Flextras AutoCompleteComboBox Component Documentation

www.flextras.com

Contents

Welcome

Hi, and thanks for checking out the Flextras AutoCompleteComboBox Component. AutoComplete allows you to filter a display list as you type, and you're probably familiar with AutoComplete functionality from the address bar of your browser and the addressing field of your e-mail client. With the Flextras AutoCompleteComboBox, you can provide this functionality to the users of your Flex Application.

This document is intended to get you started with our AutoCompleteComboBox. Believe it or not, we have four different implementations of our AutoCompletComboBox:

- **Flex 3 Version**: This is our original implementation. It is now deprecated
- **Flex 3.5 Version**: The Flex 3.5 SDK changed how the ComboBox dealt with the drop down, meaning we had to rework our component to work with Flex 3.5.
- **Flex 4 MX Version**: This is the Flex 3.5 version compiled against the Flex 4 SDK.
- **Flex 4 Spark Lite Version:** The Spark architecture is the future of Flex. All of our major development efforts will focus on building out the API of our Spark AutoCompleteComboBox. You can use our Spark AutoCompleteComboBoxLite component for free, although the API is not fully implemented yet. (Purchase the unlimited domain edition get the full source code and updates).

Unless otherwise noted, this document focuses on our Spark AutoCompleteComboBox, but many of the concepts and APIs we discuss in this document also apply to our MX implemented version.

For full details on the API, you can review the ASDocs included in the free developer edition download, or on the Flextras web site. The download also includes sample code, so be sure to check out those samples.

Thanks for taking a look. We are here to help, so don't hesitate to drop us an e-mail, send us an IM, or give us call with your questions.

Jeffry Houser (The Brains Behind Flextras)

Jeffry@dot-com-it.com | reboog711@gmail.com | 203-379-0773

A Note on Sample Code

The source code for all sample code in this document is included as part of the developer edition download. Flex 3 specific samples are located in the file *DocumentationSamples.zip*. Flex 4 specific samples are included in *DocumentationSamples4.zip*. Where appropriate, we list the relevant files at the beginning of each instruction.

Document History

Date	Description
	• Added more FAQ
2/3/2011	• Added FAQ Section and started answering questions
1/20/2011	• Initial Creation

Introduction

This section will give you an introduction to the using the Flextras Spark AutoCompleteComboBoxLite component.

Creating a Simple AutoComplete

Introduction

This example will show you how to use the Flextras AutoCompleteComboBoxLite.

Files used

- Sample1.mxml

Versions This Tutorial Applicable To

- Flex 3 Version
- Flex 3.5 Version
- Flex 4 MX Version
- Flex 4 Spark Version

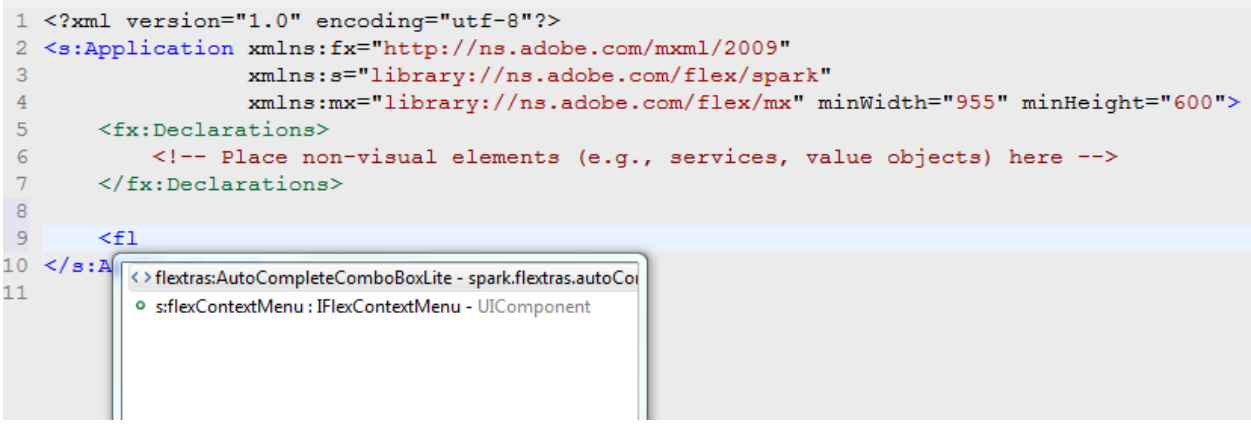
Tutorial

First create a new Flex Project in your application and add the SWC to your library path. More information on how to use SWCs is located in this blog post http://www.flextras.com/blog/index.cfm/2009/12/18/What-do-you-do-with-a-SWC .

