White Paper

How to Develop GUI Controls

2017 June Release

Fabasoft®

Contents

1 Introduction

GUI Controls are used to define the presentation of values in the graphical user interface. Fabasoft provides several controls like a calendar control or newsfeed control by default. But you can also develop your own controls especially for your needs. This document helps you to get started with developing GUI controls.

A control logically consists of two parts: the server-sided part and the client-sided part. On the server side you define the object model and specify the data the control can access. On the client-side a JavaScript file is needed that specifies the appearance of the control and that allows accessing the data on the server side.

2 Requirements

You need a Fabasoft app.ducx development environment. Some experience in using Fabasoft app.ducx is assumed. More information about Fabasoft app.ducx can be found here: http://help.appducx.com/index.php?topic=doc/An-Introduction-to-Fabasoft-appducx/index.htm

3 Getting Started

All you need to create your first running control is contained in this chapter. You are going to create a control named `CustomIntegerControl` that customizes the presentation of an integer value. A minus and a plus symbol will be shown beside the value. When clicking on the symbols the integer value will be decreased or increased by one.

Create a new Fabasoft app.ducx project and define an object model file, a user interface file and a JavaScript resource file.

Example

app.ducx Object Model Language

```
objmodel FSCCONTROLSAMPLE@1.3285
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import COOATTREDIT@1.1;
  // a simple object class containing an integer property
  class CtrlSampleInteger : BasicObject {
    common = true;
    integer ctrlsampleinteger;
  }
  // creates an instance of a control
  instance ControljQuery CustomIntegerControl {
    // the control should be applicable for integer properties
    controltypes = { INTEGER }
    // provides the JavaScript implementation of the control
    lookelements<lookbasename, COOSYSTEM@1.1:component, lookcontent> = {
      { "rendercustominteger.js", FSCCONTROLSAMPLE@1.3285,
        file("resources/rendercustominteger.js") }
    }
  }
}
```

app.ducx User Interface Language

```
userinterface FSCCONTROLSAMPLE@1.3285
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
```

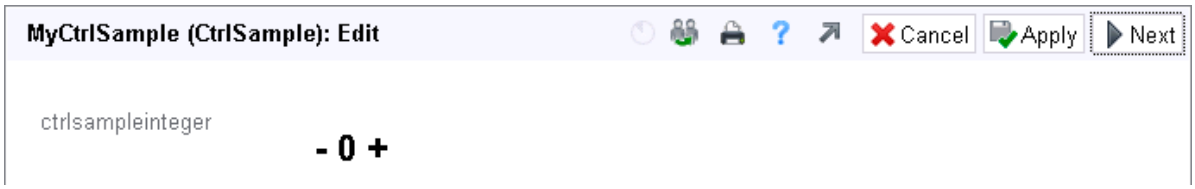
```
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  // when reading or editing the FormCtrlSampleInteger form should be used
  extend class CtrlSampleInteger {
    forms {
      ReadObjectAttributes {
        FormCtrlSampleInteger;
      }
      EditObjectAttributes {
        FormCtrlSampleInteger;
      }
    }
  }
  // a form displaying the integer property
  form FormCtrlSampleInteger {
   inherit = false;
   formgeneric = false;
    formpage PageCtrlSampleInteger {
      dataset {
        ctrlsampleinteger;
      }
      layout {
        row {
          // assigns the control to the integer property
          CustomIntegerControl ctrlsampleinteger {
            colspan = 4;
            labelposition = left;
          }
        }
      }
    }
  }
}
```

rendercustominteger.js

```
function jQueryFSCCONTROLSAMPLE_1_3285_CustomIntegerControl()
{
  // defines the HTML of the control
  this.OnRender = function CustomIntegerControl_OnRender(output)
  {
    // gets the current value and stores it in the arbitrary variable currvalue
    this.currvalue = parseInt(this.GetValue());
    if (isNaN(this.currvalue)) {
      this.currvalue = 0;
    }
    // if editable, provides buttons to change the value
    if (this.IsEdit()) {
      output.Push("<h2><a href=\"#\">-</a> <span id=\"" + this.GetId("IntValue") + "\">" +
        this.currvalue + "</span> <a href=\"#\">+</a></h2>");
    }
    else {
      output.Push("<h2>" + this.currvalue + "</h2>");
    }
  };
  this.OnLoad = function CustomIntegerControl_OnLoad()
  {
    this.valueelem = this.GetElement("IntValue");
    var ctrl = this;
    // uses jQuery to react on a click
    fscjq("#" + this.GetContainerId() + " A").click(function() {
      if (this.innerText == "-") {
        ctrl.currvalue--;
      }
      else {
        ctrl.currvalue++;
      }
      // displays the new value
      ctrl.valueelem.innerHTML = ctrl.currvalue;
```

```
      // sets the new value of the control; the parameter always has to be an array
      ctrl.SetValue([ctrl.currvalue]);
    });
  };
}
```

As a result you get a customized integer property:



Attention: If you change the JavaScript file in app.ducx, keep in mind to “Clean” the app.ducx project and to refresh the web browser such that the new JavaScript file is loaded.

