Creating a Widget

MiServer contains collections of classes which provide functionality which make JavaScript-based widgets easier to use for an APL developer. This guide explains how to add support for new widgets. From an implementors point of view, there are three fundamentally different classes of widgets

1) Widgets entirely based on standard HTML controls, like most of the "Dyalog" controls in the _DC namespace.
2) Widgets based on Javascript libraries which are already included with MiServer (JQueryUI and Syncfusion).
3) Widgets based on new Javascript libraries.

When the document is complete, it will provide step-by-step instructions for all of the above. At the moment, the only description provided is for the addition of new Syncfusion controls. Only a few Syncfusion widgets have complete wrappers and we hope to encourage users of MiServer to contribute!

Wrapping Syncfusion Controls

This document is based around recording work actually done to add support for the ejTextBoxes control. If a Syncfusion widget has been around any length of time, the MiServer team will almost certainly have generated a shell class for the widget, making it accessible to the APL developer. In the case of `ejTextBoxes`, the class looks like this:

```
:Class ejTextBoxes : #._SF._ejWidget

    :Field Public Shared Readonly DocBase←
               'http://help.syncfusion.com/UG/JS_CR/ejTextBoxes.html'
    :Field Public Shared Readonly ApiLevel←1
    :Field Public Shared Readonly DocDyalog←
               '/Documentation/DyalogAPIs/Syncfusion/ejTextBoxes.html'

    ∇ make
      :Access public
      JQueryFn←Uses←'ejTextBoxes'
      :Implements Constructor
    ∇

    ∇ make1 args
      :Access public
      JQueryFn←Uses←'ejTextBoxes'
      :Implements Constructor
    ∇
```

```
:EndClass
```

The base class `#._SF.ejWidget` provides common functionality for all Syncfusion widgets and should not be changed. The `DocBase` field documents the location of the Syncfusion documentation, and `DocDyalog` points to (currently non-existent) documentation of the widget itself. An `ApiLevel` of 1 is intended to inform developers that this is a completely basic widget with no added support for use from APL.

In fact, this class does not work as generated because it assumes that the base HTML object is a `<div>`, while the `ejTextBoxes` javascript function expects to run off an `<input>` element. To correct this, we need to add two lines of code to each of the constructors:

```
ContainerType←'input'
Container.type←'text'
```

The first statement sets the base HTML object type, and must appear before the `:Implements Constructor` statement.

```
    ∇ make
      :Access public
      JQueryFn←Uses←'ejTextBoxes'
      ContainerType←'input'
      :Implements Constructor
      Container.type←'text'
    ∇

    ∇ make1 args
      :Access public
      JQueryFn←Uses←'ejTextBoxes'
      ContainerType←'input'
      :Implements Constructor
      Container.type←'text'
    ∇
```

There are two almost identical constructors at this point, they will eventually separate so that we can properly support niladic construction and also provide arguments. We are now able to test our class fro immediate execution – assuming the `Start` function was run with a left argument of 1:

```
      p←⎕NEW MiPage ⍝ Create an instance of MiPage
      tb←'myTB' p.New _SF.ejTextBoxes ⍝ Add an instance of our class

      tb.Render
```

```
<input type="text" id="myTB" name="myTB"></input>
<script>$(function(){$("#myTB").ejTextBoxes({});});</script>
```

This illustrates the functionality exposed by the base class **`_SF.ejWidget`**, it has:

1) Created an HTML object of the requested type <input type=”text”>, with the specified id (and default name = id).
2) Added the Javascript call to the Syncfusion function specified by the JQueryFn property.
3) If the page were rendered in the context of a proper MiServer request, the “Uses” information would be used to determine Javascript dependencies, and ensure that the necessary Javascript libraries were also loaded. This is done by looking ip the ejTextBoxes name in the configuration file MiServer/Config/Resources.xml.