Create a new application; it should look something like this:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx"
              minWidth="955" minHeight="600">

</s:Application>
```

Start typing “<fl” in the main application and you should see a list of all Flextras components in the library, like below:



Select the AutoCompleteComboBoxLite and hit enter. The AutoCompleteComboBox code will automatically drop in:

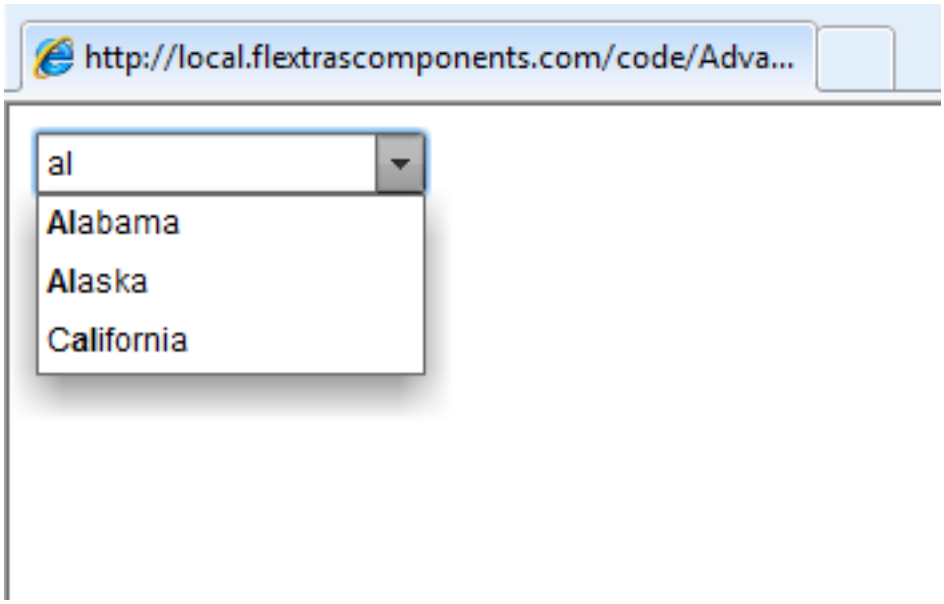
```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                    xmlns:s="library://ns.adobe.com/flex/spark"
                    xmlns:mx="library://ns.adobe.com/flex/mx"
                   minWidth="955" minHeight="600"
                   xmlns:flextras="http://www.flextras.com/mxml">

      <flextras:AutoCompleteComboBoxLite >
</s:Application>
```

You'll see that the Flextras namespace was automatically added to the main application tag and the component now resides in the body. The purpose of an AutoComplete component is to be able to filter data as the user types. To make that happen, you'll have to specify a dataProvider. For the purpose of this sample, I'll give you a hard coded dataProvider, but in your own applications, you can source it from any XML or backend, just like you were using a regular Spark ComboBox or DropDownList:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                  xmlns:s="library://ns.adobe.com/flex/spark"
                  xmlns:mx="library://ns.adobe.com/flex/mx"
                  minWidth="955" minHeight="600"
                  xmlns:flextras="http://www.flextras.com/mxml">

      <fx:Script>
            <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            public var dataProvider : ArrayCollection =
                  new ArrayCollection([
                        {label:'Alabama',data:1},
                        {label:'Alaska',data:2},
                        {label:'Arizona',data:3},
                        {label:'Arkansas',data:4},
```

```
                        {label:'California',data:5},
                    ]);
                ]]>
        </fx:Script>

        <flextras:AutoCompleteComboBoxLite
                    id="accb"
                    dataProvider="{this.dataProvider}" />
</s:Application>
```

For the purposes of this document, I kept the dataProvider short, but the sample in the documentation lists all the US States. Execute your sample, and you should see, something like this



Start typing and watch the dataProvider filter.