4 Properties of the Object Class “jQuery Control”

Following properties of the object class *jQuery Control* (`FSCVAPP@1.1001:ControljQuery`) are commonly used. Properties that are not listed here are intended for internal use only. The examples are not self-contained, but they put focus on the important points.

4.1 FSCVAPP@1.1001:initexpr

`FSCVAPP@1.1001:initexpr` provides initial data for the control (e.g. strings). The result value has to be in JSON format.

Example

app.ducx Object Model Language

```
instance ControljQuery CustomObjectListControl {
  ...
  // provides initial data for the control
  initexpr = expression {
    string[] names;
    for (Object obj : ::value) {
      names += obj.GetName();
    }
    // the dictionary contains the desired values
    dictionary initvals = {
      strings: {
        RemoveEntry: #StrRemoveEntry.Print()
      },
      names: names
    };
    // convert the dictionary to a JSON string
    return coouser.Value2JSON(initvals);
  }
}
```

render.js

```
this.OnInit = function CustomObjectListControl_OnInit()
{
  // get the values defined in initexpr
  var init = this.GetInit();
  this.strings = init.strings;
  this.names = init.names;
  if (this.names) {
    if (typeof this.names == "string") {
      this.names = [this.names];
```

```
    }
  }
};

this.OnRender = function CustomObjectListControl_OnRender(output)
{
  if (this.names) {
    for (var idx = 0, cnt = this.names.length; idx < cnt; idx++) {
      ...
      output.Push(this.names[idx]);
      ...
      output.Push(this.strings.RemoveEntry);
    }
  }
}
```

4.2 FSCVAPP@1.1001:lookelements

`FSCVAPP@1.1001:lookelements` defines resources (JS, CSS and images) needed for the control. JavaScript and CSS files are automatically included in the HTML page.

Images can be accessed within the HTML code using following path:

`../tmp/x/<SWCREF>_<majorid>_<minorid>/<imagename>.<ext>`

In CSS files use the following path:

`../x/<SWCREF>_<majorid>_<minorid>/<imagename>.<ext>`

Example

app.ducx Object Model Language

```
instance ControljQuery CTRLEditImage {
  ...
  lookelements<lookbasename, COOSYSTEM@1.1:component, lookcontent> = {
    { "editimage.js", FSCIMAGE@1.1001, file("resources/js/editimage.js") },
    { "render.js", FSCIMAGE@1.1001, file("resources/js/render.js") },
    { "jquery.Jcrop.min.js", FSCIMAGE@1.1001, file("resources/js/jquery.Jcrop.min.js") },
    { "jquery.Jcrop.css", FSCIMAGE@1.1001, file("resources/css/jquery.Jcrop.css") },
    { "Jcrop.gif", FSCIMAGE@1.1001, file("resources/css/Jcrop.gif") }
  }
}
```

render.js

```
this.OnRender = function CTRLEditImage_OnRender(output)
{
  output.Push("<img src=\"../tmp/x/FSCIMAGE 1 1001/Jcrop.gif\" alt=\"\" />");
}
```

4.3 COOATTREDIT@1.1:controltypes

`COOATTREDIT@1.1:controltypes` can be used to document for which data types the control is applicable for (e.g. `INTEGER`, `OBJECT`, `OBJECTLIST`, `Content`, `WorkFlowParticipant`).

Example

app.ducx Object Model Language

```
instance ControljQuery CustomObjectListControl {
  controltypes = { OBJECTLIST, OBJECTLISTUNIQ }
  ...
}
```

5 Control Arguments

Control arguments can be used to influence the control behavior. They are defined in the app.ducx user interface language within brackets after the control.

Syntax

app.ducx User Interface Language

```
...
layout {
  row {
    CustomControl("arg1=value arg2=value ...") property {
      ...
    }
  }
}
```

`height` and `width` are predefined arguments, which can be accessed in the client JavaScript using the `GetHeight()` and `GetWidth()` functions. You can also define arbitrary control arguments, which can be accessed using the `GetOptions()` function.

