XamlQuery v1.2

The Write Less, Do More, Silverlight Library

Contents

I. About *XamlQuery*
II. Downloading *XamlQuery*
III. Using *XamlQuery*
IV. Selectors
V. *XamlQuery* API
VI. Animation in *XamlQuery*
VII. Event Handling in *XamlQuery*
VIII. Examples and Demonstrations
IX. Frequently Asked Questions (FAQ)

I. About *XamlQuery*

What is *XamlQuery?*

XamlQuery is a lightweight yet powerful library that enables rapid web development in *Silverlight*. It simplifies several tasks like page/document traversing; finding controls by name, type, style, property value or position in control tree; event handling; animating and much more. In short, what *XamlQuery* aims to do to *Silverlight* is similar to what *jQuery* does to JavaScript.

XamlQuery selectors are feature rich and similar to *CSS* (cascading style sheet) selectors and *jQuery* selectors. The controls can be found very easily using simple to complex selector queries or managed helper methods. Also, *XamlQuery* provides a powerful API that simplifies complex operations on found controls. Most of the API functions can be invoked in a single line of code.

Why *XamlQuery?*

- Unlike other markup languages (like *HTML)* the controls tree structure in the rendered *Silverlight* output is different from markup declarations (XAML). The controls cannot be found by directly manipulating the XAML markup tree, because *Silverlight* creates many additional intermediate controls in the rendered output.

- *Silverlight* provides *LogicalTreeHelper* and *VisualTreeHelper* helper classes in order to find the controls in a rendered *Silverlight* page. But, these classes are mostly useful to find the immediate parents and children only.

- The *XamlQuery* works for all controls in *Silverlight*, not only for container controls like *Grid* and *Panel*; because even primitive controls like *ContentControl* can contain any other control like *Grid*, *TreeView* inside their templates (data-templates and control-templates).

- *XamlQuery* hides the complexity of finding controls using styles that are defined elsewhere, like merged resource dictionaries or application XAML markup (*App.xaml*).

- The controls of specific type or a range of types (using base class) can be found easily by the use of type-selectors. For example, using *Shape* as a type-selector will return all shape based controls like *Rectangle*, *Line*, *Ellipse*, etc.

II. Downloading *XamlQuery*

The source code, runtime binaries, documentation and API Reference of *XamlQuery* are available for download in http://xamlquery.codeplex.com/releases.

XamlQuery is released under Microsoft Public License.
http://www.microsoft.com/opensource/licenses.mspx#Ms-PL.

III. Using XamlQuery

XamlQuery provides two approaches for finding controls.

- Using Selector Queries, and
- Using Managed Helper Methods

Using Selector Queries

A selector is a string that contains a set of rules or conditions to find controls. The search is performed in child / descendant controls of a given reference control. Once the syntax and usage are understood, the selectors are very easy and intuitive way to find controls. Selectors are explained in detail in a later chapter.

The reference control limits the search scope. It is very important to search in a lesser scope because of *Silverlight's* behavior of creating additional controls in the rendered output. For example, if a *Grid* is placed inside a *ScrollViewer*, *Silverlight* will create a *Rectangle* control in between *ScrollViewer* and *Grid*.

For instance, if the user needs to find all the rectangles inside a *Canvas* control, consider the following two lines of code.

```
XamlQuery.Search(LayoutRoot, "Rectangle");
XamlQuery.Search(MainCanvas, "Rectangle");
```

The first line will return many unwanted Rectangles, whereas the second line will return controls inside the *MainCanvas* only. Note the *LayoutRoot* is the name of root control of the *Page* or *UserControl*.

The following method is used to find controls using selectors.

- **Search()** method accepts a reference control and a selector query-string and returns the matching controls. The reference control is the one where the search starts.

Using Managed Helper Methods

The following are helper methods that are used to find children and parent controls.

- **All()** finds all children of a given control.
- **ByType()** finds all children of specified type, for example *TextBox*, *Rectangle*, etc.

- **ByProperty()** finds children whose property value matches a specified criteria. The *FilterType* optional parameter controls the matching of property value. The possible values for *FilterType* are *Equal*, *NotEqual*, *StartsWith*, *EndsWith* and *Contains*. The string representation of the property value is used for *StartsWith*, *EndsWith* and *Contains* filter-types.
- **AllParents()** finds all parents (upto root) of a control.
- **ParentsUpto()** finds all parents, until the parent with specified type or name is found.
- **ParentByType()** finds the first parent of specified type.
- **ParentsByType()** finds all parents of specified type.
- **ParentByName()** finds a parent control by name.
- **Root()** method finds the root control in the rendered *Silverlight* page. If a *UserControl* or *Page* is embedded within another *UserControl* or *Page*, the later will be returned as root.