Final Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                    xmlns:s="library://ns.adobe.com/flex/spark"
                    xmlns:mx="library://ns.adobe.com/flex/mx"
                    minWidth="955" minHeight="600"
                    xmlns:flextras="http://www.flextras.com/mxml">

        <fx:Script>
                <![CDATA[
                import mx.collections.ArrayCollection;

                [Bindable]
                public var dataProvider : ArrayCollection =
                        new ArrayCollection([
                                {label:'United States of America',data:0},
```

```
                {label:'Alabama',data:1},
                {label:'Alaska',data:2},
                {label:'Arizona',data:3},
                {label:'Arkansas',data:4},
                {label:'California',data:5},
                {label:'Colorado',data:6},
                {label:'Connecticut',data:7},
                {label:'Delaware',data:8},
                {label:'Florida',data:9},
                {label:'Georgia',data:10},
                {label:'Hawaii',data:11},
                {label:'Idaho',data:12},
                {label:'Illinois',data:13},
                {label:'Indiana',data:14},
                {label:'Iowa',data:15},
                {label:'Kansas',data:16},
                {label:'Kentucky',data:17},
                {label:'Louisiana',data:18},
                {label:'Maine',data:19},
                {label:'Maryland',data:20},
                {label:'Massachusetts',data:21},
                {label:'Michigan',data:22},
                {label:'Minnesota',data:23},
                {label:'Mississippi',data:24},
                {label:'Missouri',data:25},
                {label:'Montana',data:26},
                {label:'Nebraska',data:27},
                {label:'Nevada',data:28},
                {label:'New Hampshire',data:29},
                {label:'New Jersey',data:30},
                {label:'New Mexico',data:31},
                {label:'New York',data:32},
                {label:'North Carolina',data:33},
                {label:'North Dakota',data:3},
                {label:'Ohio',data:35},
                {label:'Oklahoma',data:36},
                {label:'Oregon',data:37},
                {label:'Pennsylvania',data:38},
                {label:'Rhode Island',data:39},
                {label:'South Carolina',data:40},
                {label:'South Dakota',data:41},
                {label:'Tennessee',data:42},
                {label:'Texas',data:43},
                {label:'Utah',data:44},
                {label:'Vermont',data:45},
                {label:'Virginia',data:46},
                {label:'Washington',data:47},
                {label:'West Virginia',data:48},
                {label:'Wisconsin',data:49},
                {label:'Wyoming',data:50},
        ]);
]]>
```

```
	</fx:Script>

	<flextras:AutoCompleteComboBoxLite
		id="accb"
		dataProvider="{this.dataProvider}"
		x="10" y="10"
	/>
</s:Application>
```

Customize the drop down filtering

Introduction

The default filter for the autoComplete will filter items by looking into your items label, and if the text is found then that item stays in the list. If you want to use a different algorithm, that is possible using the filterFunction property of the component. This sample will show you how to add a new function that will only filter items based on the beginning of the item, not the middle.

Versions This Tutorial Applicable To

- Flex 3 Version
- Flex 3.5 Version
- Flex 4 MX Version
- Flex 4 Spark Version

Version Notes

- In the Spark version, the filter function property is named filterFunction .
- In the Halo version, the filter function property is named autoCompleteFilterFunction

Files Used

- Sample2.mxml

Tutorial

I'm going to base the code in this tutorial off the previous one. You should create a new application file, create an instance of the AutoCompleteComboBox, and add in a dataProvider. You're app should look something like this:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               minWidth="955" minHeight="600"
               xmlns:flextras="http://www.flextras.com/mxml">

	<fx:Script>
		<![CDATA[
```

```
                import mx.collections.ArrayCollection;

                [Bindable]
                public var dataProvider : ArrayCollection =
                      new ArrayCollection([
                      {label:'Alabama',data:1},
                      {label:'Alaska',data:2},
                      {label:'Arizona',data:3},
                      {label:'Arkansas',data:4},
                      {label:'California',data:5},
                      {label:'Colorado',data:6},
                      {label:'Connecticut',data:7},
                ]);

            ]]>
      </fx:Script>

      <flextras:AutoCompleteComboBoxLite id="accb"
                      dataProvider="{this.dataProvider}"
                      />
</s:Application>
```

The next step is to create a new filter function, which we can start like this:

```
protected function firstOnlyFilter ( item:Object):Boolean{

      return false;
}
```

As the AutoCompleteComboBox filters items, the filter function is called for each item in your dataProvider. If the filter function returns true, then that item stays in the resultinglist. If the filter function returns false, the, item is removed. The function, as it stands, will remove every item in the list, since it always returns true. The next step is to add a condition to see if the item should be kept in the filtered dataProvider or not.