We can now create a MiPage to test our new widget. Syncfusion widgets should have at least one sample page created in the `MS3/Exampes/`SF folder, named `<widget name>Simple.dyalog`. A minimal sample might look like this:

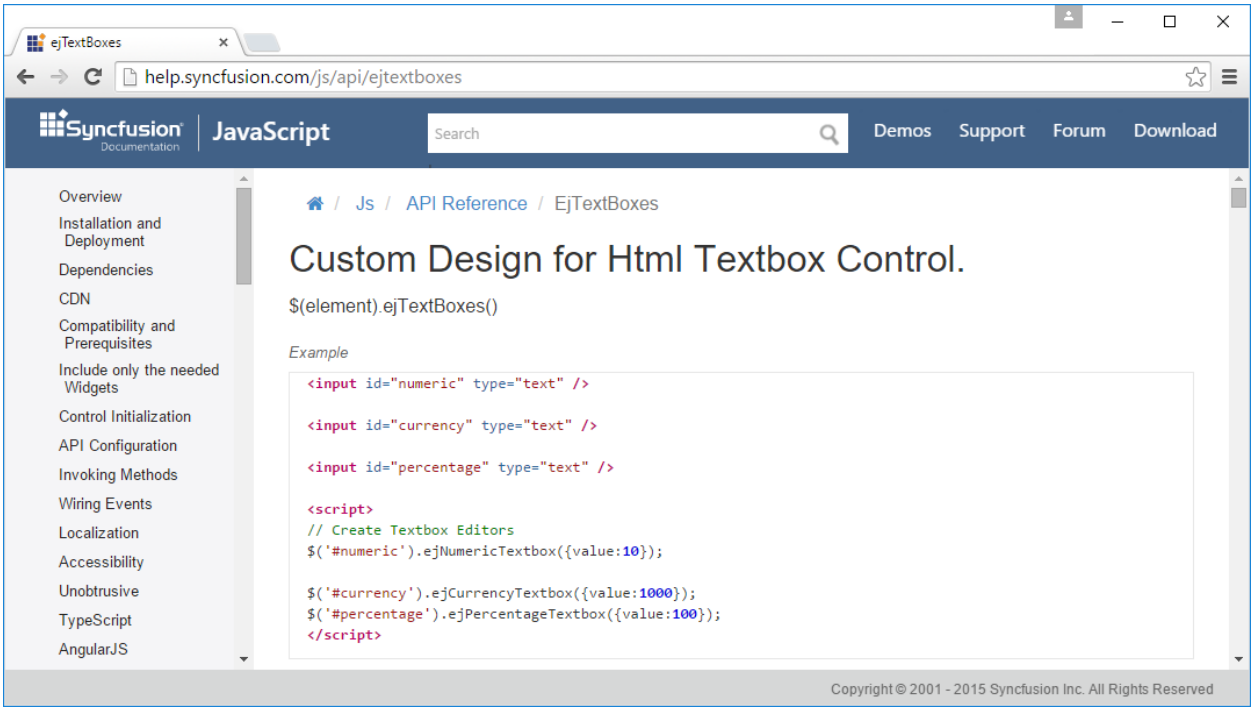
```
:Class ejTextBoxSimple : MiPageSample
⍝ Control:: _SF.ejTextBox
⍝ Description:: Validate numbers, currency and percentages

    ∇ Compose;tb1
      :Access Public

      tb1←'tb1'Add _.ejTextBoxes
    ∇

:EndClass
```

If we direct a web browser at this web page, an input field is indeed rendered, but “nothing works”. If we take a look at the Syncusion documentation for the widget, can see why:



There is no function called ejTextBoxes(), instead a different function is used for each of the variants (inconsistently names with lowercase “b” in box). We need to add the following code to our widget to set JQueryFn to the right value. First, add a public field to set the type, defaulting to numeric (Adding a setter and doing validation is left as an exercise for the reader):

```
:Field Public Type←'Numeric' ⍝ Numeric|Currency|Percentage
```

Second, we need to add a Render function, to set JQueryFN at the last moment before calling the base class render method:

```
∇ r←Render
  :Access Public

  JQueryFn←'ej',Type,'Textbox'
  r←⎕BASE.Render
∇
```

The reason for doing this at the last moment is to allow the user to modify the type after creating the instance. Our simple sample now works, displaying a single numeric `ejTextBox`.