The *ControlSet* class is used to hold the set of controls returned by *XamlQuery* methods. The *ControlSet* provides several methods/functions for carrying out useful tasks related to the controls. The *ControlSet* is explained in detail in *XamlQuery API* section.

IV. Selectors

XamlQuery Selectors are similar to selectors of *CSS* (cascading style sheets) and *jQuery*. The selector symbols and syntax are borrowed from *CSS* and *jQuery*. *XamlQuery Selectors* operate on rendered Silverlight output, where *CSS* and *jQuery* selectors operate on rendered HTML output.

What is a Selector?

A *Selector* is a string that contains a set of matching rules or conditions that determine which controls to extract from the rendered *Silverlight* output. It is like a pattern or query that contains simple control names to rich / complex patterns.

If all the rules / conditions in the pattern are true for a given control, the selector matches that control. The identifiers (like control names, control types, style names, etc.) in the selector pattern are case sensitive. The controls found by a selector are called as subjects of that selector.

Selector Syntax

The input query pattern can contain one or more selectors, separated by *comma*.

selector-1, selector-2, … … … selector-n

A selector is a chain of one or more simple selectors, separated by combinator symbols. Combinator symbols are white-space (descendant selector), ">" (child selector) and "+" (adjacent selector). Additional white-spaces around a combinator symbol are ignored. Note if there are *n* simple-selectors in a selector, there will be *n-1* combinators.

simple-selector-1 combinator-1 simple-selector-2 combinator-2 … … …
combinator(n-1) simple-selector-n

A simple-selector is either a type-selector or universal-selector followed immediately by zero or more filter selectors. Immediately implies that there should not be any white-space between a simple-selector and its filter-selectors.

[*]*filter-selector-1 filter-selector-2 … … … filter-selector-n*

or

<type>*filter-selector-1 filter-selector-2 … … … filter-selector-n*

A filter selector is a style-selector or name-selector or property-selector.

Selector Patterns

The following table summarizes the syntax of selectors.

Pattern	Name	Description	Used in/as
*	Universal Selector	Matches any control	Simple Selector
E	Type Selector	Matches any E control (control of type E or subclass of E)	Simple Selector
E F	Descendant Selector	Matches any F control that is a descendant of control E	Selector
E > F	Child Selector	Matches any F control that is a child of control E	Selector
E + F	Adjacent Selector	Matches any E or F control that are children of a same parent control	Selector
E.style	Style Selector	Matches any E control whose style is equal to or based on a given style	Filter Selector
E#name	Name Selector	Matches any E control with specified name	Filter Selector
E[property]	Property Selector (Not empty/null)	Matches any E control whose *property* value is not equal to its default value	Filter Selector
E[property=value]	Property Selector (Equals)	Matches any E control whose *property* value is equal to given value	Filter Selector
E[property!value]	Property Selector (Not Equals)	Matches any E control whose *property* value is not equal to given value	Filter Selector
E[property^value]	Property Selector (Starts With)	Matches any E control whose *property* value starts with given value	Filter Selector
E[property$value]	Property Selector (Ends With)	Matches any E control whose *property* value ends with given value	Filter Selector
E[property~value]	Property Selector (Contains)	Matches any E control whose *property* value contains given value	Filter Selector

Universal Selector

The universal selector, written as '*', matches any control. It matches any single control in the rendered *Silverlight* output, including additional controls created dynamically during rendering. The symbol '*' can be omitted, if it is not the only component of a simple-selector.

For example,

- **#MainCanvas* is equivalent to *#MainCanvas*
- **.GreenRect* is equivalent to *.GreenRect*
- **[Tag=Highlighted]* is equivalent to *[Tag=Highlighted]*

Type Selector

The type selector finds controls whose type is equal to given type or subclass of given type. It matches every instance of the control type within the search scope.

Type selectors can be used to find controls of specific type or a range of types (represented by a base-class). For example, *Shape* will match all controls that are extended from *Shape* like *Rectangle*, *Line*, *Ellipse*, etc.