To make the determination as to whether the item matches or not, you can use the search() method of the String class. In order to make the search non case sensitive, the toLowerCase() will also be used. First, get the label of the item:

```
var label : String = item.label;
```

Then use the search method to compare it to the typeAheadText of the AutoComplete component:

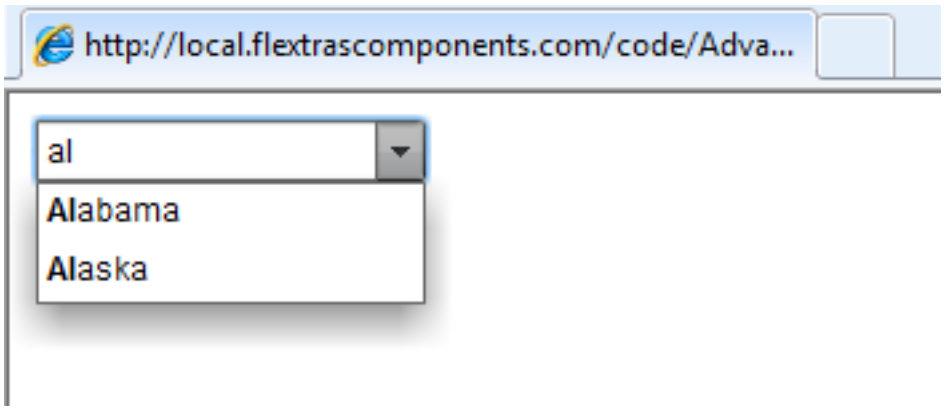
```
if(label.toLowerCase().search(accb.typeAheadText.toLowerCase() == 0)
 return true;
}
```

If the search method returns 0 then the text the user typed is equal to the start of the string, and therefore kept in the result set. If the search method returns false, then the condition isn't executed, and the filter function will return false, which is the default.

Here is the complete filterFunction:

```
protected function firstOnlyFilter ( item:Object):Boolean{

      var label : String = item.label;

      if(label.toLowerCase().search(accb.typeAheadText.toLowerCase())
            == 0){
           return true;
      }

      return false;
}
```

Here is a screenshot of the app with the custom filter function:



You can use the filterFunction API of the component to provide any customized sorting mechanisms, such as comparing on multiple properties of your dataPRovider objects, turning on or off case sensitivity of the comparison, or drilling down into properties on complicated data structures.

Final Code

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                xmlns:s="library://ns.adobe.com/flex/spark"
                xmlns:mx="library://ns.adobe.com/flex/mx"
                minWidth="955" minHeight="600"
                xmlns:flextras="http://www.flextras.com/mxml">

<fx:Script>
```

```
<![CDATA[
	import mx.collections.ArrayCollection;

	[Bindable]
	public var dataProvider : ArrayCollection =
	new ArrayCollection([
		{label:'United States of America',data:0},
		{label:'Alabama',data:1},
		{label:'Alaska',data:2},
		{label:'Arizona',data:3},
		{label:'Arkansas',data:4},
		{label:'California',data:5},
		{label:'Colorado',data:6},
		{label:'Connecticut',data:7},
		{label:'Delaware',data:8},
		{label:'Florida',data:9},
		{label:'Georgia',data:10},
		{label:'Hawaii',data:11},
		{label:'Idaho',data:12},
		{label:'Illinois',data:13},
		{label:'Indiana',data:14},
		{label:'Iowa',data:15},
		{label:'Kansas',data:16},
		{label:'Kentucky',data:17},
		{label:'Louisiana',data:18},
		{label:'Maine',data:19},
		{label:'Maryland',data:20},
		{label:'Massachusetts',data:21},
		{label:'Michigan',data:22},
		{label:'Minnesota',data:23},
		{label:'Mississippi',data:24},
		{label:'Missouri',data:25},
		{label:'Montana',data:26},
		{label:'Nebraska',data:27},
		{label:'Nevada',data:28},
		{label:'New Hampshire',data:29},
		{label:'New Jersey',data:30},
		{label:'New Mexico',data:31},
		{label:'New York',data:32},
		{label:'North Carolina',data:33},
		{label:'North Dakota',data:3},
		{label:'Ohio',data:35},
		{label:'Oklahoma',data:36},
		{label:'Oregon',data:37},
		{label:'Pennsylvania',data:38},
		{label:'Rhode Island',data:39},
		{label:'South Carolina',data:40},
		{label:'South Dakota',data:41},
		{label:'Tennessee',data:42},
		{label:'Texas',data:43},
		{label:'Utah',data:44},
		{label:'Vermont',data:45},
```

