How to Embed Flash Components in Visual Composer 7.0

Applies to:

Visual Composer for SAP enhancement package 2 for SAP NetWeaver 2004s Visual Composer 7.0.

For more information, visit the User Interface Technology homepage.

Summary

Step by Step guide for creating Flash Components to embed in Visual Composer.

Author: Amir Mimran

Company: SAP

Created on: December 13, 2009

Author Bio

Amir Mimran is a software developer in the Visual Composer development group at SAP Israel, and holds a B.Sc. in Computer Science from Academic College of Tel Aviv.

.

Table of Contents

Overview

Visual Composer 7.0, while providing means to rapidly model application UIs, is limited in the amount of UI controls it offers the application modeler. While the current set of UI controls and components covers the main business application needs (Tables, Charts, etc.), different customers will often require specific controls which are not available out of the box.

These **custom** or **3rd party** controls can include variations on existing controls (such as different styled gauges) or controls which are missing in Visual Composer altogether (e.g. "World Map").

This feature enables incorporating custom or 3rd party ***Adobe Flex*** components ("Components") within Visual Composer models, and opens the Visual Composer environment to customer-specific UI requirements.

This how-to guide will explain how to create and customize Flash components so they can be used inside the Visual composer environment.

How to?

1. General

A Flash component is imported as a ZIP archive that includes the following three folders:

*"**APP_FILES**"* – includes the Flex Application file (SWF) and it's used resources (i.e. images, multimedia files, other flash applications), in the required folder structure, this is to ensure that internal links to resources inside the existing Flex application will function as planned.

(See article 2 below)

"METADATA" – includes a ***VCXL*** file (must be named "***metadata.vcxl***"), which describes the interface with the flash component (see the referenced "*VCXL Specification*" document)

(See article 3 below)

*"**IMAGES**"* – includes three design time image files: 16x16 and 32x32 pixels icons and a preview image for the layout pane.

(See article 4 below)

2. Creating the Flash Component application file

For the main application file of the component you can use a Flash application that you have developed by yourself, or use a component from the variety of controls that exists in Xcelsius, that you have compiled to an SWF file.

2.1. A Flash component based on a custom Flash application.

Upon creating your new custom flash controls, you need to customize it to communicate with the Visual composer runtime. To accomplish that, you need to define some input and output methods in your component's code.

The type of variable the Visual composer runtime sends and expects to receive from those functions, is a native Flash array.

To make it clearer, let's assume that the data is the following dataset:

BANK_NAME	*BANK_KEY*	*BANK_CTRY*
National Bank of Neverland	*123456*	*Neverland*
Acme Bank	*234567*	*Disneyland*
…	*…*	*…*

Each row in the dataset is represented by an Object contained in each of the Array's cells:

inArr[0]	*{ BANK_NAME:" National Bank of Neverland", BANK_KEY:"123456", BANK_CTRY:"Neverland" }*
inArr[1]	*{BANK_NAME:" Acme Bank", BANK_KEY:" 234567", BANK_CTRY:" Disneyland" }*
inArr[2]	..
inArr[3]	..
inArr[4]	..

- *Input functions* – for data coming *from* the Visual Composer runtime *into* your custom component, should be defined to receive an input parameter of type ***Array***, as defined above.
  For **example**:

```
public function myInputFunction(inArr:Array) {
   // the following line gets the bank's key value from the
   // dataset's second row.
   var bankKey:String = inArr[2]["BANK_KEY"] as String;
}
```

Note:
- This function must be defined as ***public***
- in the metadata XML which describes the component (see article 3) – The matching input port name will be equal to your input function name.

- Output functions – for data you wish to send from your component to the Visual Composer runtime.
  In order to pass data outside your component, you should dispatch an event.
  You should create an ActionScript class which:

1. Extends the flash.events.Event class.
2. Includes a property called "data" which sets and gets an Array that its structure is as explained above.

