SAP BusinessObjects Dashboard Design Component SDK User Guide

■ SAP BusinessObjects Dashboard Design Component SDK User Guide

2011-08-16

2011-08-16

Contents

About This Guide

Dashboard Design gives users the capability to easily visualize complex data modeled in Excel. The Dashboard Design Component SDK enables Flex developers to add new components in Dashboard Design. Flex developers can convert any flex widget or component and add it to the Dashboard Design designer.

Note:
The Dashboard Design Component SDK is for use with Adobe Flex SDK 2.0.1 Hotfix 3 only.

The SDK enables Adobe Flex developers to:

- Create new Adobe Flex visual components, such as charts and gauges.
- Create custom Adobe Flex connections.
- Add support for additional Excel functions using Adobe Flex.
- Take existing Adobe Flex components and re-use them within Dashboard Design.
- Package components as add-ons for installing into Dashboard Design.

Note:
The Dashboard Design Component SDK is available in English only.

Who should read this guide

Adobe Flex developers who want to understand how to develop new or re-use existing Adobe Flex components within Dashboard Design.

This guide assumes that you are familiar with Dashboard Design and also with using Adobe Flex Builder 3 and writing ActionScript.

Note:
Before using this guide, check for any changes or updates in the Dashboard Design Component SDK Release Notes and Dashboard Design Component SDK Installation Guide.

Known Issues and Workarounds

For information about known issues and workarounds for the Dashboard Design Component SDK, refer to the *SAP BusinessObjects Dashboard Design Component SDK Release Notes*.

Installing the Sample Add-on Components

- Dashboard Design with Service Pack 3 is installed.
- Dashboard Design Component SDK is installed.

  Note:
  For more information about installing the SDK, see the *SAP BusinessObjects Dashboard Design Installation Guide* and the *SAP BusinessObjects Dashboard Design Component SDK Installation Guide*.

Dashboard Design includes several sample add-ons that were created using the Component SDK. When you install Dashboard Design with the Component SDK, the sample files are saved in the Dashboard Design program folder in the path: `...\Xcelsius\SDK\Samples`. Each add-on sample includes the add-on file (.xlx file type) and all the source files for the add-on.

You can add the Dashboard Design Component SDK samples to Dashboard Design using one of the following methods:

- Use the Dashboard Design Add-On Manager.

  For more information about the integrated Add-On Manager, refer to the "Customizing Dashboard Design with Add-Ons" section in the *SAP BusinessObjects Dashboard Design User Guide*.

- Use the command-line version of the Add-On Manager to install each of the sample add-on (`.xlx`) files.

  For more information about using the Add-On Manager command line, see Appendix C: Add-On Manager Command Line Interface.

Overview of Steps to Create and Distribute Add-Ons

Develop and test your component:

- Create or re-use an existing Adobe Flex component.
  - For visual components, use the Dashboard Design Default Property Sheet or create your own custom property sheet using the property sheet API. For additional information about using the Dashboard Design Default Property Sheet, see Appendix A: Dashboard Design Default Property Sheet API Reference.
  - For custom connections, create your own property sheet for each connection using the property sheet API.

  Note:
  Excel functions do not use property sheets in Dashboard Design.

- Package the Debug version of your component as a Dashboard Design add-on (as an `.xlx` file type) using the Dashboard Design Add-On Packager.
- Install your add-on using the Add-On Manager in Dashboard Design or the command-line version.
- Test your installed add-on component.

Package and distribute your component:

- Package the Release Build version of your component as an add-on (as an `.xlx` file type) for distribution using the Dashboard Design Add-On Packager.
- Distribute your add-on to Dashboard Design users.
- Dashboard Design users install your add-on into Dashboard Design using the Add-On Manager or the command-line version.

Creating Adobe Flex Projects

Add-ons for Dashboard Design are created as Adobe Flex projects. You can create three types of Adobe Flex projects to extend Dashboard Design functionality: visual components, custom connections, and Excel functions.

After your add-on is added to Dashboard Design, visual components are accessible from the "Component" browser, custom connections are available in the Data Manager, and custom Excel functions are invisible during design time but, if they are used in the Excel spreadsheet, they execute in preview or export mode.

The process for creating the Adobe Flex project for each type of add-on is basically the same, although there are some steps that are specific to the add-on type.

To create Adobe Flex projects for Dashboard Design, the Dashboard Design SDK Framework must be included in your Adobe Flex project. Before you create a project, make sure `xcelsiusframework.swc` is installed in the *`<flex_sdk_2>`*`/frameworks/libs` folder (For example, `C:\Program Files\Adobe\Flex Builder3\sdks\2.0.1\frameworks\libs`).

For more information see the *Dashboard Design Component SDK Installation Guide*.

4.1 To create Adobe Flex projects for use with Dashboard Design

Create a basic Adobe Flex project:

1. Start Adobe Flex Builder 3
2. Select **File** > **New** > **Flex Project**. The "New Flex Project" wizard opens.
3. Set the **project name** and **location** for the type of add-on you are creating.

   Note:
   If the **Use Default Location** option is selected, your Adobe Flex Project is created in your workspace, which is `MyDocuments\Flex Builder 3` by default.