Note: `GetOptions()` is also used for control options as described in the next chapter.

Example

app.ducx User Interface Language

```
...
layout {
  row {
    // provide three control arguments
    CustomIntegerControl("height=100 width=50 style='h3'") ctrlsampleinteger {
      colspan = 4;
      labelposition = left;
    }
  }
}
...
```

render.js

```
this.OnRender = function CustomObjectListControl_OnRender(output)
{
  // retrieve the control arguments
  var options = this.GetOptions();
  // print height and width; options.style is directly used as HTML tag
  output.Push("<" + options.style + ">" + this.GetHeight() + "</" + options.style + ">");
  output.Push("<div>" + this.GetWidth() + "</div>");
}
```

6 Control Options

Control options are evaluated in the `controloptions` expression, which is defined in the app.ducx user interface language in the corresponding property block.

Syntax

app.ducx User Interface Language

```
...
layout {
  row {
    CustomControl property {
```

```
      controloptions = expression {
        ...
      }
    }
  }
}
```

The result value has to be in JSON format. The control options can be accessed in the client JavaScript using the `GetOptions()` function.

Example

app.ducx User Interface Language

```
...
layout {
  row {
    CustomIntegerControl ctrlsampleinteger {
      colspan = 4;
      labelposition = left;
      // the values are available in the client JavaScript
      controloptions = expression {
        dictionary @options = {
          optstyle: 'h2',
          num: 42
        };
        return coouser.Value2JSON(@options);
      }
    }
  }
}
```

render.js

```
this.OnRender = function CustomObjectListControl_OnRender(output)
{
  // retrieve the control options
  var options = this.GetOptions();
  // print the defined options
  output.Push("<" + options.optstyle + ">" + options.num + "</" + options.optstyle + ">");
}
```

Note: Values of control arguments override values of control options with the same name. It is highly recommended to avoid this situation.

7 JavaScript Template for a Control

To provide the implementation of a GUI control following JavaScript functions can be used.

Syntax

```
function jQuery<SWCREF>_<majorid>_<minorid>_<ControlName>()
{
  this.OnInit = function <ControlName>_OnInit()
  {
  };
  this.OnRender = function <ControlName>_OnRender(output)
  {
  };
  this.OnLoad = function <ControlName>_OnLoad()
  {
  };
  this.OnLayout = function <ControlName>_OnLayout(isinit)
  {
```

```
  };
  this.OnSubmit = function <ControlName>_OnSubmit(isinit)
  {
  };
  this.OnCleanup = function <ControlName>_OnCleanup()
  {
  };
  this.MyFunction = function <ControlName>_MyFunction()
  {
  };
}
```

7.1 Naming Scheme

Define a function with following naming scheme:
`jQuery<SWCREF>_<majorid>_<minorid>_<ControlName>()`

- `SWCREF`
  The reference of the software component that provides the control. The reference is set when creating an app.ducx project.
- `majorid`
  The major domain ID. The major domain ID is set when creating an app.ducx project.
- `minorid`
  The minor domain ID. The minor domain ID is set when creating an app.ducx project.
- `ControlName`
  An arbitrary name for the control.

Example: `jQueryFSCCONTROLSAMPLE_1_3285_CustomIntegerControl`

7.2 Predefined Functions

Within the function described above you can provide implementations for `OnInit`, `OnRender`, `OnLoad`, `OnLayout`, `OnCleanup` and your own functions.

- `OnInit`
  This function is called after the control object has been created.
- `OnRender`
  This function is called when rendering the HTML page. The parameter `output` allows printing HTML tags.
- `OnLoad`
  This function is called after the control HTML is written on the HTML page.
- `OnLayout`
  This function is called after `OnLoad` is carried out and for each resize of the web browser window. The optional parameter `isinit` is `true` when `OnLayout` is called the first time. In case of a window resize the parameter is `false`.
- `OnSubmit`
  This function is called before the page is submitted. It is a good point to call `this.SetValue()` if you did not set the value during runtime.
- `OnCleanup`
  This function is called when the dialog that contains the control is closed.
- `GetTabTarget`
  This function is called after a form was rendered. It has to return an element which should be focused.
  Default return value is `this.Container()`.

- Your own functions
  You can specify your own helper functions.

