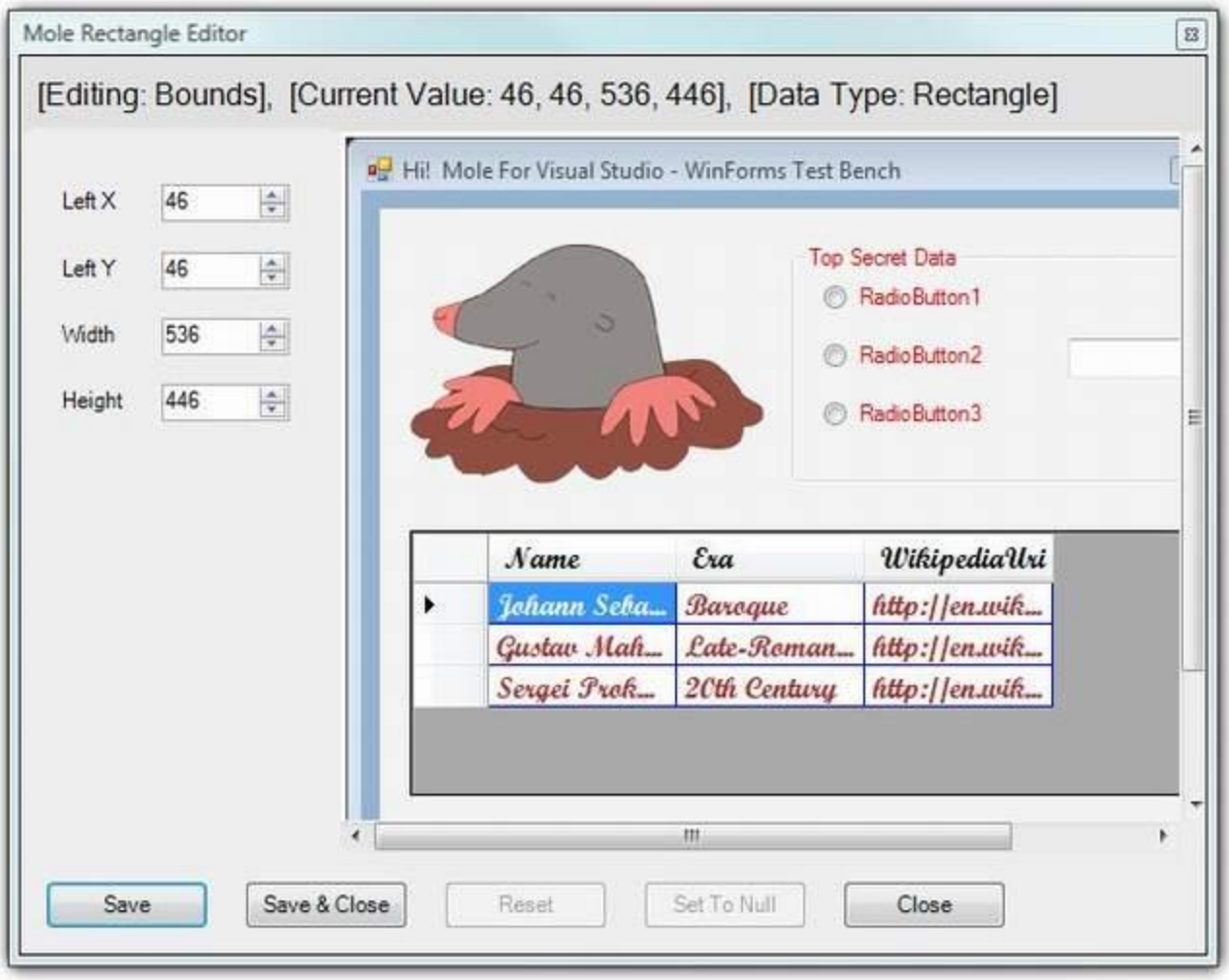
Text Number Editor



In the above, the `Window.Text` property is being edited. Notice that the Title text was changed by pre-pending, "Hi!" to the title. Again, the editor displays the result of the edit without having to close the editor.

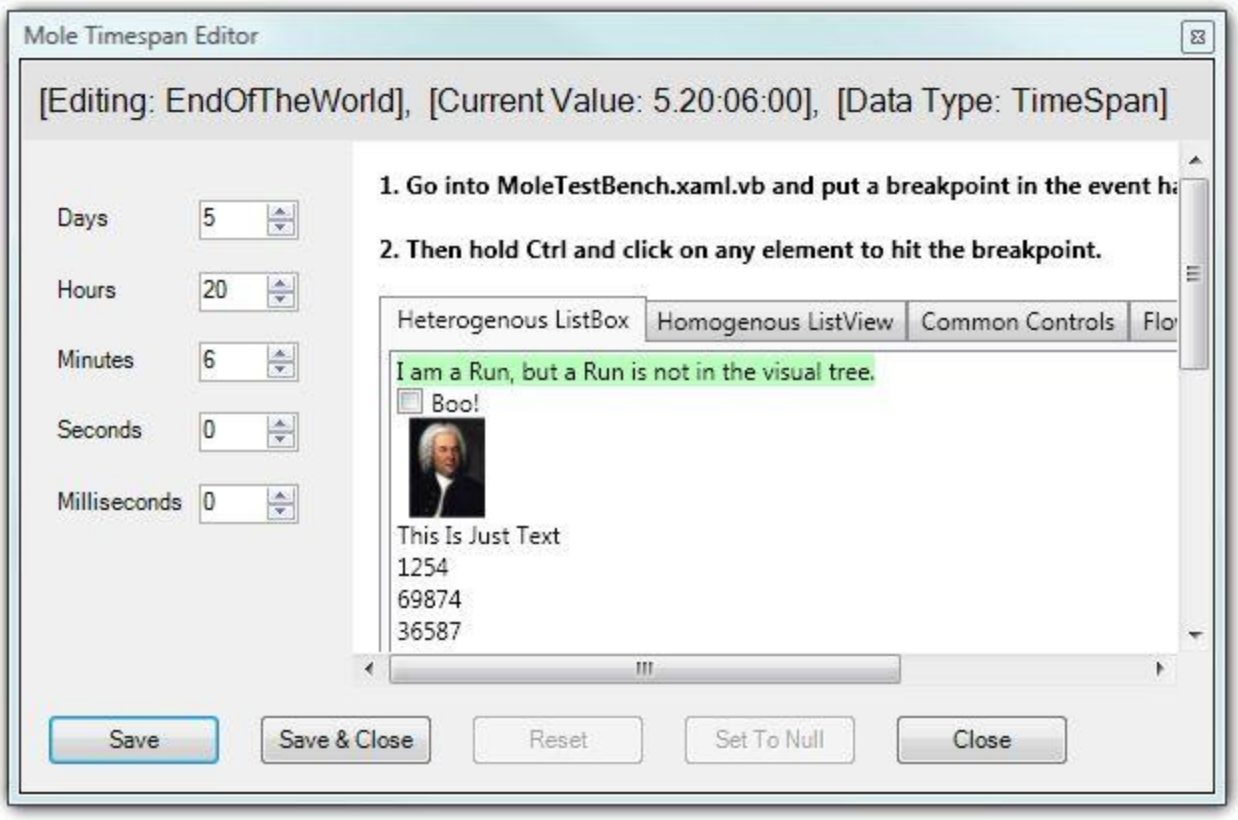
For some WPF properties that are of type `Double`, the text, `Auto` is a valid entry. Mole supports this type of editing.