Example for such an event**:**

```
package src {
  import flash.events.Event;

  public class OutEvent extends Event
  {
        public function OutEvent(type:String,
                                 bubbles:Boolean=false,
                                 cancelable:Boolean=false)
        {
              super(type, bubbles, cancelable);
        }

        private var mArray:Array = []

        public function set data(value:Array):void {

              mArray = value;
        }
        public function get data():Array {
              return mArray;
```

```
        }
    }
}
```

Example for a function that sends output data:

```
// The following function sends two Out bounding parameters:
// "myString" and "myNumber"
// inside a three-rowed dataset

private function sendOut():void {
   var arr:Array = [{myString:"String 1", myNumber:1},
                    {myString:"String 2", myNumber:2},
                    {myString:"String 3", myNumber:3}
                   ];
   var evt:OutEvent = new OutEvent("myDataChanged");
   evt.data = arr;
   dispatchEvent(evt);
}
```

Note:

- In the metadata XML which describes the component (see article 3) – The matching output port name should be equal to the event *type* you are sending, (e.g. "myDataChanged" in the above example, and **not** "sendOut" or "OutEvent".)
- You can use the same Event Class (e.g. OutEvent) for several out bounding ports - By passing different event types to the event constructor.

- *Configuration Data* (i.e. "Properties")

  In terms of configuration data, your application will receive the configuration parameters in its query string, as key-value pairs.

  Therefore all you need to do is to fetch them in the initialization step of your application.

  For example, if your configuration data contains three alert colors for a gauge control:

```
<mx:Application applicationComplete="init() … />
.
.
.
// The following function retrieves three range colors for a
// gauge control.

private function init():void {
   var paramObj:Object = this.parameters;
   var c1:String = paramObj["range1Color"] as String;
   var c2:String = paramObj["range2Color"] as String;
   var c3:String = paramObj["range3Color"] as String;
   theGauge.setStyle("alertColors",[c1, c2 ,c3]);
}
```

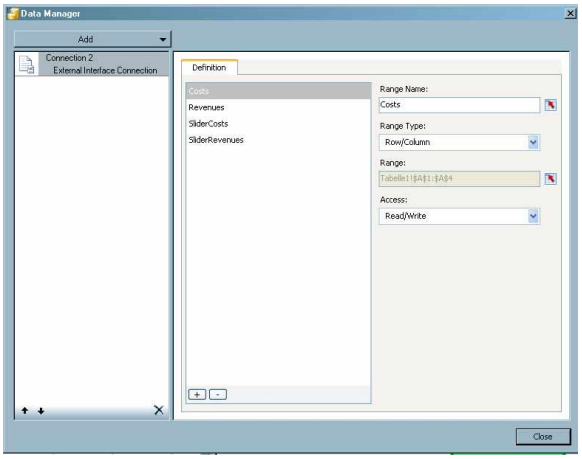
2.2. A Flash component based on an Xcelsius application.

Xcelsius has the ability to create an interface, where you can define which values could be read and/or written by external applications (the Visual Composer's Flash runtime in our case), into the internal excel sheet that acts as the xcelsius application's data source.

This interface is called "Data Connection", and following is the way to use it, and customize it so it will work with the Visual Composer Flash runtime.

1) Press the "Manage Connections"- Button  of the XCelsius tool bar.

2) The Data Manager dialog will open:



3) In the "Data Manager" popup choose Add -> External Interface Connection

4) Type in the following information:

Range Name: The Range Name will be the relevant mapping name for your Flash Component.

Range Type: see 3.2 for more information regarding this property.

Range: Choose the area in the embedded excel sheet where your data exist, and should be read from or written to.

Access: Choose Read/Write in case you want to change values in both directions (Component → Runtime and Runtime → Component)

- If you wish to use configuration data for your component, i.e. properties that will be set before any other action on the component (such as colour, titles etc…)

  You should define a range with its name set to "**config**".

  Its range type could be Row/column (for multiple configuration properties) or "Cell" (for one property)

3. Create the Metadata XML (VCXL)

Now you need to create an XML descriptor for your component, this XML describes the external interface of your component in terms of its attributes (such as id, name, version etc…) along with the data ports and component configuration.