7.3 Predefined Objects

Within the JavaScript implementation following two objects are available:

- `output`
  Provides the function `Push` that can be used to write HTML tags.
  Example: `output.Push("<h2>" + this.currvalue + "</h2>");`
- `fscjq`
  Allows to access the jQuery provided by the Fabasoft product. `fscjq` has to be used instead of the `$` function.
  Example: `fscjq("#" + this.GetContainerId() + " A")`

7.4 Predefined Attributes

Within the JavaScript implementation following attributes are available:

- `this.info`
  Provides additional information for the instance of the control.
  - `label`
    A string containing the label of the form field.
  - `labelid`
    If the label for the form field is visible: a string containing the ID of the DOM element of the label.
  - `mustbedef`
    Boolean, `true` if the form field has to be defined.
- `this.config`
  - `ariarole`
    Default: `"document"`, the role is used for the attribute `aria-role` on the DOM element containing the control.

7.5 Customization options

Within the JavaScript implementation following options can be set on the parent control:

- `this.parent.isfullsize = true`
  In an overlay the control will be rendered fullsize (without any padding or margin)

8 JavaScript Functions

An overview of all JavaScript functions that can be used in the context of GUI controls can be found here: http://help.appducx.com/index.php?topic=doc/GUI-Controls-API/index.htm.

Attention: JavaScript functions not listed on that page are only for internal use.

In this chapter you can find examples for some of these functions.

8.1 Five Functions at a Stroke

In the following example you are going to create a control that allows calling an application. It demonstrates the usage of `GetId`, `GetObject`, `GetUrl`, `GetElement`, `StartOverlayApp` and a user-defined helper function.

Example

app.ducx Object Model Language

```
objmodel FSCCONTROLSAMPLE@1.3285
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import COOATTREDIT@1.1;

  fields {
    Object obj;
    string url;
  }

  class CtrlSampleString : BasicObject {
    common = true;
    string startapp;
  }

  // creates an instance of a control
  instance ControljQuery CustomStringControl {
    // the control should be applicable for integer properties
    controltypes = { STRING }
    // provides the JavaScript implementation of the control
    lookelements<lookbasename, COOSYSTEM@1.1:component, lookcontent> = {
      { "rendercustomstring.js", FSCCONTROLSAMPLE@1.3285,
        file("resources/rendercustomstring.js") }
    }
  }
}
```

app.ducx User Interface Language

```
userinterface FSCCONTROLSAMPLE@1.3285
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  import FSCEXPEXT@1.1001;

  // when reading or editing the FormCtrlSample form should be used
  extend class CtrlSampleString {
    forms {
      ReadObjectAttributes {
        FormCtrlSampleString;
      }
      EditObjectAttributes {
        FormCtrlSampleString;
      }
    }
  }

  // a form displaying the integer property
  form FormCtrlSampleString {
   inherit = false;
   formgeneric = false;
    formpage PageCtrlSampleString {
      dataset {
        startapp;
      }
      layout {
        row {
          CustomStringControl startapp {
            colspan = 4;
            labelposition = left;
```

```
          }
        }
      }
    }
  }
  // a form that is used for the called application
  form CustomDataForm {
    audience = enduser;
    formpage CustomDataPage {
      audience = enduser;
      dataset {
        field obj {
          get = expression {
            containerobject;
          }
        }
        field url {
          get = expression {
            ctrlurl
          }
        }
      }
      layout {
        row {
          obj {
            colspan = 4;
            labelposition = left;
          }
        }
        row {
          url {
            colspan = 4;
            labelposition = left;
          }
        }
      }
    }
  }
}
```