Note that we can already make use of ***all*** the features of the Syncfusion widget at this point, by using the `Set` function to give values to the members documented by Syncfusion. For example, we could enhance our `Compose` function as follows:

```
∇ Compose;tb1
  :Access Public

  tb1←'tb1' Add _.ejTextBoxes
  'watermarkText' 'width'tb1.Set'Enter a 3-digit Number' 200
  'minValue' 'maxValue' tb1.Set 100 999
∇
```

This gives us a form with a watermark and a spinner which will go from 100 to 999.

Design - Public Fields and Constructor Arguments

So far, we have just been ironing out wrinkles in naming, to get to the point where the Syncfusion widget can be used. For most Syncfusion widgets, these would not even have been necessary, the automatically generated class would have been sufficient.

To complete the design, we need to read the Syncfusion documentation for the control and decide:

1) Whether there are other properties of the control which have a structure which means that it would be advantageous to expose as properties or fields of the APL class, rather than just setting them using the Set function.
2) What the constructor arguments should be – this is typically closely related to the first question.

In this case, we have already decided to expose the Type of the TextBox. None of the other properties really seem worth supporting, but we will add support for a “value” property. This group of Syncfusion

controls is behaving slightly badly in that they ignore the actual content of the underlying HTML control (`<input>`) – instead there is a `value` member that overrides this. By exposing a "value" property in our class, and having a constructor which takes type and value, we can increase the likelihood of correct use of our widget. The final code for our widget follows:

```
:Class ejTextBox : #._SF._ejWidget

    :Field Public Shared Readonly DocBase←… ⍝ As before
    :Field Public Shared Readonly ApiLevel←3
    :Field Public Shared Readonly DocDyalog←…

    :Field Public Type←'Numeric' ⍝ Numeric|Currency|Percentage
    :Field Public value←''

    ∇ make
      :Access Public
      Uses←'ejTextBoxes'
      ContainerType←'input'
      :Implements Constructor
      Container.type←'text'
    ∇

    ∇ make1 args
      :Access Public
      ⍝ args: Type [value]
      args←eis args ⍝ Enclose if simple
      (Type value)←args,(⍴args)↓'Numeric' ''
      Uses←'ejTextBoxes'
      ContainerType←'input'
      :Implements Constructor
      Container.type←'text'
    ∇

    ∇ r←Render
      :Access Public

      'Invalid Type set for ejTextBox'⎕SIGNAL
                   ((⊂Type)∊'Numeric' 'Currency' 'Percentage')↓11
      JQueryFn←'ej',Type,'Textbox'
      :If value≠'' ⋄ 'value'Set value ⋄ :EndIf
      r←⎕BASE.Render
    ∇

:EndClass
```

Renaming the Widget

In the above code, the widget was also renamed to the singular `ejTextBox`. The Syncfusion widget was described as “ejTextBoxes” in Syncfuction documentation, but turns out to be a collection of functions, none of which have that name. The singular form seems more natural.

Validation

Can’t get validation to work… For Brian to look at – try commenting out the line in the middle of `ejTextBoxSimple`.

Resources

Also explain why is is necessary to add to resources to get validation to work.

```
 <resource>
   <name>ejTextBoxes</name>
   <uses>Syncfusion</uses>
   <uses>jquery.globalize</uses>
   <uses>jquery.validate</uses>
   <uses>cultures</uses>
  </resource>
```

, we can see that the reason for this is that

Internal Events

We should also document how to set these…