For more details on how to write the VCXL file review the referenced "*VCXL Specification*" document.

Following are type-specific comments.

3.1. A Flash component based on a custom Flash application

- The *type* attribute in the VCXL root tag (vcx:Component) should be set to "**flex2**".

  For example:

```
<vcx:Component id="…." name="…" type="flex2" … />
```

- The name of *inbound* ports should be equal to the matching input functions in your custom code.
- The name of *outbound* ports should be equal to the matching event types that your code dispatches.

  E.g. for the banks example above, the Data section of the VCXL will look as following:

```
<vcx:Data>
        <vcx:Inports>
              <vcx:ArrayPort name="myInputFunction">
                    <vcx:string name = "BANK_NAME" />
                    <vcx:string name = "BANK_KEY" />
                    <vcx:string name = "BANK_CTRY" />
              </vcx:ArrayPort>
        </vcx:Inports>
        <vcx:Outports>
              <vcx:ArrayPort name="myDataChanged">
                    <vcx:string name = "myString" />
                    <vcx:float name = "myNumber" />
              </vcx:ArrayPort>
        </vcx:Outports>
</vcx:Data>
```

3.2. A Flash component based on an XCelsius application.

- The *type* attribute in the VCXL root tag (vcx:Component) should be set to "**xcelsius**".
- The names of your ports are equal to the Xcelsius's Data-connection names you've defined.
- The configuration parameters should be defined according to the "**config**" range as defined in the Xcelsius application, from the left to right.
- The *name* attribute of fields don't have to correlate to the excel coordinates (e.g. "A23"), it recommended that they will have a user-friendly name, as they are used in the Visual composer design-time.
- You should define your fields in order – from left to right.
- ***Which port-type should I map for each Xcelsius range type***?

  The VCXL port types (*vcx:SingletonPort* and *vcx:ArrayPort)*, are mapped to the Xcelsius range types (*cell, row/column, table*), as follows:

  1) Simple value

Expected Values:	123, "Acme Bank"
Excel range for example :	A0 (one cell)
Xcelsius range type:	cell.

```
<vcx:SingletonPort name="aPort">
      <vcx:[type] name="[CELL NAME]">
</vcx:SingletonPort>
```

2) Simple (1D) array

Expected Value: [1, 2, 3]

Excel range example: A0:Z0 (row) or A0:A500 (column)

Xcelsius range type: Row/Column

In this case, the runtime/component sends one data-row.

There are two choices in that case:

a) An “array” that each of its cells contains one value

```
<vcx:ArrayPort name="aPort">
            <vcx:[type] name = "xyz" />  <== Only one.
</vcx:ArrayPort>
```

b) An “object” of several fields

```
<vcx:SingletonPort name="aPort">
      <vcx:[type] name="A">
      <vcx:[type] name="B">
        ...
        ...
</vcx:SingletonPort>
```

E.g. for this range:

	A	B	C	D
1	1	2	3	4
2	5	6	7	8

For the 2nd row:

```
// an object of 4 fields.
// data = {A:5, B:6, C:7, D:8}
<vcx:SingletonPort name=”..”>
      <vcx:[type] name="A">
      <vcx:[type] name="B">
      <vcx:[type] name="C">
      <vcx:[type] name="D">
</vcx:SingletonPort>

But also:
// an array where each cell contains one value
// data = [5,6,7,8])

<vcx:ArrayPort name=”…”>
      <vcx:int name=”Second_Row” />
</vcx:ArrayPort>
```

For the B column:

```
// an array where each cell contains one value
// data = [2,6]
<vcx:ArrayPort name="…">
      <vcx:int name="B" />
</vcx:ArrayPort>)

But also:
// an object of 2 fields.
// data = {one1:2, two2:6}
<vcx:SingletonPort name="..">
      <vcx:[type] name="one1">
      <vcx:[type] name="two2">
</vcx:SingletonPort>
```

Note: The quantity of cells in the array is limited by the quantity of excel cells you defined as the range.