Thickness Editor (Margins, padding, rectangles, etc.)



This editor handles the rectangles, margins, thickness, etc. properties.

TimeSpan Editor



Mole is editing the EndOfTheWorld `TimeSpan` property.

Logging

```
'------------------------------------------------------
'Tree View Object Name    :
'Tree View Type Full Name: Mole.TestBench.MoleTestBench
'New Value                : WidthAndHeight
'Editor Type              : Int32Value
'Write Operation Type     : WriteValue
'Write Error Message      :


'------------------------------------------------------
'Tree View Object Name    :
'Tree View Type Full Name: Mole.TestBench.MoleTestBench
'New Value                : Hi!  Mole For Visual Studio - WPF Test Bench
'Editor Type              : Text
'Write Operation Type     : WriteValue
'Write Error Message      :
```

Above is what the editing log looks like. After closing Mole, just paste the contents of the clipboard into your code editor to view them.

In order to support developers using C# and VB.NET a new option, "Language" has been added to the Options tab. Please select your language so that the comments will be formatted correctly for you.

ASP.NET

I had to think long and hard about allowing ASP.NET properties to be edited. Most ASP.NET developers will probably agree that this can be problematic. We have enabled this feature, but please keep in mind that not all edits will be persisted to the page. ASP.NET has a complex page life-cycle, stores values in ViewState, cache, etc. I have been able to edit and persist many edits and see them when the page loads, but others are not persisted.

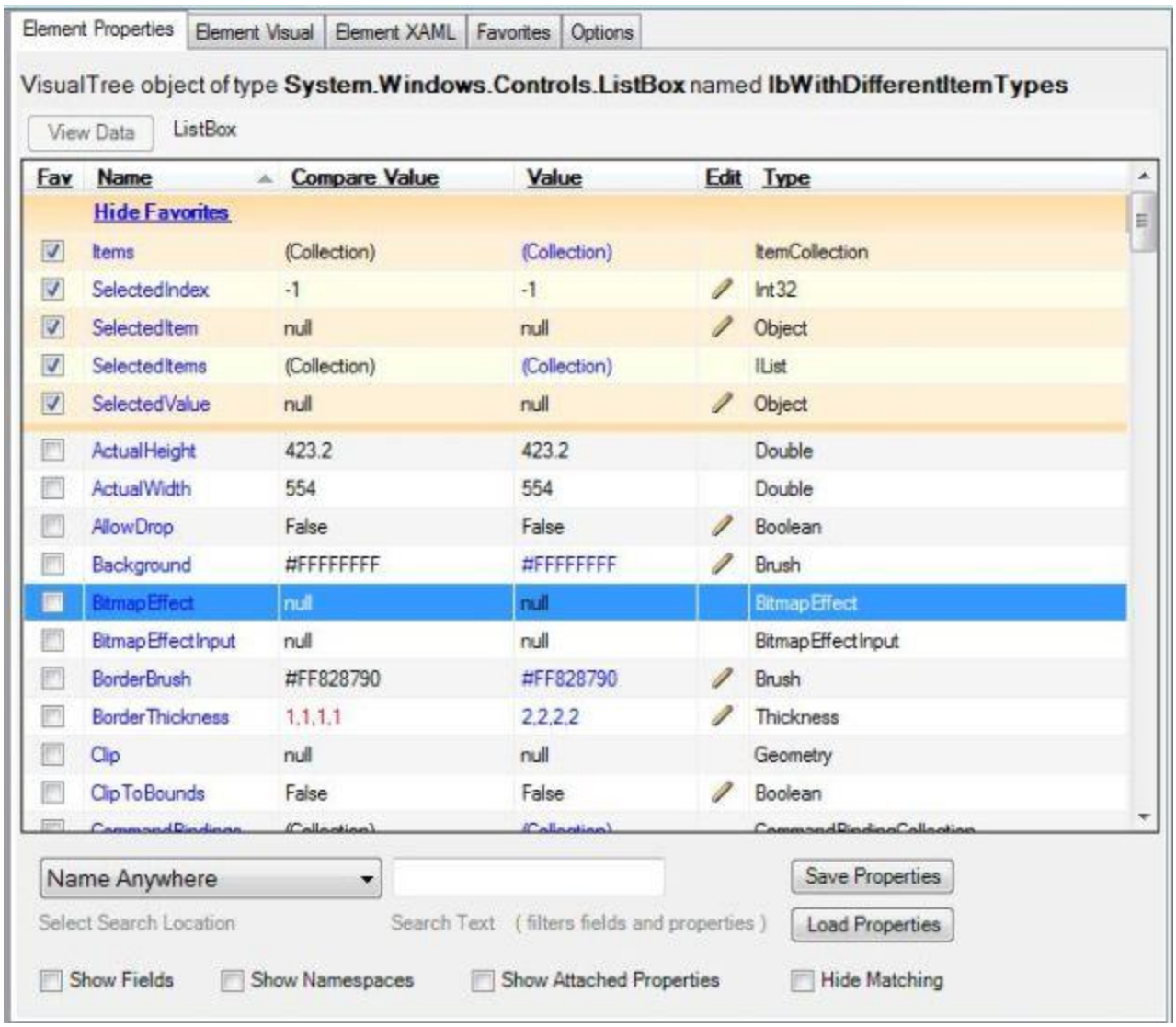
Searching

Mole allows you to search and filter the properties displayed. When searching, the Black Ops Field data and the normal property items are targeted by the search.

The "Select Search Location" `ComboBox` allows you to select where and how you want the search text to be used. The search is conducted as you type in your search text.

The "Show Attached Properties" `CheckBox` toggles the display of properties that are attached properties for those applications that support them.

Property Comparison

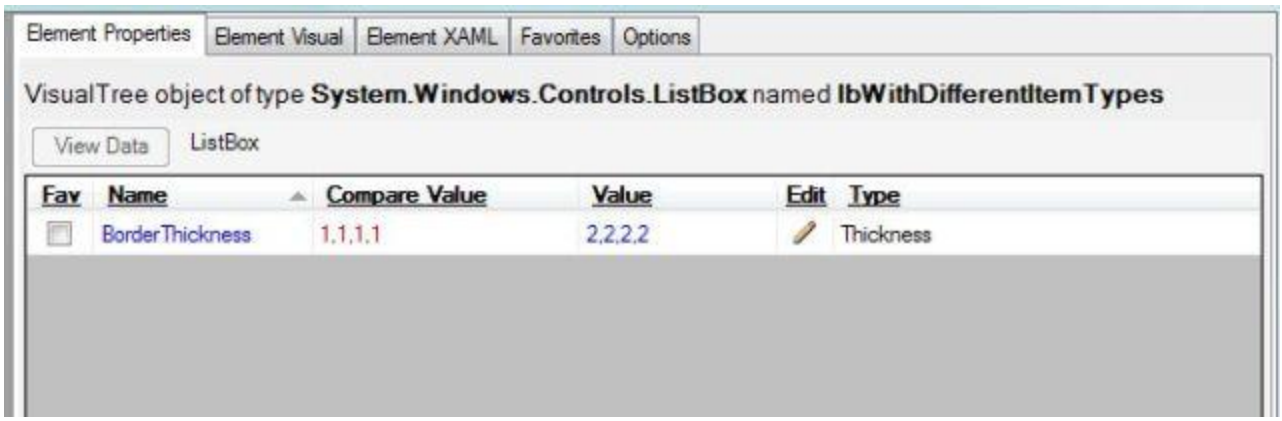


Purpose of Comparision Feature: This feature is handy when you need to know why something is working for one element and not for another and sometimes the best way to do that is to see what property values are different between the objects.