app.ducx Use Case Language

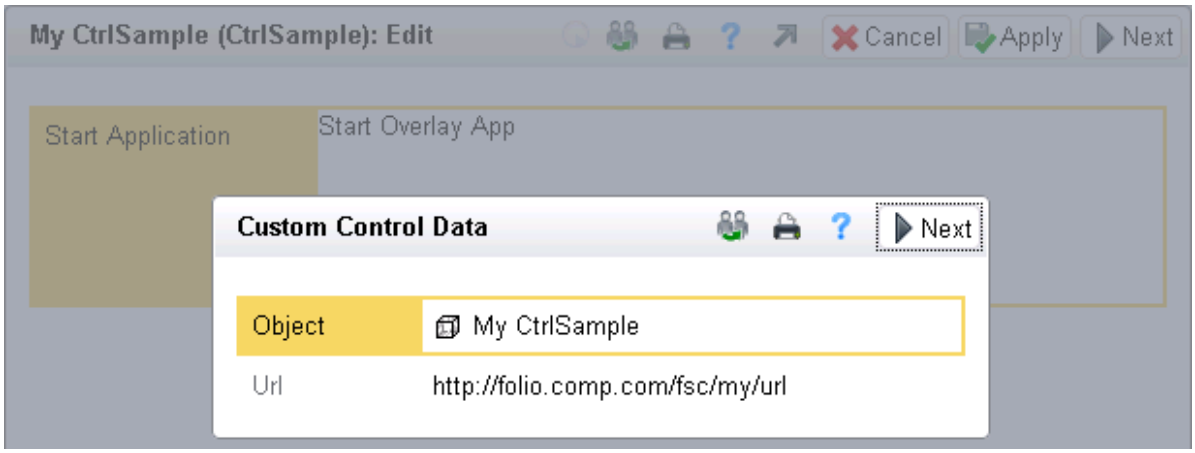
```
usecases FSCCONTROLSAMPLE@1.3285
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;
  import FSCVIEW@1.1001;
  // call the application
  usecase ShowCustomControlData(Object containerobject, string ctrlurl) {
    variant Object {
      impl = application {
        expression {
          ->CustomControlDataDialog;
        }
        dialog CustomControlDataDialog {
          form = CustomDataForm;
          target = coouser;
          edittype = EDITTYPE VIEW;
          overlaysize = OVERLAYSIZE SMALL;
          nextbranch;
        }
      }
    }
  }
}
```

rendercustomstring.js

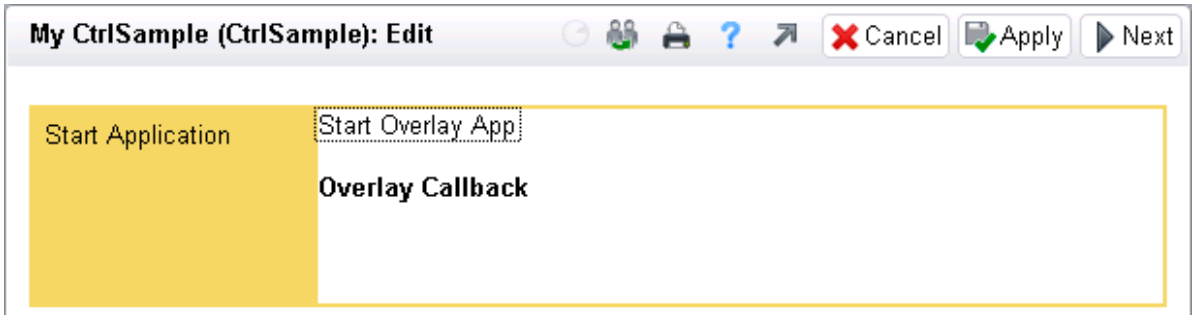
```
function jQueryFSCCONTROLSAMPLE_1_3285_CustomStringControl()
{
```

```
  this.OnRender = function CustomStringControl_OnRender(output)
  {
    // use GetId to get an ID that is unique on the whole page
    output.Push("<a id=\"" + this.GetId("StartOverlayApp") + "\" href=\"#\">Start App</a>");
    output.Push("<div id=\"" + this.GetId("MessageBox") + "\"></div>");
  };
  this.OnLoad = function CustomStringControl_OnLoad()
  {
    var ctrl = this;
    var args = {
      // the COO address of the current object
      containerobject: this.GetObject(),
      // concatenates the web service URL and the given URL part
      ctrlurl: this.GetUrl("/my/url")
    };
    fscjq("#" + this.GetId("StartOverlayApp")).click(function() {
      // start the overlay application
      // you have to replace "COO.1.3285.3.10" with your application's address
      ctrl.StartOverlayApp("COO.1.3285.3.10", args, ctrl.OverlayAppCallback);
    });
  };
  // this function gets called when closing the overlay
  this.OverlayAppCallback = function CustomTestControl_OverlayAppCallback()
  {
    // get the element defined by GetId("MessageBox")
    this.GetElement("MessageBox").innerHTML += "<br/><b>Overlay Callback</b>";
  };
}
```

You can click “Start Overlay App”. The overlay shows the passed parameters.



After clicking “Next” the callback function modifies the HTML and prints “Overlay Callback”.



8.2 Fun With Object Lists

In the following example you are going to create your own object list. It demonstrates the usage of `GetInit`, `GetId`, `GetArgs`, `GetSymbolPath`, `GetValue`, `SetValue` and `Refresh`.