The following example finds all textbox controls inside a grid.

#RegisterGrid TextBox

The following query finds all rectangles inside a canvas.

#MainCanvas Rectangle

The following query finds all Shapes (*Rectangle*, *Line*, *Ellipse*, etc.) inside a canvas.

#MainCanvas Shape

The following query finds all Selectors (*ComboBox*, *ListBox*, etc.) inside a grid.

#RegisterGrid Selector

Descendant Selector

A descendant selector is made up of two or more selectors separated by white-space. A descendant selector of the form "E F" matches all F controls which are descendants of control E. A descendant means a control that appears as a child (any depth, not immediate child) in the visual tree of rendered *Silverlight* output. That is, E can be reached by traversing upwards from F, no matter how many iterations in the traversal.

The following example finds all ellipses inside a canvas.

#MapCanvas Ellipse

The following example finds all textboxes that are grand child of a scroll viewer. The '*' indicates that a *ScrollViewer* must be the ancestor of some control and that control must be an ancestor of a *TextBox*.

*ScrollViewer * TextBox*

The following example finds all controls whose tag value is equal to 'Highlighted' (irrespective of the control type).

#MapCanvas [Tag=Highlighted]

Child Selector

A child selector is made up of two or more selectors separated by '>' symbol. A child selector of the form "E > F" matches all F controls that are children of control E. The control E should be a container control. A container control is a control that is extended from *Panel* class. Note that a child is a descendant but not all descendants are children. In fact, a child means an immediate descendant.

The following example gets all shapes with style named 'Stroke'.

#MainCanvas > Shape.Stroke

This query is similar to the following, assuming that *MainCanvas* is a descendant of *LayoutRoot*.

LayoutRoot #MainCanvas > Shape.Stroke

The following example gets all grids that are immediate children of all stack-panels in the search scope.

StackPanel > Grid

The following example gets all children of a wrap-panel named 'ScoresContainer'.

*WrapPanel#ScoresContainer > **

Adjacent Selector

Adjacent selectors have the syntax E1 + E2, where both E1 and E2 are subjects of the selector. The selector matches if both E1 and E2 are children of a same parent. The parent control should be a container control. A container control is a control that is extended from *Panel* class.

The following query matches all green rectangles and red lines inside a canvas.

#MainCanvas Rectangle.GreenRect + Line.RedLine

Style Selector

A style selector contains a dot (.) immediately followed by the style name. The selector matches all controls whose style is equal to or based on a given style. That is, the style name can be the name of exact required style or name of any base styles (declared with *Style.BasedOn* property).

The styles can reside in the same XAML file or any merged resource dictionary or Application XAML (*App.xaml*). *XamlQuery* will search for styles and base-styles in all these locations.

The following query finds all controls with style 'GrayText', irrespective of control type.

**.GrayText*

The following query finds all ellipses of style named 'HighlightedNode'.

#LayoutRoot Ellipse.HighlightedNode

The following query finds all ellipses of style named 'MapNode'.

#LayoutRoot Ellipse.MapNode

If 'HighlightedNode' style is based on 'MapNode' style, then this query will match highlighted nodes also.

Name Selector

A name selector contains a hash (#) immediately followed by the control name. The selector finds the control with the given name.

Only one control can exist with a given name in current namescope. Certain features of *Silverlight* such as templates or calls to APIs such as *XamlReader.Load* method can define separate XAML namescopes, which leads to two controls under the same name. However, there can be only one control in a given namescope. The name-selector finds and returns one control whose name is equal to given name in the current namescope. The current namescope means namescope of reference control passed to *Search* method of *XamlQuery*.

The following query finds the control named 'RegisterGrid'.

#RegisterGrid

The following example finds the control named 'EmailTextBox' that is a descendant of 'RegisterGrid'.

#RegisterGrid #EmailTextBox

Property Selectors

The property selectors are used to choose a set of controls based on the value of a given property. The property selectors can be applied to both normal properties (regular properties of a class) and dependency properties.

The dependency property names should contain the name of the defining class delimited by an underscore. Some examples are *Canvas_Left*, *Grid_Row*, *FrameworkElement_Visitbility*, etc. The property names need not to end with 'Property'. That is, *FrameworkElement.VisibilityProperty* should be given as *FrameworkElement_Visibility*.

[property]

> Matches if the property is set i.e. the value is not equal to default property value. For reference properties, the default value will be *null* and for value properties (like *int*) the default value will be a constant.