The `Save Properties Button` allows the developer to save the data from the properties grid to an XML file. Developers can view and/or work with this XML file as they would any other XML file.

The `Load Properties Button` allows a saved XML file to be imported and the values from the saved file compared to the current values in the properties grid. Once the file is loaded, the `Compare Value` column is visible. Where the `Compare Value` and `Value` are different, the `Compare Value` text has been set to `red`.

When an XML file has been loaded, the `Hide Matching Checkbox` is enabled. When clicked this will hide all matching values making it very easy for developers to find property values that are different.



The above image shows the properties grid with only non-matching value rows being displayed.

When the developer changes the selected element or drills up or down, the properties grid unloads the saved values and the properties grid returns to normal.

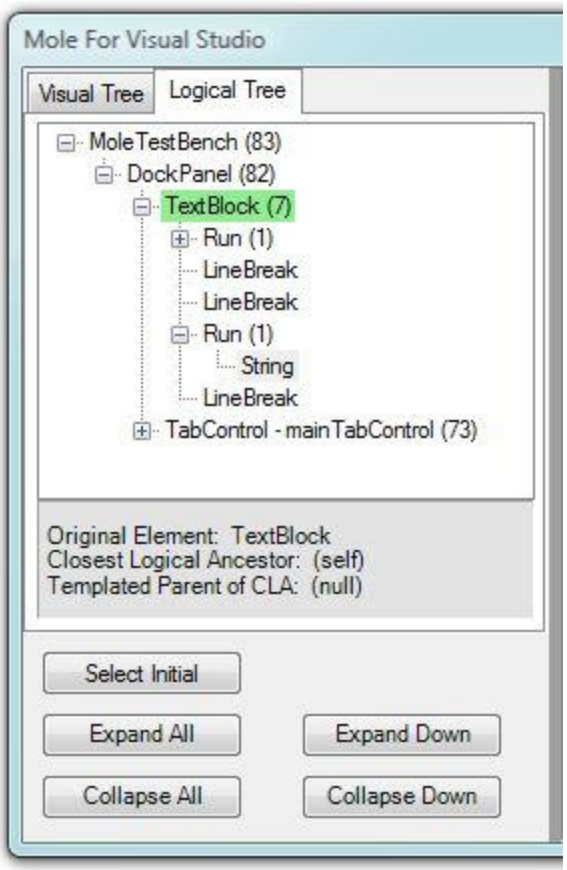
View the Logical Tree in a TreeView Control

This section's in-depth research and coding of the WPF logical tree was all done by Josh Smith. This is fantastic information! Thanks Josh for enhancing our understanding of the WPF logical tree!

In Mole v1 you could view the visual tree of the UI you were debugging, but not the logical tree(s). In Mole v2 we added support for viewing logical trees, as well as the properties, image snapshots, and XAML view of elements in the logical tree. The impetus behind this feature was to allow developers to inspect the logical tree so that they could understand how inherited dependency properties (such as `FontSize` and `DataContext`) get their values. Since property inheritance flows down the logical tree, this should help people better understand how and why their dependency properties inherit certain values.

Implementing this feature turned out to be rather challenging, because a significant amount of research and experimentation were necessary to fully grasp what needed to be done. After Josh completed his research into the differences between the visual tree and logical tree, he published the Understanding the Visual Tree and Logical Tree in WPF article on Code Project. If you intend on using the Logical Tree feature, you should be sure to read that article because it will explain the subtleties and quirks involved with the logical tree.

Here is a screenshot of the Logical Tree tab:

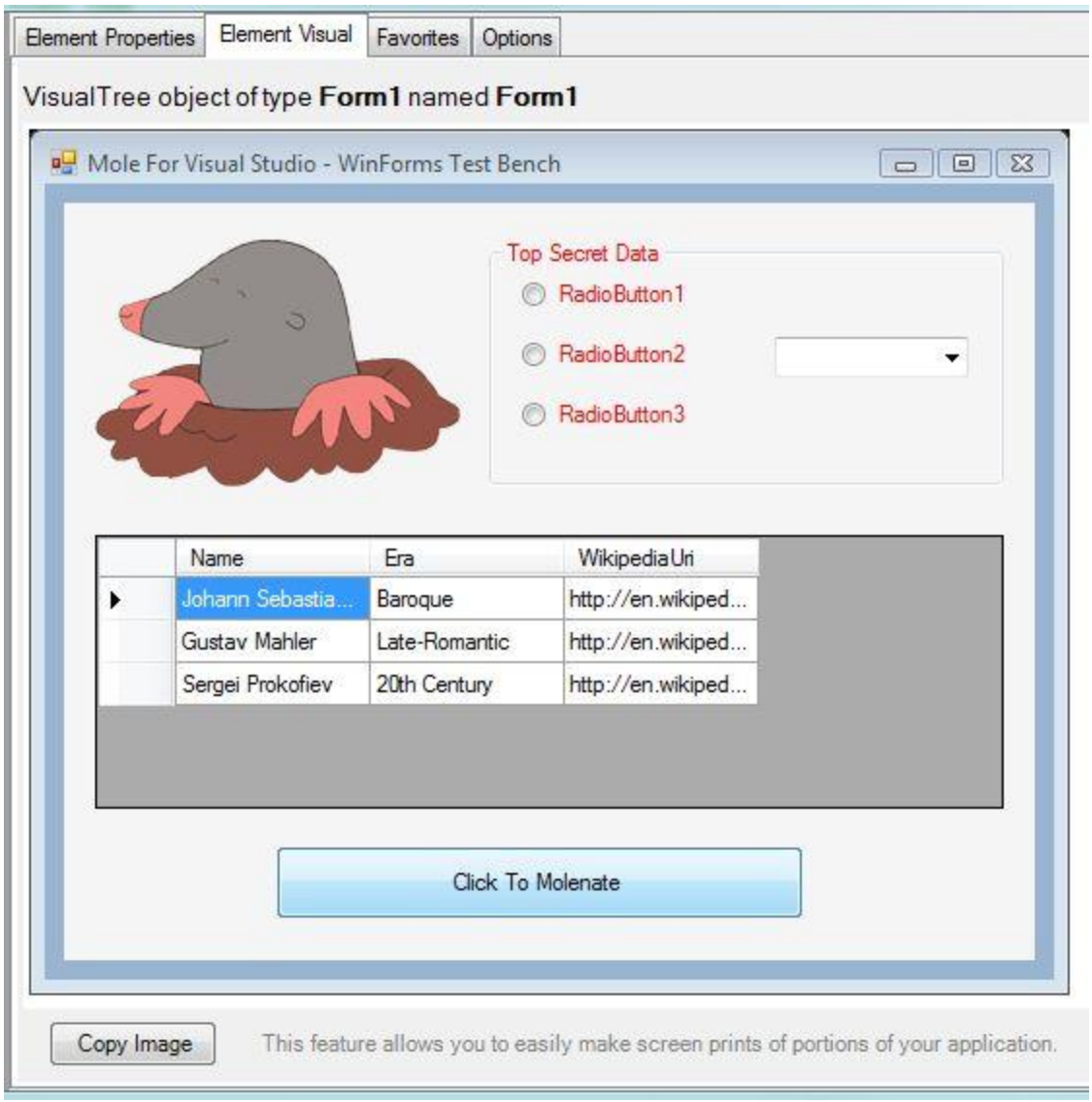


Notice how beneath the TreeView there is a text area which lists three pieces of information. The "Original Element" refers to whatever element you had selected in the Visual Tree tab before opening the Logical Tree tab. The Logical Tree tab shows the logical tree of that visual element, or, if that element is not in a logical tree, the logical tree of the element's closest ancestor which is in a logical tree. Clear as mud, right?

The "Closest Logical Ancestor" refers to the first ancestor element of the original element which is in a logical tree. If the original element (i.e. the one selected in the Visual Tree tab) is in a logical tree, then this value will read "(self)".

Lastly, the "Templated Parent of CLA" refers to the templated parent of the closest logical ancestor element. This is useful information when elements are generated by templates and are disconnected from the logical tree of the templated element/control in which they exist. If you have no idea what any of this means, we recommend that you read the article Josh wrote about this topic and hopefully it won't seem so bizarre afterwards.

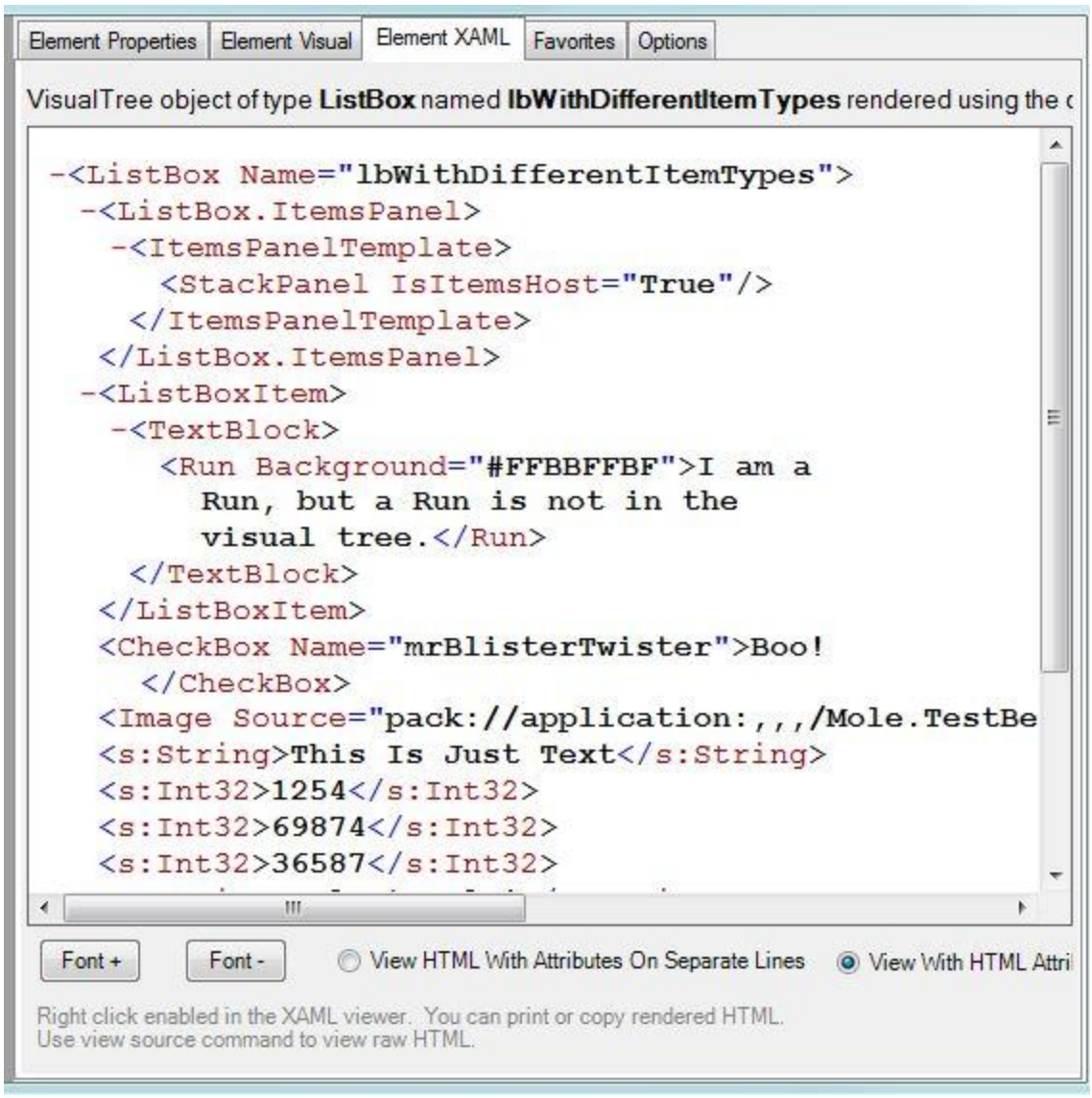
Element Visual



The "Element Visual" `TabPage` displays an image of the selected element. This tab is pretty cool because it gives you insight into how the various elements in the element/control tree participate in the rendering process. This feature is supported in WPF and WinForms. Thanks to Andrew for the code to make an image from a WinForms control.

The displayed image can easily be copied to the Windows Clipboard by clicking on the "Copy Image" Button. This Button makes it super easy to get snapshot images and copy them to other documents, emails, etc.

Element XAML



The "Element XAML" `TabPage` displays the run-time XAML which was transformed into HTML and loaded in the `WebBrowser` control. The label under the `WebBrowser` control follows the format of the other two `TabPages`.

The 'Font +' and 'Font -' Buttons allow you to change the font size of the displayed text. This setting is also persisted between Mole sessions.

The two `RadioButtons` allow you to select the format for the displayed XAML, expanded or compressed. This setting is also persisted between Mole sessions.

You may right-click the `WebBrowser` control to access the copy, print and view source functions.