Example

app.ducx Object Model Language

```
objmodel FSCCONTROLSAMPLE@1.3285
{
  import COOSYSTEM@1.1;
  import FSCVAPP@1.1001;
  import COOATTREDIT@1.1;
  import FSCVENV@1.1001;
  import FSCEXPEXT@1.1001;

  class CtrlSampleObjectList : BasicObject {
    common = true;
    Object sourceobject;
    Object[] objectlist;
  }

  // creates an instance of a control
  instance ControljQuery CustomObjectListControl {
    // the control should be applicable for object list properties
    controltypes = { OBJECTLIST }
    // provides the JavaScript implementation of the control
    lookelements<lookbasename, COOSYSTEM@1.1:component, lookcontent> = {
      { "rendercustomobjectlist.js", FSCCONTROLSAMPLE@1.3285,
        file("resources/rendercustomobjectlist.js") }
    }
    // provides initial data for the control
    initexpr = expression {
      string[] names;
      // the names of the objects in the object list
      for (Object obj : ::value) {
        names += obj.GetName();
      }
      // the dictionary contains a string and the object names
      dictionary initvals = {
        strings: {
          RemoveEntry: #StrRemoveEntry.Print()
        },
        names: names
      };
      // convert the dictionary to a JSON string
      return coouser.Value2JSON(initvals);
    }
  }
}
```

app.ducx User Interface Language

```
userinterface FSCCONTROLSAMPLE@1.3285
{
  import COOSYSTEM@1.1;
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import FSCVAPP@1.1001;
  import FSCEXPEXT@1.1001;

  // when reading or editing the FormCtrlSample form should be used
  extend class CtrlSampleObjectList {
    forms {
      ReadObjectAttributes {
        FormCtrlSampleObjectList;
      }
      EditObjectAttributes {
        FormCtrlSampleObjectList;
      }
```

```
    }
  }

  // a form displaying the integer property
  form FormCtrlSampleObjectList {
   inherit = false;
   formgeneric = false;
    formpage PageCtrlSampleObjectList {
      dataset {
        sourceobject;
        objectlist;
      }
      layout {
        row {
          sourceobject {
            colspan = 4;
            labelposition = left;
            // puts the selected object in the object list
            change = expression {
              if (null != ::value) {
              Object[] @values = cooobj.GetAttribute(cootx,
                                  #FSCCONTROLSAMPLE@1.3285:objectlist);
              @values += ::value;
              cooobj.SetAttribute(cootx, #FSCCONTROLSAMPLE@1.3285:objectlist, @values);
              }
            }
          }
        }
        row {
          CustomObjectListControl objectlist {
            rowspan = 2;
            colspan = 4;
            labelposition = top;
          }
        }
        row {
        }
      }
    }
  }
}
```

rendercustomobjectlist.js

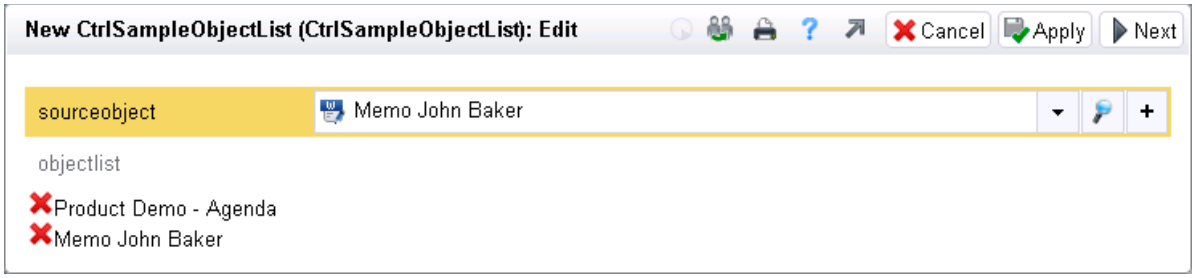
```
function jQueryFSCCONTROLSAMPLE_1_3285_CustomObjectListControl()
{
  this.OnInit = function CustomObjectListControl_OnInit()
  {
    // get the values defined in initexpr
    var init = this.GetInit();
    this.strings = init.strings;
    this.names = init.names;
    if (this.names) {
    if (typeof this.names == "string") {
      this.names = [this.names];
      }
    }
  }

  this.OnRender = function CustomObjectListControl_OnRender(output)
  {
    if(this.names){
      var values = this.GetValue();
      output.Push("<div id=\"" + this.GetId("CustomObjectList") + "\">");
      // prints all objects in the object list
      for (var idx = 0, cnt = values.length; idx < cnt; idx++) {
        output.Push("<div " + this.GetArgs(values[idx]) + ">");
        // a button to be able to remove the entry
        output.Push("<img class=\"remove\" src=\"");
        output.Push(this.GetSymbolPath("COO.1.1001.1.38424"));
        output.Push("\" alt=\"" + this.strings.RemoveEntry + "\" data-i=\"" + idx +"\" />");
        output.Push(this.names[idx] + "</div>");
```

```
      }
      output.Push("</div>");
    }
  };

  this.OnLoad = function CustomObjectListControl_OnLoad()
  {
    var ctrl = this;
    // a remove button is clicked
    fscjq(".remove").click(function() {
      var idx = this.getAttribute('data-i');
      var values = ctrl.GetValue();
      // remove the entry from the array
      values.splice(parseInt(idx), 1);
      // store the new array
      ctrl.SetValue(values);
      // refresh the list
      ctrl.Refresh();
    });
  };
}
```