For example, the following query will get all text-boxes whose *Tag* property is set.

TextBox[Tag]

This query is equivalent to the following

TextBox[FrameworkElement_Tag]

[property=value]

Matches if the property value is exactly equal to the given *value*. *Equals* method of object is used to check the equality.

The following query will get all text-blocks in second column of a grid.

#SalesReportGrid TextBlock[Grid_Column=1]

The following query will get the textbox whose tag value is 'Current'.

#CustomerGrid TextBox[FrameworkElement_Tag=Current]

The following example gets all lines whose stroke-thickness is 2.

#MapCanvas Line[Shape_StrokeThickness=2]

[property!value]

Matches if the property value is not equal to the given value. *Equals* method of object is used to check the equality.

The following query will get all shapes that are not opaque.

Shape[UIElement_Opacity!1]

The following query will get all shapes with some height.

Shape[Height!0]

[property^value]

Matches if the property value starts with the given value.

The following example will find all controls whose name starts with 'Rect'

*[Name^Rect]

[property$value]

Matches if the property value ends with the given value.

The following query gets all controls whose name ends with 'TextBox'

#RegisterGrid [Name$TextBox]

[property~value]

Matches if the property value contains the given value.

The following query gets all controls whose tag property contains the text ‘Red’.

*[Tag~Red]

The starts-with (^), ends-with ($) and contains (~) symbols are ideal for string properties only (like *Name*, *Tag*, etc.), because string-representation of the property value will be used for matching. The string-representation is get by *ToString()* method of object class.

V. *XamlQuery* API

The *ControlSet*

All the methods of *XamlQuery* returns *ControlSet* object. The list of methods provided by *XamlQuery* are explained in *Using XamlQuery* section above.

The *ControlSet* class is used to hold the set of controls returned by *XamlQuery* methods. The *ControlSet* provides several methods/functions for carrying out useful tasks related to the controls. Each method operates on all the controls in the *ControlSet*.

Filter Methods

These methods reduce the set of matched controls to those that pass a criteria (defined by the method).

- **FilterByType()** method finds and returns controls of specified type. For example, you can find all text-boxes in a set of input controls.
- **FilterByTypes()** method finds and returns controls whose type is any one of given list of types. For example, you can find all text-boxes and combo-boxes in a set of input controls.
- **FilterByProperty()** method finds and returns controls whose property value matches a specified criteria. The string representation of property value is used to find the controls.
- **RemoveByType()** method removes controls of specified type from the *ControlSet*.
- **RemoveByTypes()** method removes controls (from *ControlSet*) whose type is any one of given list of types.
- **Not()** method removes the given list of controls and returns the resultant *ControlSet*.
- **Even()** returns even controls. Since index is zero-based, it returns first-control, third-control, and so on.
- **Odd()** returns odd controls. Since index is zero-based, it returns second-control, fourth-control, and so on.
- **Gt()** returns all controls at index greater than specified index. Since index is zero-based, it returns (i + 1)th to (n - 1)th controls.
- **Lt()** returns all controls at index lesser than specified index. Since index is zero-based, it returns (0)th to (i - 1)th controls.
- **Enabled()** returns all controls that are enabled. This applies to controls that extend from *Control* class.
- **Disabled()** returns all controls that are enabled. This applies to controls that extend from *Control* class.
- **Visible()** returns all controls that are visible. This applies to controls that extend from *UIElement* class.

- **Invisible()** returns all controls that are invisible. This applies to controls that extend from *UIElement* class.
- **Checked()** returns all toggle-buttons (like checkboxes) whose checked status is true. This applies to controls that extend from *ToggleButton* class.
- **Unchecked()** returns all toggle-buttons (like checkboxes) whose checked status is false. This applies to controls that extend from *ToggleButton* class.

Value-Related Methods

These methods operate on property values.

- **SetValue()** method sets the specified property of all controls in the *ControlSet* to a specified value.
- **GetValue()** method gets the specified property value of all controls in the *ControlSet*. It returns *List<object>* instance.
- **SetBinding()** method attaches a binding to the matched controls, using the provided binding object.
- **ClearValue()** method clears the local value of a specified property.
- **Val()** method returns current value of first control in the *ControlSet*. For *TextBlock* and *TextBox*, it returns Text property; for Selector controls (like *ListBox*, *ComboBox*), it returns *SelectedItem* property.
- **Data()** method gets or sets an arbitrary value to all controls in the *ControlSet*. Any number of objects can be assigned to a control, each value identified by a string (key). This is similar to having multiple Tag properties for a *Silverlight* control.