3) 2D Array

Example: [[1, 2, 3], [4, 5, 6]])

Excel range for example: A0:Z100 (more than one excel rows)

Xcelsius range type: Table

In this case the runtime/component sends a dataset of more than one row.

The <vcx:ArrayPort /> you define, actually defines one row of a dataset.

```
<vcx:ArrayPort name="aPort">
      <vcx:[type] name="A">
      <vcx:[type] name="B">
      <vcx:[type] name="C">
</vcx:ArrayPort>
```

E.g. for this range:

	A	B	C	D
1	1	2	3	4
2	5	6	7	8

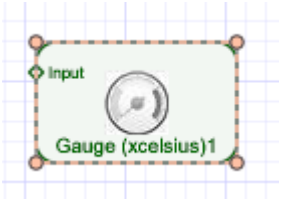
For a "table" range the VCXL definition will be as follows:

```
<vcx:ArrayPort name="aPort">
      <vcx:int name="A">
      <vcx:int name="B">
      <vcx:int name="C">
      <vcx:int name="D">
</vcx:ArrayPort>
```

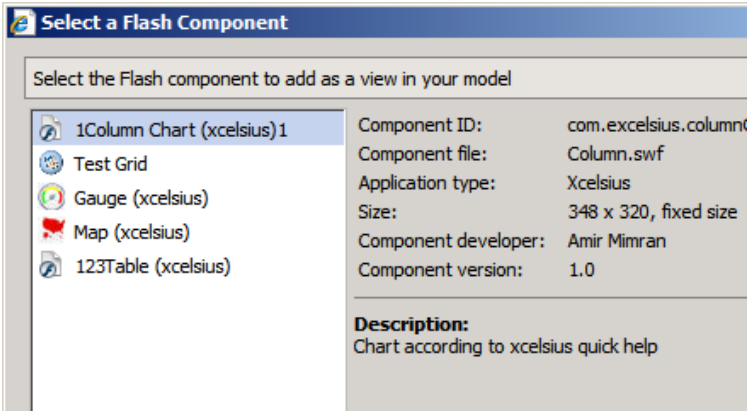
4. Add Design-time images

In order to uniquely identify your component in the Visual Composer design time, you may provide the following three images:

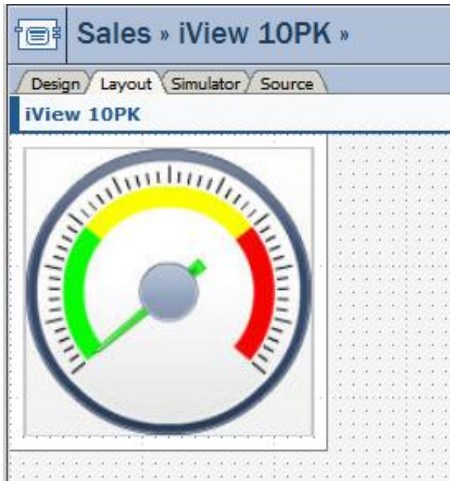
- A 32x32 icon file that will be used as the entity icon on the design board



- A 16x16 icon file that will be used in the “Flash Component Manager” and “Select a Flash Component” dialogs.



- A preview image, for display in the layout board.



Note: if your component is fixed-size it’s recommended that this image size will be identical to the fixed-size dimensions you provide in the VCXL Component size attribute.

- If you will not provide one of these image files, a default icon will appear

The 16x16 icon: 

The 32x32 icon and preview image (centered): 

5. Pack the component in a Zip Archive

- After completing the above three steps, you should pack the *APP_FILES*, *METADATA* and *IMAGES* folder into a zip archive.
- It's recommended that the component archive file-name will be identical to the component's ID as defined in the *id* attribute of the VCXL root <vcx:Component> tag.

Your component is now ready to be uploaded and used in Visual composer.

Related Content

Using Flash Components in Visual Composer Models

VCXL Specification

For more information, visit the User Interface Technology homepage.

Copyright