You can add objects to the customized object list using “sourceobject”. Removing object from the list can be done with the “Remove” button.



9 jQuery - Everywhere?

In GUI controls you can use jQuery (`fscjq`) and benefit from the library but there is no necessity to use it. Good old plain JavaScript does the job, too.

Another case is that you already have a standard jQuery plug-in. Since the Fabasoft product provides `fscjq` instead of `$` it is a bit tricky. To be able to use the jQuery plug-in functions with `fscjq` you have to provide a `prepare.js` and a `cleanup.js` file.

Example

app.ducx Object Model Language

```
instance ControljQuery CustomCalControl {
  ...
  lookelements<lookbasename, COOSYSTEM@1.1:component, lookcontent> = {
    { "prepare.js", FSCCONTROLSAMPLE@1.3285, file("resources/js/prepare.js") },
    { "jquery.plugin.js", FSCCONTROLSAMPLE@1.3285, file("resources/js/jquery.plugin.js") },
    { "render.js", FSCCONTROLSAMPLE@1.3285, file("resources/js/render.js") },
    { "cleanup.js", FSCCONTROLSAMPLE@1.3285, file("resources/js/cleanup.js") }
  }
}
```

prepare.js

```
if ($ && $.noConflict) {
  window._$ = $.noConflict(true);
}
if (jQuery && jQuery.noConflict) {
  window._jQuery = jQuery.noConflict(true);
}
```

```
$ = jQuery = fscjq;
```

cleanup.js

```
fscjq = jQuery.noConflict(true);
if (window._$) {
  window.$ = _$.noConflict(true);
  delete window._$;
}
if (window._jQuery) {
  window.jQuery = _jQuery.noConflict(true);
  delete window._jQuery;
}
```

10 Accessibility

In order to make your control available for assistive technologies and Fabasoft app.test you need to implement the correct semantic and keyboard navigation.

The Fabasoft product is implemented as a WAI-ARIA application (http://www.w3.org/WAI/intro/aria.php).

The default role for a new jQuery Control is `document` (see chapter 7.4 “Predefined Attributes”) and the focused element is `this.Container()` (see chapter 7.2 “Predefined Functions”).

If you want more than just providing information as plain HTML and standard form elements, you need to implement your control as WAI-ARIA widget (http://www.w3.org/TR/wai-aria-practices/#accessiblewidget).

In the following example the control renders a custom checkbox:

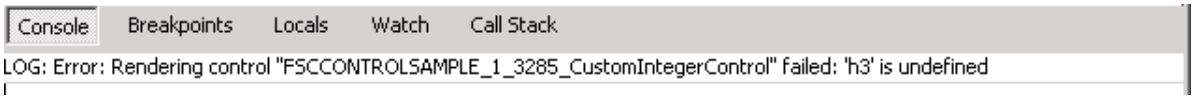
Example

```
function jQuery<SWCREF>_<majorid>_<minorid>_MyCheckbox()
{
  // do not set ariarole on container (override default behavior)
  this.config.ariarole = "";
  this.OnRender = function MyCheckbox_OnRender(output)
  {
    // add a unique id
    output.Push("<div id=\"" + this.GetId("MyCheckbox") + "\" " +
    // add a tabstop to the element which is returned by this.GetTabTarget()
                "tabindex=\"0\" " +
    // add the correct role (http://www.w3.org/TR/wai-aria/roles#checkbox)
                "role=\"checkbox\" " +
    // add the required attributes and states
                "aria-label=\"" + this.info.label + "\" " +
                "aria-checked=\"" + this.value + "\">");
    output.Push("</div>");
  };
  this.GetTabTarget = function MyCheckbox_GetTabTarget ()
  {
    // return the element with role="checkbox"
    return this.GetElement("MyCheckbox");
  };
  ...
}
```

11 Debugging

11.1 JavaScript Errors

JavaScript errors are caught to avoid affecting the whole client. The errors get logged in the web browser console.