Visibility-Related Methods

These methods are used to hide/show controls, with optional animation. The speed of animation and the opacity can be specified through arguments.

- **Hide()** method hides all the controls in *ControlSet* (without animation).
- **Show()** method displays all the controls in *ControlSet* (without animation).
- **Toggle()** method alternates the visibility of controls (without animation).
- **FadeOut()** method hides the controls by fading them to transparent.
- **FadeIn()** method displays the controls by fading them to opaque.
- **FadeTo()** method sets the opacity of controls to a specified value.
- **FadeToggle()** method alternates the transparency of controls.
- **SlideUp()** method hides the matched controls with a sliding motion.
- **SlideDown()** method displays the matched controls with a sliding motion.

- **SlideTo()** method sets the height of matched controls to a specified value.
- **SlideToggle()** method displays or hides the matched controls with a sliding motion.

Animation Methods

These methods are used to carry out a custom animation on a given property.

- **Animate()** method animates a control based on a given property. The speed of animation and begin/end values for property can be specified through arguments. An optional callback function can be specified, which is invoked after the animation is finished. This method currently supports animation for double properties (like *Opacity*, *Width*, *Height*).

Event-Related Methods

These methods simplifies the tasks related to event-handling. The events can be specified by high-level event names, defined by *EventType* enumeration. For more information, see *Event Handling in XamlQuery* section below.

- **AddHandler()** method adds a routed event handler for a specified routed event of matched controls.
- **RemoveHandler()** method removes the specified routed event handler from matched controls.
- **Bind()** method binds a event handler method to an event specified by *EventType*. The *EventType* is an enumeration that assists event-handing mechanism using high-level names. Event-handlers added through this method can be invoked manually without the actual event being occurred.
- **Trigger()** method invokes an event attached using Bind method. The event specified by *EventType* is fired immediately without the actual event (like mouse moved or key pressed by user) being occurred.

Each event specified by *EventType* has two helper methods associated with it. These methods are used to trigger or bind the corresponding event. These methods are actually shortcuts to *Bind()* and *Trigger()* methods.

Layout-Related Methods

These methods operate on layout attributes like position, size, etc.

- **Detach()** method deletes/removes all controls in the *ControlSet* from the rendered output. The removed controls will no longer be available in the rendered output of *Silverlight*. The

controls that are inside a container control (like Grid, StackPanel, WrapPanel, etc.) can be detached/removed. The container controls are those that are extended from Panel.

- **DetachByType()** method deletes/removes all controls of specified type from the rendered output.
- **DetachByTypes()** method deletes/removes all controls of specified types from the rendered output.
- **Empty()** method removes all controls from a container control (like Grid, StackPanel, WrapPanel, etc.)
- **Width()** method returns the actual-width of first control in the *ControlSet*.
- **Height()** method returns the actual-height of first control in the *ControlSet*.
- **Position()** method returns the position of first control in the *ControlSet*. The position is calculated with reference to its immediate parent or container control.

Manipulating the *ControlSet*

Most of the methods of *ControlSet* simplifies a complex task and can be invoked in a single line. The following example changes the stroke-thickness and stroke-fill properties of all rectangles in a canvas control. This example is illustrated in *Examples and Demonstrations* section below.

```
XamlQuery.ByType<Rectangle>(MainCanvas).SetValue(Shape.StrokeThicknessProperty,4.0);
XamlQuery.ByType<Rectangle>(MainCanvas).SetValue(Shape.FillProperty, new
SolidColorBrush(Colors.LightGray));
```

These two lines are equivalent to the following:

```
XamlQuery.Search(MainCanvas,"Rectangle").SetValue(Shape.StrokeThicknessProperty,4.0);
XamlQuery.Search(MainCanvas,"Rectangle").SetValue(Shape.FillProperty, new
SolidColorBrush(Colors.LightGray));
```

Throughout *XamlQuery*, the *DependencyObject* class is used instead of control types, so that any type of *Silverlight* controls can be used as arguments and return types in *XamlQuery*. For control-specific operations on *ControlSet*, each element has to be converted (type casting) to appropriate control type. For example, the following code finds the total length of text in all Textbox controls inside a Grid.

```
ControlSet allTextBoxes = XamlQuery.ByType<TextBox>(RegisterGrid);

int totalLength = 0;
allTextBoxes.ForEach(delegate(DependencyObject child)
{
    TextBox textBox = (TextBox)child;
    totalLength += textBox.Text.Length;
});
```

VI. Animation in *XamlQuery*

XamlQuery provides some methods that simplifies the animation related tasks.