```
                {label:'Virginia',data:46},
                {label:'Washington',data:47},
                {label:'West Virginia',data:48},
                {label:'Wisconsin',data:49},
                {label:'Wyoming',data:50},
            );

            protected function firstOnlyFilter ( item:Object):Boolean{

                var label : String = item.label;


      if(label.toLowerCase().search(accb.typeAheadText.toLowerCase())
== 0){
                return true;
            }

            return false;
      }
            ]]>
      </fx:Script>

      <flextras:AutoCompleteComboBoxLite id="accb"
                dataProvider="{this.dataProvider}"
                x="10" y="10"
                filterFunction="{this.firstOnlyFilter}"
            />
</s:Application>
```

Change the autoComplete highlight in the drop down list?

Introduction

The default itemRenderer of the AutoCompleteComboBox will highlight the text the user typed by bolding it in the label of your dataProvider's item. Through the use of itemRenderers, this is completely customizable. This will show you how to underline the text instead of bolding it.

Versions This Tutorial Applicable To

- Flex 4 Spark Version

Files Used

- Sample3.mxml
- com.flextras.autoCompleteRenderer.CustomAutoCompleteRenderer

Tutorial

The AutoComplete component will support any itemRenderer you want to use, but for best results with our AutoCompleteComboBox component we recommend that your itemRenderer implement our

IAutoCompleteRenderer interface. There are two properties defined in this interface that you'll have to implement:

- **autoCompleteComboBox**: A reference to the AutoCompleteComboBox instance that created the renderer
- **typeAheadText**: The text that the user typed into the input in order to filter the text.

This tutorial will tell you everything you need to know about creating a custom itemRenderer.

First, create a new MXML Component that extends the DefaultItemRenderer. The DefaultItemRenderer component is the default item renderer for the Flex ComboBox, which our component extends. You don't need to extend it to create your own renderer, but for the purposes of this sample, we will. You should see something like this:

```
<spark:DefaultItemRenderer xmlns:fx="http://ns.adobe.com/mxml/2009"
         xmlns:s="library://ns.adobe.com/flex/spark"
         xmlns:mx="library://ns.adobe.com/flex/mx"
         xmlns:spark="spark.skins.spark.*">

</spark:DefaultItemRenderer>
```

You'll want to make sure this component implements the IAutoCompleteRenderer interface, so add that in, and you'll get this:

```
<spark:DefaultItemRenderer xmlns:fx="http://ns.adobe.com/mxml/2009"
      xmlns:s="library://ns.adobe.com/flex/spark"
      xmlns:mx="library://ns.adobe.com/flex/mx"
      xmlns:spark="spark.skins.spark.*"
      implements="spark.flextras.autoCompleteComboBox.renderers.IAutoCo
      mpleteRenderer">
      <fx:Script>
            <![CDATA[
                  import
spark.flextras.autoCompleteComboBox.renderers.IAutoCompleteRenderer;
            ]]>
      </fx:Script>

</spark:DefaultItemRenderer>
```

Next, you'll want to implement the properties in the interface. Properties can't technically be defined in an interface, but get and set methods can be, which is how we implemented it:

```
private var _autoCompleteComboBox : AutoCompleteComboBoxLite;
public function get autoCompleteComboBox():AutoCompleteComboBoxLite{
```