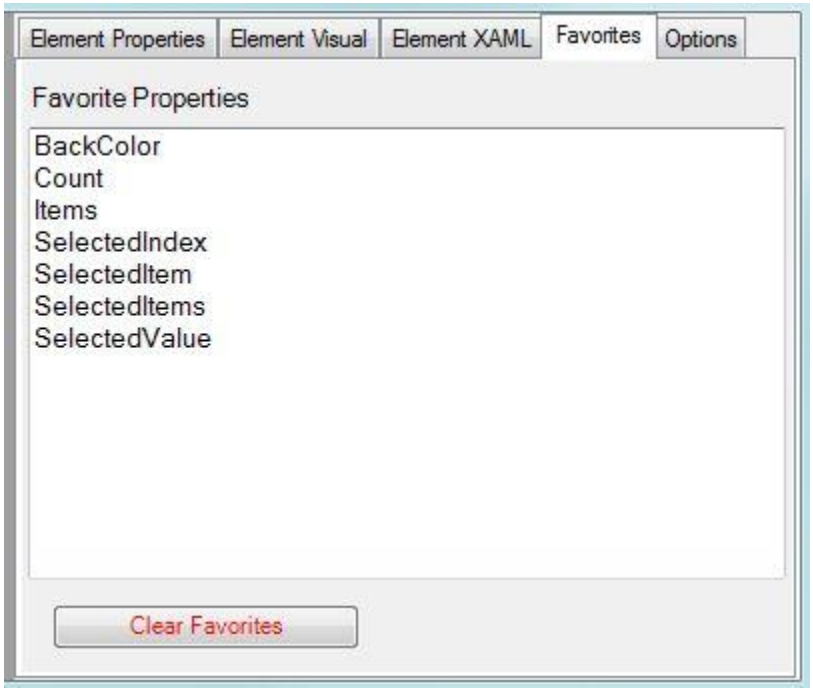
This `TabPage` took longer to write than any other section of code. I'm not joking. I really wish Microsoft would expose the default `XSLT` object which the `WebBrowser` control uses

internally or just add a public method like, `RenderXML` and the control would just perform the transformation like it does when it opens a file from the disk. This would make this control SO MUCH easier to use. To be clear, what developers need is a method that takes a `string` or `stream` of XML and the control renders it as transformed HTML. The control does this with XML files but hides this for text or stream input. At any rate, I downloaded a *defaultss.xslt* and edited it to get the above transformations. These files are in the solution source and are also embedded resources.

This `TabPage` is cool because if the element contains collections, those collections will be displayed here. I have loaded 20,000 items into a `ListBox` and viewed the XAML for them in this XAML viewer!!

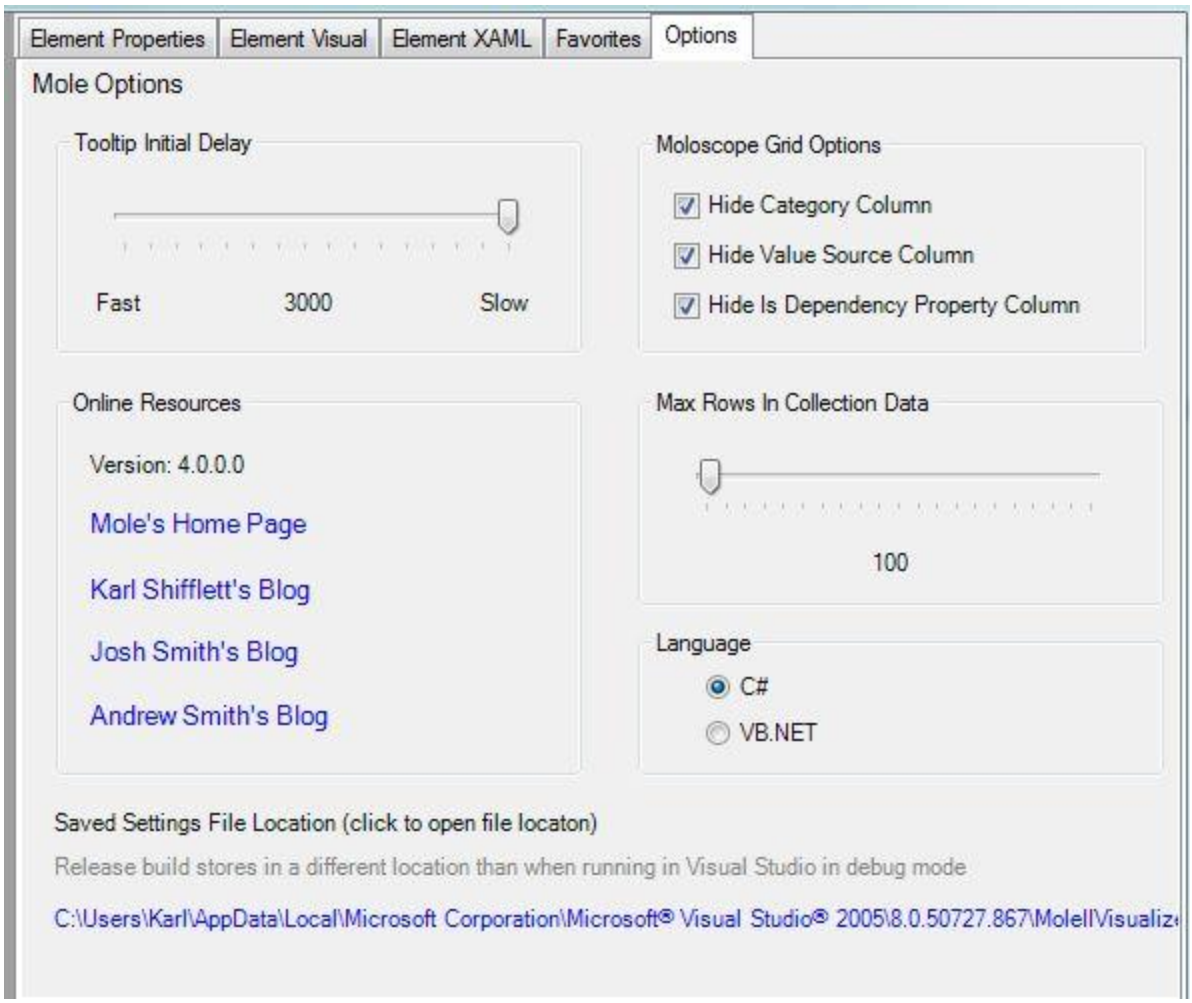
PLEASE READ THIS: If you want to conduct your own crazy list or combobox tests, remember, if your items are in a virtualizing `StackPanel`, only the visible items will be listed. In order to conduct one of the amusing tests, you must change the `ItemsPanel` in your control to a regular (non-virtualizing) `StackPanel`.

Favorites



This `TabPage` lists the properties and fields you have selected as favorites. There is also a "Clear Favorites" Button that can be used to remove all your favorites, if desired.

Options



Mole automatically remembers many settings for you, such as the window size, window location, screen number, which column is sorted in the grid, the width of the TreeViews, and more.