The *Animate()* method can be used to carry out a custom animation on a specified property. The speed of animation and begin/end property values can be specified through arguments to this method. Currently, this method supports animation on properties of data-type double (like *Opacity*, *Width*, *Height*, etc.)

The *FadeOut()*, *FadeIn()*, *FadeTo()*, *FadeToggle()*, *SlideUp()*, *SlideDown()*, *SlideTo()* and *SlideToggle()* methods can be invoked in animation-mode using the optional parameter *Speed* or custom duration (in milliseconds).

The *Speed* is an enumeration constant with the following values.

- *Normal (1000 milliseconds)*
- *Slow (2000 milliseconds)*
- *VerySlow (3000 milliseconds)*
- *Fast (500 milliseconds)*
- *VeryFast (200 milliseconds)*

For demonstration of these animation methods, see *Examples and Demonstrations* section below.

VII. Event Handling in *XamlQuery*

Events in XamlQuery

The *AddHandler()* and *RemoveHandler()* methods of *XamlQuery* are used to attach and detach routed events to all the controls in *ControlSet*. In additional to these methods, *XamlQuery* defines an enumeration *EventType* for easier handling of events. Any method derived from *Delegate* class can be attached to any event, unless the signature of event-arguments match.

The following are the values of the enumeration constant *EventType*.

Events of controls extended from *UIElement*

- *GotFocus*
- *LostFocus*
- *KeyDown*
- *KeyUp*
- *MouseEnter*
- *MouseLeave*
- *MouseMove*
- *LostMouseCapture*
- *MouseLeftButtonDown*
- *MouseLeftButtonUp*
- *MouseWheel*

Events of controls extended from *FrameworkElement*

- *BindingValidationError*
- *Loaded*
- *LayoutUpdated*
- *SizeChanged*

Events of controls extended from *Control*

- *IsEnabledChanged*

Events of *TextBox*

- *TextChanged*
- *TextSelectionChanged*

Events of controls extended from *Selector*

- *SelectionChanged*

XamlQuery defines a method for each of these events. These methods are used as shortcuts for *Bind()* and *Trigger()* methods.

Examples of Events

There are several ways to bind a event-handler to an event. The following example illustrates these.

```
ControlSet allShapes = XamlQuery.ByType<Rectangle>(MainCanvas);

//method-1
allShapes.AddHandler(UIElement.MouseLeftButtonDownEvent, new
MouseButtonEventHandler(delegate(object senderObject, MouseButtonEventArgs eventArgs)
{
    Rectangle rectangle = (Rectangle)senderObject;
    rectangle.SetValue(Shape.FillProperty, new SolidColorBrush(Colors.Green));
}));

//method-2
allShapes.Bind(EventType.MouseEnter, new MouseEventHandler(delegate(object
senderObject, MouseEventArgs eventArgs)
{
    Rectangle rectangle = (Rectangle)senderObject;
    rectangle.SetValue(Shape.FillProperty, new SolidColorBrush(Colors.Green));
}));

//method-3
allShapes.Bind(EventType.MouseEnter, new MouseEventHandler(rectangle_MouseEnter));

//method-4
allShapes.MouseEnter(new MouseEventHandler(delegate(object senderObject,
MouseEventArgs eventArgs)
{
    Rectangle rectangle = (Rectangle)senderObject;
    rectangle.SetValue(Shape.FillProperty, new SolidColorBrush(Colors.Green));
}));

//method-5
allShapes.ForEach(delegate(DependencyObject child)
{
    ((UIElement)child).MouseEnter += rectangle_MouseEnter;
});
```

The *rectangle_MouseEnter* used in this example is a event-handler method, defined below.

```
void rectangle_MouseEnter(object sender, MouseEventArgs e)
{
    Rectangle rectangle = (Rectangle)sender;
    rectangle.SetValue(Shape.FillProperty, new SolidColorBrush(Colors.Green));
}
```