```
      return this._autoCompleteComboBox;
}
public function set autoCompleteComboBox(
                              value:AutoCompleteComboBoxLite):void{
      this._autoCompleteComboBox = value;
}

private var _typeAheadText : String;
public function get typeAheadText():String{
      return this._typeAheadText;
}
public function set typeAheadText(value:String):void{
      this._typeAheadText = value;
}
```

The code block just contains simple implementations of the get and set methods. A private variable contains the value, and the get and set methods expose them.

The DefaultItemRenderer creates a labelDisplay component for the display of our label. The labelDisplay component extends TextBase. However, to massage the data before displaying it you can replace this with an instance of a RichText component. Override the createChildren() method to make that happen:

```
override protected function createChildren():void{
      if (!labelDisplay)    {
            labelDisplay = new RichText();
            addChild(DisplayObject(labelDisplay));
      }
      super.createChildren();
}
```

This will create our label field and give us flexibility to style the text in order to create the highlight. The next step is to create the highlight. As with most itemRenderers, you can change the displayed data using the dataChange event. Add an event listener to the top level application tag:

```
<spark:DefaultItemRenderer xmlns:fx="http://ns.adobe.com/mxml/2009"
      xmlns:s="library://ns.adobe.com/flex/spark"
      xmlns:mx="library://ns.adobe.com/flex/mx"
      xmlns:spark="spark.skins.spark.*"
      implements="spark.flextras.autoCompleteComboBox.renderers.IAutoCo
      mpleteRenderer"
      dataChange="onDataChange(event)">
```

This is the event handler stub:

```
protected function onDataChange(event:FlexEvent):void{
```

```
}
```

If the typeAheadText is empty, you don't need to worry about special formatting on the renderer. That can be accomplished like this:

```
if(this.typeAheadText != ''){

} else {
 labelDisplay.text = this.autoCompleteComboBox.itemToLabel(data);
}
```

To create the highlighted text, you'll have to make use of the new Text Layout Framework. First, get the label as a RichText element:

```
var labelAsRichText : RichText = this.labelDisplay as RichText;
```

You can use a Regex to parse the label text to find the spots that need formatting:

```
var regEx : RegExp = new RegExp(this.typeAheadText ,'i');
```

The 'I' flag on the regular expression tells the expression to ignore case sensitivity. Next, use the regular expression to turn the selected text into HTML Text:

```
var htmlText : String =
     this.autoCompleteComboBox.itemToLabel(data).replace(
          regEx ,
          '<u>$&</u>'
     )
```

This line uses the itemToLabel method of the AutoCompleteComboBox to turn the object into a label for display. Finally use some Text Layout Framework magic to translate the underlined HTML text into something displayable:

```
labelAsRichText.textFlow = TextConverter.importToFlow
                                htmlText,
                                TextConverter.TEXT_FIELD_HTML_FORMAT);
```

The TextConvertor method magically knows how to handle our html formatted text so it will display properly with the Rich Text component. This is the full data change method:

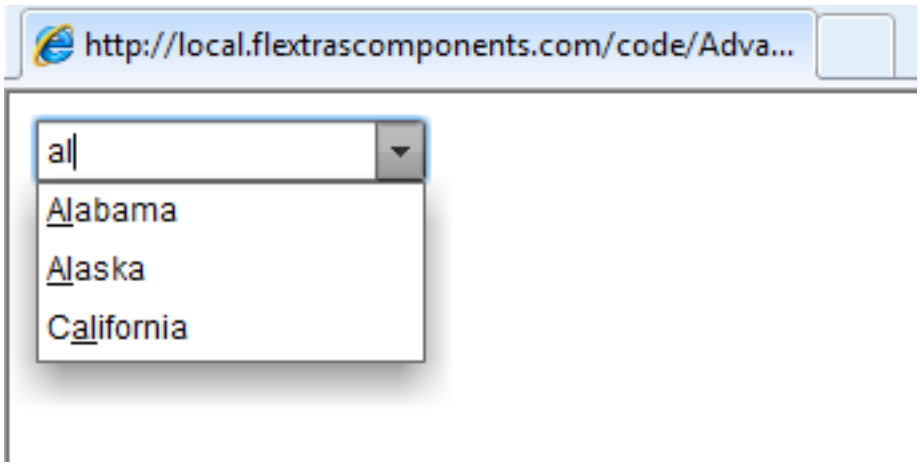
```
protected function onDataChange(event:FlexEvent):void{
```