It also provides you with additional useful settings in the "Options" tab. Here is what that tab has to offer:

Tooltip Initial Delay

This feature was requested by Josh, 2 minutes after using Mole v1. The tooltips became a hindrance so I added the ability for the user to change the amount of time the mouse must hover over an element before the tooltip pops up.

Online Resource

Please notice the Version number. You can easily compare your version with the version number on Mole's Home Page. I thought about adding in automatic checking but ditched that idea. You can also visit Team Mole's blogs by clicking the links.

Moloscope Grid Options

You can hide some columns based on your usage preference. Please note, when inspecting a non-WPF object, the Value Source and Is DP columns will be hidden, since these are WPF-only columns.

Max Rows In Collection Data

After all the stress testing and crazy things we were doing, I added the ability to limit the number of rows in collections that gets displayed. You can adjust this to suit your needs.

Language

Please select the language you use. This setting effects the formatting of the Mole Editing Log.

Saved Settings File Location

Thought someone might want to back their settings up so I provided a link to the directory. Just click the hyperlink and the directory where your settings are stored will be displayed in Windows Explorer.

All Visual Studio Projects

Collections

Viewing collection data in Mole is much easier than in Visual Studio. Basically with one or two mouse clicks you can view data inside your collection objects and collection objects that are part of the .NET Framework. Example: `Children` or `Controls` properties.

Also Mole displays collections with multiple data types in them. Example `ArrayLists` can contain data with dissimilar data types. The WPF `ItemsCollection` can contain just about anything. Mole handles this scenario with ease.

Mole provides two views for the same data. Individual members of a collection are displayed in the properties grid and can also be viewed in data tables using the "View Data" button. The collection members in the properties grid can be drilled into exposing rich data about that member. The data table view provides the typical data row view of collection members.

There are two images of viewing collection data in this article.

Favorites

This was my first feature in Mole v1. I really wanted this.

Mole allows establishing Favorites to make getting to data you normally view much faster.

Let's use the `ComboBox` for an example. When debugging in Visual Studio and needing to inspect a `ComboBox`, I'm always wanting to view the `SelectedIndex`, `SelectedValue`, etc. properties. These properties are towards the bottom of list and I need to scroll down.

With Mole, you can simply add these properties to your Favorites. Then, every time you look at a `ComboBox`, these properties are at the top of the properties grid.

Searching

Mole allows you to search the properties grid to find what you're looking for. You can search the property Name column and property Value column. With one or two keystrokes you can zero in on exactly what you're looking for.

WinForm Project Example

Mole allows you to easily drill around your application, Visual Studio does not.

For example open a WinForms application, place a breakpoint and perform a "Quick Watch" on the form object. For C# target, `this` and for VB.NET target, `Me`. Look around, you'll see lots of information about a `Window` object in the Visual Studio debugger.

Now visualize the same object in Mole. Much more data that is easy to find and drill around is displayed. In fact the entire application is easily navigated. You can even view an image of each WinForm control.

This is one reason the Visualizer feature was put into Visual Studio. Visualizers allow the display of rich data in a customized format. The DataSet Visualizer is a good example of this. Mole just treats your entire application as rich data, displays it and allows you to drill around it.

The editing feature can help with making changes to forms that are dynamically generated. For WinForm applications, Visual Studio is a rock solid WYSIWYG editor and designer. For those applications where the UI is generated dynamically, Mole can help you edit your project at run-time, make the changes, view them and then returns a log of what you changed so that you can edit your source code.

WPF Project Example

The same features and examples from the WinForm project apply here as well.

If you are still using VS2005 and do not use Blend, Mole can really help out with your UI adjustments.

ASP.NET Project Example

The same features described in the above WinForm section also apply to ASP.NET projects.

Depending on when you open Mole up in relation to the `Page` cycle, Mole allows the developer to visualize the web page from a rendered `Page` point of view.

Open the provided Test Bench program for ASP.NET. When you view the *default.aspx*page using Mole, the Literal Controls added to the output stream of *default.aspx* are visible in Mole.

When using Mole with ASP.NET applications, developers can really "see" the difference in the "state" of their applications between "before a PostBack" and "after a PostBack". When debugging a page before a PostBack has occurred objects are being constructed, databases queried, `DropDownList` controls filled with data, etc. When debugging after a PostBack, these operations are probably not occurring because these objects can be persisted between PostBacks using `ViewState`. I think for new ASP.NET developers this tool is invaluable in teaching this important concept.

In all honesty, I may only be scratching the surface here. Maybe in the future, we'll see blog postings and articles on getting the most from Mole or tools like Mole.

WF Project Example

As of Mole v4.2 can be used on WorkFlow (WF) applications. We have also posted a video on using Mole with WF on YouTube. You can learn a good bit by watching the vidoes we have posted.

Two attributes were added to the `Mole.Visualizer.dll`, `System.Workflow.Runtime.WorkflowInstance` and `System.Workflow.ComponentModel.DependencyObject`. These attributes instruct Visual Studio to display a data tip when the developer hovers their mouse over objects of these types or objects that are derived from these types.

A new WF project Test Bench was added to the source and test benches download. There are two places in the source code that we have left instructions for where to place break points to view WF data. These are marked with the 'TODO comment.

Mole really shines with respect to the `WorkflowInstance` object. This type only has two public members. When you use Mole to visualize this type, be sure to click the "`Show Details`" `CheckBox` to view the private members. You will be AMAZED at all the data that is available about the `WorkflowInstance` class.

Check out the `Workflow1.vb` code file. View this activity in the GUI designer, then the code, then use Mole on it. It's great to see the branches in the GUI designer and then using Mole to see how this all fits together, the delegates that are created, etc.

Launch Mole Using WeakReference Hack

"The Rock Star Hack of 2008!" - We at Team Mole have been trying for a long time to figure out a generic way for developers to launch Mole. On 1/19/2008, Josh "Rock Star" Smith has finally breached this stone wall with his awesome hack!!

Once you get a hold on how **powerful** this hack is, your debugging days will never be the same again! I will also post a video on this feature and Josh Smith will be blogging about this too.

This feature enables you to start Mole on any object any time you want to. You do NOT need to modify the attributes on the Mole.Visualizer.dll file.

Mole has been modified so that Visual Studio can reconginze `System.WeakReference` as a supported `Type` that Mole can visualize.

All that is required for a developer to use this hack, is to place a breakpoint anywhere in your code. Then use the Visual Studio Watch Window, and add a new watch, wrapping the object you want to visualizer in the constructor of the `Weakreference` object like so:

```
new system.weakreference( object to visualize )
```

If you wanted to view an object class like `Customers` that had a variable `objMyCustomers` declared you would launch Mole like this:

```
new system.weakreference(objMyCustomers)
```

After you add the new watch, press <ENTER>. You will then see the magnifying glass appear. Just click it. Presto, you are viewing your object in Mole.

If you are using Visual Studio 2005, you must use the full type name, `System.WeakReference` when adding the watch. When using Visual Studio 2008, you can omit "`System.`" and simply enter `new weakreference( object ).`

When used this way, Mole tries to figure out what project type your are using and will attempt to build the familar application tree. If Mole can't figure this out, it will load your object in the application tree, and then expose the object in the properties grid. You can then drill around like normal.