VIII. Examples and Demonstrations

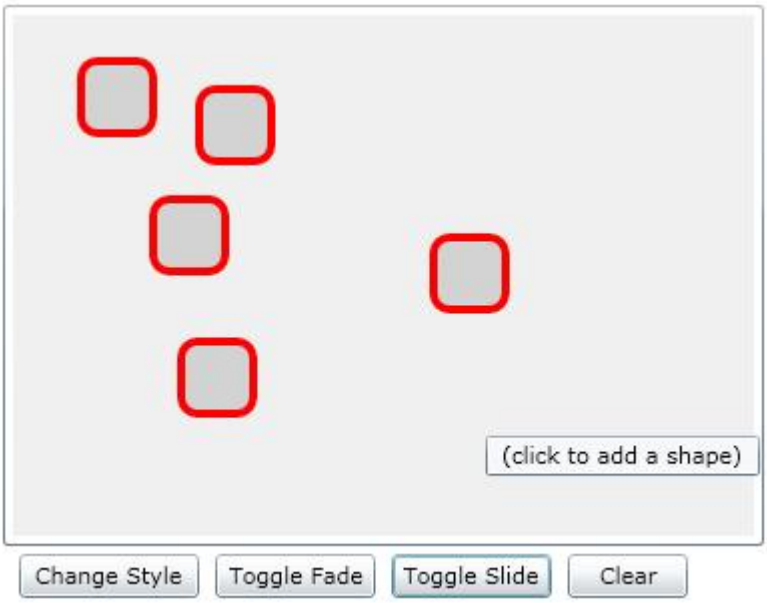
Examples of *XamlQuery*

The following are some examples on the usage of *XamlQuery*. These examples illustrate that *XamlQuery* can help simplify complex tasks for developers.

```
//hide all lines (method-1)
XamlQuery.ByType<Line>(MainCanvas).Hide();

//hide all lines (method-2)
XamlQuery.Search(MainCanvas, "Line").Hide();

//disable all textboxes (method-1)
XamlQuery.ByType<TextBox>(RegisterGrid).SetValue(Control.IsEnabledProperty, false);

//disable all textboxes (method-2)
XamlQuery.Search(RegisterGrid, "TextBox").SetValue(Control.IsEnabledProperty, false);

//remove border of all textboxes
XamlQuery.ByType<TextBox>(RegisterGrid).SetValue(Control.BorderBrushProperty, new
SolidColorBrush(Colors.Transparent));

//set font-style of all textboxes
XamlQuery.ByType<TextBox>(RegisterGrid).SetValue(Control.FontStyleProperty,
FontStyles.Italic);

//change IsTabStop and Foreground of all hyperlinks
XamlQuery.ByType<HyperlinkButton>(LayoutRoot).SetValue(Control.IsTabStopProperty,
false);
XamlQuery.ByType<HyperlinkButton>(LayoutRoot).SetValue(Control.ForegroundProperty, new
SolidColorBrush(Colors.Black));

//find all controls with name "ArrowLine" (method-1)
ControlSet allArrowLines = XamlQuery.ByProperty(MainCanvas,
FrameworkElement.NameProperty, "ArrowLine");

//find all controls with name "ArrowLine" (method-2)
ControlSet allArrowLines = XamlQuery.Search(MainCanvas,
"*[FrameworkElement_Name=ArrowLine]");
```

Demonstration of *XamlQuery* using a *Canvas*

The following image is the screenshot of a demo that illustrates the usage of *XamlQuery* with some animation and style related tasks using a Canvas control.



The event-handler of the buttons in this demo are shown below.

```
private void ChangeStyleButton_Click(object sender, RoutedEventArgs e)
{
    //find all rectangles inside "MainCanvas"
    ControlSet allShapes = XamlQuery.Search(LayoutRoot, "#MainCanvas Rectangle");

    //set style for all rectangles
    allShapes.SetValue(Shape.StrokeThicknessProperty, 4.0);
    allShapes.SetValue(Shape.FillProperty, new SolidColorBrush(Colors.LightGray));
    allShapes.SetValue(Rectangle.RadiusXProperty, 8.0);
    allShapes.SetValue(Rectangle.RadiusYProperty, 8.0);

    //set mouse-enter event for all rectangles
    allShapes.Bind(EventType.MouseEnter, new MouseEventHandler(delegate(object
senderObject, MouseEventArgs eventArgs)
    {
        Rectangle rectangle = (Rectangle)senderObject;
        rectangle.SetValue(Shape.FillProperty, new SolidColorBrush(Colors.Green));
    }));
}

private void ToggleFadeButton_Click(object sender, RoutedEventArgs e)
{
    XamlQuery.Search(MainCanvas, "Rectangle").FadeToggle(Speed.Fast);
}

private void ToggleSlideButton_Click(object sender, RoutedEventArgs e)
```

```
{
    XamlQuery.ByType<Rectangle>(MainCanvas).SlideToggle(Speed.Fast);
}

private void ClearButton_Click(object sender, RoutedEventArgs e)
{
    //XamlQuery.All(MainCanvas).Detach(); //method-1
    XamlQuery.Search(MainCanvas, "*").Detach(); //method-2
}
```