By default the JavaScript source code of a control is deployed in one minified file. You can use the URL parameter `&cd=true` to include the JavaScript source code as multiple, not minified files.

Example:

`http://fo.comp.at/fsc/fscasp/content/bin/fscvext.dll?ax=COO.1.1001.1.32498&cd=true`

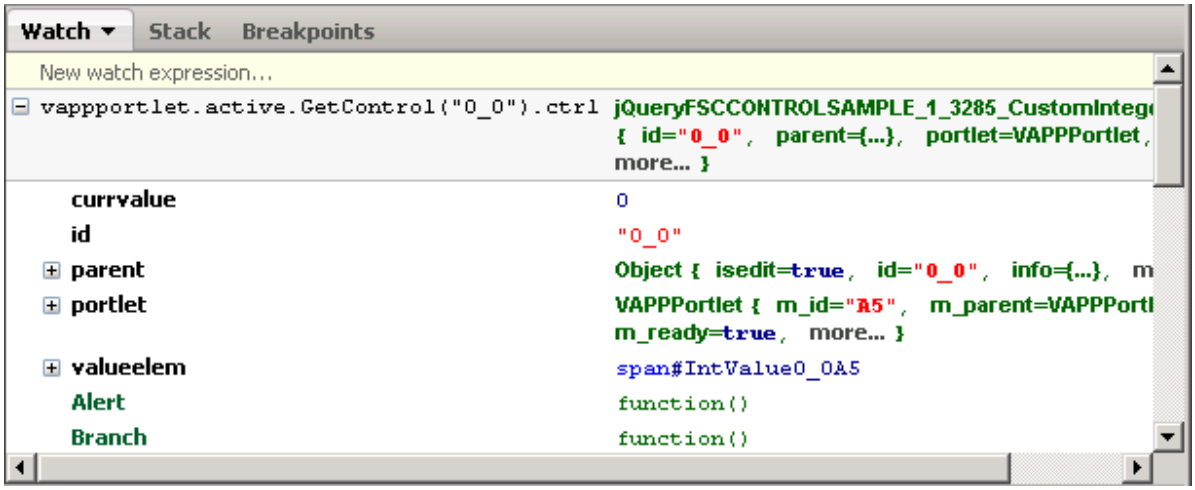
11.2 Inspecting the Control

In some situations it may be useful to be able to inspect the control in the web browser. In this case Mozilla Firefox with the Firebug extension is used for easy debugging. Inspect the HTML of the control and look for the `data-ctrl` attribute. In this case the value is “0_0”.

```
<tr class="FscGrid">
    <td id="grl_ctrlsampleinteger_A5" class="FscGridLabel" rowspan="1" colspan="1" tabindex="0">
    <td id="grd_ctrlsampleinteger_A5" class="FscGridData" data-ctrl="0_0" rowspan="1" colspan="3">
        <div id="FscCustomCtrl0_0A5" class="FscCustomCtrl" style="height: 319px;">
            <h2>
                <a href="#">-</a>
                <span id="IntValue0_0A5">0</span>
                <a href="#">+</a>
```

Now you can define following watch expression based on the `data-ctrl` attribute:

`vappportlet.active.GetControl("0_0").ctrl`



12 Troubleshooting

Problem:

- I have changed the JavaScript file, but after running the app.ducx project the changes are not reflected in the Fabasoft product.

Possible solution:

- Changed resource files are not uploaded automatically. Before running the app.ducx project execute “Project” > “Clean”.
- JavaScript files are only loaded when refreshing the page. Refresh the page (`F5`).

Problem:

- The control is not displayed.

Possible solution:

- JavaScript errors are caught to avoid affecting the whole client. The errors get logged in the web browser console.

Problem:

- `Value2JSON` cannot be resolved.

Possible solution:

- The software component `FSCEXPEXT@1.1001` provides the action `Value2JSON`. Make sure to import `FSCEXPEXT@1.1001`.