Please read: You can not use `Value Types` with this method of starting Mole. When `Value Types` are passed in the constructor of the `WeakReference` object, they are lost when the `WeakReference` object is serialized. If you pass a `Value Type` using this method, you will get a message box stating that you can't use `Value Types` in the `WeakReference` constructor.

Team Mole hopes you all like this new and exciting debugging feature!

Limitations

So far, the only limitation I've run into is the Element XAML. Under certain circumstances the `System.Windows.Markup.XamlWriter.Save` function can throw a `StackOverFlow` Exception or a `GenericTypeSerialization` Exception. These two exceptions are thrown on the debuggee side of the remoting conversation that handles data transfer requests. Once its stack is corrupted, that process must be terminated, thus ending our visualizer session. Mole has no control over this. The `StackOverFlow` will be reported in a `MessageBox` and the visualizer will close. Just reopen it and don't select that element and view its XAML. I have seen this happen with complex 3D models in WPF projects.

You would get these same exceptions in your own code if you called `System.Windows.Markup.XamlWriter.Save` and passed the same object. Maybe this will be fixed in a .NET 3.0, 3.5 service pack.

Developers Please Read

We have left 'TODO and 'HACK' Visual Studio comments in the source code. Please take a look at these.

Visual Studio 2008 Warning Message

Opening up a release build of Mole can cause Visual Studio to display a message box stating: "This module was built either without optimizations enabled or without debug information." To make that stop, go to Tools | Options | Debugging | General and then uncheck the “Warn if no user code on launch” option.

Mole & Visualizers 101

The first article I read was Creating Debugger Visualizers with Visual Studio 2005 by Julia Lerman. She does an outstanding job of getting you started. Another good starting point is the MSDN Visualizer Architecture article. Also there many articles here on CodeProject and the Internet covering visualizers.

Basically there are four partners, running in two processes, that work together in the visualizer world.

Debugger Side

- Visual Studio Debugger - provides UI to select a visualizer to open.
- Visualizer UI - runs within the VS debugger process.

Debuggee Side

- Your Program - the program you are debugging. This process has the visual tree we want to visualize.
- Visualizer Data Object - runs within your program's process.

Visual Studio plays the middle man handling communications between the two processes. The bottom line is, your visualizer UI has no data, until it makes a request to the `VisualizerObjectSource`. 'All data which moves between the two processes must be serialized.' This is important to remember when building your data structures.

Both sides of the conversation are active throughout the visualizer's life. This allows your UI to make repeated requests to the data object whenever it needs more data to display. The communications plumbing Microsoft provided is incredibly fast, even with all the serialization and deserialization going on.

Visualizer Project

If you are used to writing executables, this is going to blow your mind. Your visualizer complies into one class library DLL, part of which runs in one process and the other part is running in the other process. The classes from the Debugger process communicate through Visual Studio to classes in the Debuggee process.

How Does Mole's Project Fit Into This?

Visual Studio is only concerned about two classes in your visualizer. First, your Debugger Side class which derives from `DialogDebuggerVisualizer`. In Mole, this is the `Mole.Burrow` class. Second, your Debuggee Side class which derives from `VisualizerObjectSource`. In Mole, this is the `Mole.MoleVisualizerObjectSource` class.

Visual Studio identifies these classes by looking at the following attributes placed on the assembly in which your visualizer lives. Since all visualizers must be in certain directories, it is very easy for Visual Studio to locate all visualizers on your system and determine which types have visualizers. You cannot have Visual Studio honor an attribute with a `TargetType` of `System.Object`. (Microsoft, did I miss something?) So, Mole has several attributes applied to it.

Collapse

```
'TODO developers feel free to add more Types that Mole can visualize
'     You only have to add additional attributes here.
'     Mole can handle just about ANYTHING you throw at it.

<Assembly: System.Diagnostics.DebuggerVisualizer(GetType(Mole.Burrow),_
   GetType(Mole.MoleVisualizerObjectSource), _
   Target:=GetType(System.Windows.DependencyObject), _
   Description:="Mole (Exploring WPF)")>

<Assembly: System.Diagnostics.DebuggerVisualizer(GetType(Mole.Burrow), _
  GetType(Mole.MoleVisualizerObjectSource), _
  Target:=GetType(System.Web.UI.Control), _
  Description:="Mole (Exploring ASP.NET)")>

<Assembly: System.Diagnostics.DebuggerVisualizer(GetType(Mole.Burrow), _
  GetType(Mole.MoleVisualizerObjectSource), _
  Target:=GetType(System.Windows.Forms.Control), _
  Description:="Mole (Exploring WinForms)")>

<Assembly: System.Diagnostics.DebuggerVisualizer(GetType(Mole.Burrow), _
  GetType(Mole.MoleVisualizerObjectSource), _
  Target:=GetType(System.Data.DataSet), _
  Description:="Mole (Exploring DataSet)")>

<Assembly: System.Diagnostics.DebuggerVisualizer(GetType(Mole.Burrow), _
  GetType(Mole.MoleVisualizerObjectSource), _
  Target:=GetType(System.Data.DataTable), _
  Description:="Mole (Exploring DataTable)")>

<Assembly: System.Diagnostics.DebuggerVisualizer(GetType(Mole.Burrow), _
  GetType(Mole.MoleVisualizerObjectSource), _
  Target:=GetType(System.Windows.RoutedEventArgs), _
  Description:="Mole (Exploring RoutedEventArgs)")>

<Assembly: System.Diagnostics.DebuggerVisualizer(GetType(Mole.Burrow), _
  GetType(Mole.MoleVisualizerObjectSource), _
  Target:=GetType(System.EventArgs), _
  Description:="Mole (Exploring EventArgs)")>

<Assembly: System.Diagnostics.DebuggerVisualizer(GetType(Mole.Burrow), _
  GetType(Mole.MoleVisualizerObjectSource), _
  Target:=GetType(System.ServiceModel.ServiceHost), _
  Description:="Mole (Exploring WCF)")>
```

```
<Assembly: System.Diagnostics.DebuggerVisualizer(GetType(Mole.Burrow), _
  GetType(Mole.MoleVisualizerObjectSource), _
  Target:=GetType(System.Data.SqlClient.SqlException), _
  Description:="Mole (Exploring Exception)")>
```

First attribute parameter is the type of our visualizer (`Mole.Burrow`).

Second is the type of our data source (`Mole.MoleVusualizerObjectSource`).

Third tells Visual Studio what type our visualizer wants to visualize. (In Mole speak that is, "what type Mole wants to Burrow into and molenate".) In our first attribute, Mole can visualize `DependencyObjects`, or any class which descends from `DependencyObject`.

The fourth attribute parameter is the description Visual Studio displays in the listing of available visualizers when you are debugging. Glance back to the second image in this article and you'll see the above description.

The remaining attributes simply register their object types with Visual Studio so that this Mole visualizer will be listed in the data tip during debugging.