Demonstration of *XamlQuery* using a simple Form

The following image is the screenshot of a demo that illustrates the usage of *XamlQuery* using a simple registration form.

The event-handler of the buttons in this demo are shown below.

```
private void RegisterButton_Click(object sender, RoutedEventArgs e)
{
    //validate controls marked compulsory (faster version)
    bool allValid = true;

    XamlQuery.Search(LayoutRoot, "Control[FrameworkElement_Tag=Compulsory]").
        ForEach(delegate(DependencyObject obj)
    {
        if (ControlSet.Create(obj).Val().Equals(string.Empty))
        {
            MessageBox.Show("Enter Value for " + ((Control)obj).Name);
            allValid = false;
            return;
        }
    });

    if (allValid)
    {
        MessageBox.Show("You are ready to be registered.");
    }
}

private void ChangeStyleButton_Click(object sender, RoutedEventArgs e)
{
    ControlSet inputControls = XamlQuery.ByType<Control>(LayoutRoot);
```

```
    inputControls.RemoveByType<Button>();
    inputControls.SetValue(Control.BorderBrushProperty, new
SolidColorBrush(Colors.Green));
    inputControls.SetValue(Control.BackgroundProperty, new
SolidColorBrush(Colors.LightGray));
    inputControls.SetValue(Control.FontWeightProperty, FontWeights.SemiBold);
    inputControls.SetValue(Control.FontStyleProperty, FontStyles.Italic);
}
```

In this example, the input controls (like *TextBox*, *ComboBox*) whose input is mandatory are marked as "Compulsory" using Tag property.

IX. Frequently Asked Questions

1) While using style selectors, styles should be defined in the same *UserControl*/*Page*?

   No, styles can reside in same *UserControl*/*Page* or any of merged resource dictionaries or *App.xaml* also. *XamlQuery* will search for styles in all these places.

2) While using style selectors, using a base style will match controls of its dependent styles?

   Yes, using a base style in selector will match controls whose style is extended from that base-style. For example, if style 'GreenRect' and 'BlueRect' are based on style 'BaseRect', then the selector query '*.BaseRect' will match both green and blue rectangles.

3) Is type selectors can be used to find subclasses of a control type?

   Yes, type-selectors match controls of sub classes also. For example, the selector query 'Shape' will match controls of type *Rectangle*, *Line*, *Ellipse*, etc. Similarly, the selector query 'Selector' will match controls of type *ComboBox*, *ListBox*, etc.

4) Can we use multiline selectors?

   Yes, a selector can span to multiple lines.

5) Can we use any number of white-spaces inside a query string?

   Yes, there is no restriction in number of white-spaces in a selector. However, the location of white-space between two simple-selectors is significant.

6) Should dependency property names end with 'Property'?

   No, the 'Property' keyword should be ignored while using a dependency property in a selector. For example, *UIElement.OpacityProperty* should be given as 'UIElement_Opacity'.

7) Why Search() method requires a reference control as argument?

   The reference control is used to limit the search scope. The search scope is very important, because of *Silverlight's* behavior of creating additional controls in the rendered output. The lesser the search scope, the higher the accuracy of matching. For example, if the user needs to find

controls inside a *Grid*, the reference control shall be that *Grid* instead of root control in the Page/UserControl.

8) Is "*.GreenRect" and "* .GreenRect" are different?

Yes, white-space is a symbol used for descendant selectors. '*.GreenRect' will match all controls with style 'GreenRect'. '* .GreenRect' will match all controls with 'GreenRect' that are descendants of some other control in the rendered output.