```
        if(this.typeAheadText != ''){
            var labelAsRichText : RichText =
                                        this.labelDisplay as RichText;
            var regEx : RegExp = new RegExp(this.typeAheadText ,'i');

            var htmlText : String =
                        this.autoCompleteComboBox.itemToLabel(data).
                              replace(regEx , '<u>$&</u>')

            labelAsRichText.textFlow = TextConverter.importToFlow
                                    htmlText,
                                    TextConverter.TEXT_FIELD_HTML_FORMAT);
        } else {
            labelDisplay.text =
                        this.autoCompleteComboBox.itemToLabel(data);
        }
}
```

Finally, remember to specify the new itemRenderer in your main application file:

```
<flextras:AutoCompleteComboBoxLite id="accb"
                        dataProvider="{this.dataProvider}"
                        itemRenderer="com.flextras.autoCompleteRenderer.
                                        CustomAutoCompleteRenderer"
                    />
```

You'll want to use fewer line breaks in your own code. The results look like this:



Final Code

```
<?xml version="1.0" encoding="utf-8"?>
<spark:DefaultItemRenderer xmlns:fx="http://ns.adobe.com/mxml/2009"
      xmlns:s="library://ns.adobe.com/flex/spark"
      xmlns:mx="library://ns.adobe.com/flex/mx"
      implements="spark.flextras.autoCompleteComboBox.renderers.IAutoCo
mpleteRenderer"
```

```
      xmlns:spark="spark.skins.spark.*"
      dataChange="onDataChange(event)">

      <fx:Script>
      <![CDATA[
            import flashx.textLayout.conversion.TextConverter;
            import mx.events.FlexEvent;
            import spark.components.RichText;
            import
spark.flextras.autoCompleteComboBox.AutoCompleteComboBoxLite;
            import
spark.flextras.autoCompleteComboBox.renderers.IAutoCompleteRenderer;

            import spark.utils.TextFlowUtil;

            private var _autoCompleteComboBox :
AutoCompleteComboBoxLite;
            public function get
autoCompleteComboBox():AutoCompleteComboBoxLite{
                  return this._autoCompleteComboBox;
            }
            public function set
autoCompleteComboBox(value:AutoCompleteComboBoxLite):void{
                  this._autoCompleteComboBox = value;
            }

            private var _typeAheadText : String;
            public function get typeAheadText():String{
                  return this._typeAheadText;
            }
            public function set typeAheadText(value:String):void{
                  this._typeAheadText = value;
            }

            override protected function createChildren():void{
                  if (!labelDisplay){
                        labelDisplay = new RichText();
                        addChild(DisplayObject(labelDisplay));
                  }
                  super.createChildren();
            }


            protected function onDataChange(event:FlexEvent):void{


                  if(this.typeAheadText != ''){
                        var labelAsRichText : RichText =
                                          this.labelDisplay as RichText;
                        var regEx : RegExp =
                                    new RegExp(this.typeAheadText ,'i');
```

```
                    var htmlText : String =
                        this.autoCompleteComboBox.itemToLabel(data).
                            replace(regEx , '<u>$&</u>')

                    labelAsRichText.textFlow =
                        TextConverter.importToFlow(
                            htmlText,
                            TextConverter.TEXT_FIELD_HTML_FORMAT);
                } else {
                    labelDisplay.text =
                      this.autoCompleteComboBox.itemToLabel(data);
                }
            }

            ]]>
        </fx:Script>


</spark:DefaultItemRenderer>
```

Frequently Asked Questions

This section deals with some commonly asked questions.

If multiple instances of the component are connected to the same dataProvider then things get weird.

We know this happens, although this is not unique to our AutoComplete components. Weird things could happen if you use the same dataProvider for any list based class. With our AutoCompleteComboBox, consider this scenario:

1. Type something in ACCB1. It filters a list down from 50 items to 20. Things work as expected.
2. Type something in ACCB2. Which in turn filters the dataProvider again; which fires off the collectionChange event of the collection; which in turn fires off the collectionChange event handlers in both ACCB1 and ACCB2.

One customer mentioned that in our Halo AutoCompleteComboBox this scenario will clear the text entered into ACCB1 even though he was typing into the second one.

The solution is to create an independent listCollectionView using the same data source for each ACCB. Do something like this:

```
mySecondSource : ArrayCollection = new ArrayCollection(myFirstSource.source);
```

Then you can happily filter one of those sources without affecting the other..

Why do I see a "Implicit coercion of a value of type array to an unrelated type mx.collection:IList" error?

Can I use an Array as a dataProvider to the Spark AutoCompleteComboBox?

You get an error like this with our Spark AutoCompleteComboBox if you use an Array as the dataProvider:

1067: Implicit coercion of a value of type Array to an unrelated type mx.collections:IList.

The Spark ComboBox needs an IList as the dataProvider. For the filtering in our Spark AutoCompleteComboBox to work, you'll also need a dataProvider that implements the IListCollectionView interface. A simple Array fulfills neither of these contracts and cannot be used as the dataProvider. We recommend converting your Array to an ArrayCollection:

```
myNewDP : ArrayCollection = new ArrayCollection( myOldDP);
```

In the MX components, the ComboBox dataProvider would accept any object, and convert it to a ListCollectionView internally.

Why is the selectedIndex wrong in my change event handler?

It is probably not wrong. At the time the change event of the AutoCompleteComboBox fires, the dataProvider has not refreshed itself yet, so the selectedIndex of the AutoCompleteComboBox and in the IndexChangeEvent instance relate to the filtered dataProvider.

The change event does not fire again when the dataProvider filter is removed, however the selectedIndex will stay in sync with the selectedItem's proper place in the dataProvider.

Why won't the AutoCompleteComboBox work if there is a question mark, or other special character, in my drop down?

Our default filterFunction does not handle the special characters, such as question marks. You can write your own filterFunction to handle this if need be.

How can I get the text value of the selected item?

In the MX Version of our AutoComplete, you can the text property to get at the text value of the selectedItem.

The Spark ComboBox API does not have the same property. One approach is to reference the TextInput skin part, like this:

autoCompleteInstance.textInput.text

It should give you the text data you're after.

Another approach is to get the selectedItem property and run it through the itemToLabel function. The code would be like this:

autoCompleteInstance.itemToLabel(autoCompleteInstance.selectedItem);

Why does the MX AutoCompleteComboBox look like a regular ComboBox?

Why is the text MX AutoCompleteComboBox text blurry after I select something?

Either of these situations can occur if you have created a Flex 4 MX only project using the Spark theme. In this situation, TextInput boxes do have a background color set. An easy fix is to address this in CSS. This CSS will set the background of the AutoComplete's TextInput to white, while the text color to black:

```
<fx:Style>
@namespace flextras "http://www.flextras.com/mxml";
@namespace mx "library://ns.adobe.com/flex/mx";

.autoCompleteTextInputStyle{
```

```
      backgroundColor : #FFFFFF;
      color : #000000;
}

flextras|AutoCompleteComboBox{
      autoCompleteTextInputStyleName : "autoCompleteTextInputStyle";
}

</fx:Style>
```

You can also set the autoCompleteTextInputStyleName on each instance of the Flextras AutoComplete for more flexibility:

```
<flextras:AutoCompleteComboBox autoCompleteEnabled="true"
      dataProvider="{dataProvider1}"
      autoCompleteTextInputStyleName="autoCompleteTextInputStyle" />
```

Support

We are more than happy to help out with issues surrounding the AutoCompleteComboBox, or any of our other components. Many of our samples and some of the questions and tutorials in this document came about because people asked us for help.

Here are some ways you can contact us:

- Stop by a Flextras Friday Lunch live session. More info at www.meetup.com/flextras
- Send us an e-mail at info@dot-com-it.com
- Send us an IM on Jabber/GTalk at flextras@gmail.com
- Find us on Twitter; the account name is Flextras.
- Give us a phone call at 203-379-0773.

For best results, please provide as much details as you can, including a sample application and steps to reproduce your problem.

We can also provide consulting services to develop custom extensions of our components, if our component doesn't explicitly fit your use case, but comes very close.

Thank You

We wanted to thank you for checking out the Flextras AutoCompeteComboBox. We can't wait to see what type of applications you're going to build with this. We're here to help so be sure to contact us if you have questions, enhancement requests, or if you just want to chat.

Jeffry Houser (The Brains Behind Flextras)

Jeffry@dot-com-it.com | reboog711@gmail.com | 203-379-0773