Using Mole To View Exceptions

I've added a section of code to the WinForm Test Bench program that throws an `SqlException`. This code is commented out. You can uncomment the `Try Catch` block, place your breakpoint where indicated in the code and run the WinForm Test Bench program. When the `SqlException` is thrown, use Mole to drill around it. Remember that the `SqlException` has an `Errors` Collection that can contain one or more SQL errors.

I did try to get the visualizer to recognize all exceptions by setting the `TargetType` to `System.Exception`, but it would not allow other exceptions that derived from `System.Exception` to be visualized. So, as you can see in the above block of code, I have added an attribute with the `TargetType` of `System.Data.SqlClient.SqlException`.

Viewing the exception was for instructional purposes and to spread Mole's wings to see what the application is capable of.

Where Are We

At this point, all Visual Studio developers have enough information to begin using Mole for their advanced debugging needs. Additionally you can target more .NET Framework types for Molanization.

If there is some interest, Team Mole could write a paper on the internals of Mole and possibly give a presentation also.

I know this article is long, but for all the new users of Mole, necessary.

Mole Fun Stuff

The following provides information for becoming a Mole Geek.

Mole Speak

- Molenator - Creator of Mole
- Burrow - To dig through the visual or logical trees
- Molenate - The use Mole
- Moloscope - Hi tech object used to manage the drilling back into related visual or logical tree objects and sub-objects, located in the managed heap of another process
- Moletones - Moles color scheme. Mole v1 sported earth tones, while Mole v2 sports default control colors.
- Moleskin - Similar to Moletones
- Max-O-Drilling - A technology employed by the Moloscope that Mole uses to manage accessing objects in the managed heap that is located in a separate process
- Molecular - Refers to the size of Mole's data structures - way small
- MoleCrumb - A breadcrumb that is left when a drilling operation is executed inside the Molescope
- MoleCrumbs - A collection of individual MoleCrumb's. This is also the name of the breadcrumb style control I wrote for the Moloscope
- Drilling Operation - The act of drilling into an object in the Moloscope
- Molathon - What we went through to deliver this product
- Molahack - A unique implementation of code to work around a base class's lack of public features
- Molet - A single row in the Moloscope
- Molets - A collection of individual Molet
- Molahole - A side effect of Insect Programming (see below). A black hole that just keeps sucking you into creating more FEATURES!
- Black Ops - A cool feature of Mole that allows the viewing and drilling into of private and protected members.
- Rock Star Hack of 2008 - Josh Smith's awesome idea to use the WeakReference object as a generic wrapper to allow Mole to be started with ANY object.

Sample Questions You Can Ask Fellow Developers

- Are you Mole'n? Response: oh yea, I'm Mole'n.
- Do you Molenate? Response: doesn't everyone.
- Do you Burrow? Response: every time I need to debug my WPF code.

Insect Programming Methodology

Insect Programming was heavily utilized during Mole's coding cycle. This is where most of Mole's features came from.

The basic tenet of Insect Programming, as opposed to eXtreme Programming, is whatever features fly into our heads, we program.

The Insect methodology increases developer creativity because it completely bypasses the boring specification process and all those meetings and long drawn out discussions, UML diagrams, etc.

What is really funny is that at work, I'm the exact opposite; no spec, no keyboard activity.

Mole Quotes From Users Of Mole

- *I just wanted to tell you GREAT JOB with Mole… I simple love it… Thanks for giving this to the community…*
- *Tip of the Week - Visual WPF Debugging: This week's tip goes out to all the WPF developers out there: Mole II is a graphical debugger for WPF and once used you won't know how you could ever live without it. A must have tool for the tool belt of every serious WPFer.*
- *Mole is a member of the Visual Studio Visualizer family in the order Debugger. Mole lives in Visual Studio and burrows holes. Mole has a cylindrical body covered in fur with small or covered eyes; the ears are generally not visible. It feeds on WPF controls living underground in the debugger.*
- *Absolutely fantastic! I wish I could give you guys more than a 5. This is one of the best Code Project projects of all time.*
- *This is tremendous. No buts about it, this is a tremendous piece of work, and a huge contribution. This is going to save lots of time. Thanks guys, fabulous job.*
- *Great work guys. It is truly fast and lets you gain quick insight into WPF much beyond what you can do watching variables.*
- *It's keep getting better and better. 5 again*
- *Well done all of you. This tool is truly excellent.*
- *Kudos! Simply amazing to see how you keep it going and going and push the idea further and further!*
- *As the title says, only 5! You all R O C K !! Thanks for showing the way.*
- *Awesome stuff, great application and article - you've got my 5! The new visual looks great!*

Resources

Check here Team Mole & Mole Project Home and Mole's Home Page for additional information.

If you subscribe to Karl's Blog RSS feed you will be notified of all enhancements to Mole.

You can leave comments at the bottom of this article, on our blogs or send Mole an email at: moleproject@yahoo.com.

References

Woodstock – It all started here
http://www.codeproject.com/KB/WPF/WoodstockForWPF.aspx

Mole v1 article
http://www.codeproject.com/KB/WPF/MoleForWPF.aspx

Mole v2 article
http://www.codeproject.com/KB/WPF/moleIIforWPF.aspx

Mole v2.2 Black Ops article
http://www.codeproject.com/KB/WPF/MoleBlackOpsVersion.aspx

Mole v3 Visual Studio Visualizer article
MoleForVisualStudio.aspx

Understanding the Visual and Logical Tree In WPF article
http://www.codeproject.com/KB/WPF/WpfElementTrees.aspx

Josh Smith's Blog
http://joshsmithonwpf.wordpress.com/

Karl Shifflett's Blog
http://karlshifflett.wordpress.com/

Andrew Smith's Blog
http://agsmith.wordpress.com/

MSDN Visualizers (has links to many MSDN visualizer topics)
http://msdn2.microsoft.com/en-us/library/zayyhzts.aspx

Close

Team Mole hopes you find Mole For Visual Studio a productive developer tool and that this article helps learn a little more about Visual Studio Visualizers.

From Team Mole, have a wonderful day and enjoy Mole!

History

- 31 Dec 2007: Initial release
- 13 Jan 2008: Mole v4.1: Made changes to VisualSnapshot code to catch and handle possible exceptions caused by WinForm object property editing. Added Full Type Name ToolTips to the TreeViews, MoleCrumbs and property grid Type column. Added Show Namespaces option to have the TreeViews, MoleCrumbs and property grid Type column show the full type names. Added ability to export the property grid to an XML file. That file can then be imported to allow element comparisons. Hide the Show Attached Properties CheckBox when not viewing WPF applications.
- 19 Jan 2008: Mole v4.2: Now supports using Mole with WorkFlow (WF) applications. A new test bench project for WF was added to the Source and Test Benches solution. For full details, please see WF Project Example section below. **"The Rock Star Hack of 2008!" -** We at Team Mole have been trying for a long time to figure out a generic way for developers to launch Mole. Josh "Rock Star" Smith has finally breached this stone wall with his awesome hack. Mole has been modified so that Visual Studio can reconginze `System.WeakReference` as a supported `Type` that Mole can visualize. For full details please see the Launch Mole using a WeakReference Hack section below.