   The following table lists sample project names and locations for each add-on type:

Add-on Type	Project Name	Location
Visual Component	MyComponentSource	C:\xcelsius2008\componentdeveloper\MyComponentSource

Add-on Type	Project Name	Location
Connection	MyConnectionSource	C:\xcelsius2008\comonentdeveloper\MyConnectionSource
Excel Function	MyExcelFunctionSource	C:\xcelsius2008\componentdeveloper\MyExcelFunction Source

4. Click **Finish** to create your Adobe Flex Project.

The Adobe Flex project is created in the specified location.

After you create a basic Adobe Flex project, you need to compile it using Adobe Flex 2.

4.1.1 Avoid naming clashes for your add-on source

Many Adobe Flex developers can create new components for Dashboard Design and each component must have a unique name.To avoid name clashes with other developed add-ons, use a fully-qualified, unique class name that consists of the package name and your class name.

1. Decide on a package name for your component based on your company name and project name.

   The Dashboard Design Component SDK sample package names are based on company name and product name. This is a common practice for package names.

   For example: `com.businessobjects.xcelsius.sdk.samples`
2. Open your Adobe Flex project in Adobe Flex Builder.
3. Select **File** > **New** > **Folder**.
4. Type your package name.

   Hint: If you use a package name that contains backslashes (\) sub-folders are also created.

   For example: `com\businessobjects\xcelsius\sdk\samples`
5. Click **Finish**.

The package folder(s) are created and added to the Adobe Flex project.

4.1.2 Create an add-on source file

Since Adobe creates the Adobe Flex components using ActionScript, we recommend using ActionScript to develop components for Dashboard Design.

To avoid naming clashes, place your add-on source class in a package. For example, the following code sample creates a package for a basic horizontal slider:

```
package com.businessobjects.xcelsius.sdk.samples
{
    ...

    public class BasicHorizontalSlider extends HSlider
    {
        ...
    }
}
```

Note:
Visual components must inherit from `UIComponent`.

In addition, if you want to include connection refresh and status options in your custom connection component, you must extend Adobe Flex class `EventDispatcher` and implement the IConnection interface provided in the Xcelsius Component SDK Framework. You also must import any necessary Adobe Flex and Xcelsius classes that your connection extends and implements.

Note:
In Dashboard Design, connection refresh and status options are configured in the **Usage** tab on the Data Manager. For more information about using these options, see the "To set refresh options" and "To create load and idle messages" sections in the *SAP BusinessObjects Dashboard Design User Guide*.

For example, the following code sample creates a package for an RSS connector with a trigger:

```
package com.businessobjects.xcelsius.sdk.samples
{
    import flash.events.Event;
    import flash.events.EventDispatcher;
    import flash.events.IEventDispatcher;

    import xcelsius.rpc.IConnection;
    import xcelsius.rpc.TriggerType;

    ...

    public class RSSConnectorWithTrigger extends EventDispatcher implements IConnection
    {
        ...
    }
}
```

Note:
Every connection that implements `IConnection` must define all properties used in the **Usage** tab of the Data Manager. It must also overwrite the `trigger()` function that defines any events the connection must handle. When the connection finishes loading the data, it then needs to raise a `COMPLETE` event.

For examples of creating source files, see the following sample add-on components:

Note:
For information about using the Sample add-on files, see Installing the Sample Add-on Components

Example	Sample file/documnent
Creating components with MXML	The "MXMLHorizontalSlider" source code. . **Note:** For more information about using MXML components, see the Known Issues section
Creating a component with ActionScript	*Tutorial 1 Creating a basic horizontal slider* available with the SAP BusinessObjects Dashboard Design Component SDK documentation
Creating a basic custom connection that does not use refresh options and loading status	The "RSSConnector" source code.
Creating a connection that uses refresh options and loading status	The "RSSConnectorWithTrigger" source code

4.2 To add the add-on source to the top-level application MXML

Add your component to the top-level application MXML so it is included in the generated Shockwave Flash (SWF) file when you build your Adobe Flex project.

1. Find and open the top-level application MXML file.

   The top-level MXML file has the same filename as your Adobe Flex project and it contains the `<mx:Application>` tag.

   For example (in the Sales Funnel source Adobe Flex project sample), the MXML file is named `SalesFunnelSource.mxml`
2. Place the cursor between the `<mx:Application>` and `</mx:Application>` tags. Type < followed by your component name. A drop-down list is displayed with a list of components with that name.
3. Pick your component from the list and the application code updates with a reference to your component.

   For example (after `com.businessobjects.xcelsius.sdk.samples.SalesFunnel` was added to the application):

   ```
   <?xml version="1.0" encoding="utf-8" ?>
   <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" xmlns:ns="com.businessob
   jects.xcelsius.sdk.samples.*">
       <ns:SalesFunnel/>
   </mx:Application>
   ```

You have added a reference to your component in your top-level application MXML file. Now when you build your Adobe Flex project your component is also included in the generated SWF file.

4.3 To change how visual components look

When you create a visual component in Adobe Flex, you can customize its appearance based on styles available in Adobe Flex. To apply visual styles, such as border colors, font types, and font sizes, to visual components, use the `Style` metadata tag. For any styles you want users to be able to bind to the spreadsheet, use properties (with get and set functions).

Note:
Styles cannot be bound to the spreadsheet.

Use the Adobe Flex Style Explorer to see the styles you can customize in Adobe Flex. The Adobe Flex Style Explorer is available from the Adobe website at: http://examples.adobe.com/flex2/consulting/styleexplorer/Flex2StyleExplorer.html.

For additional information about controling the appearance of visual components, see the SAP BusinessObjects Dashboard Design SDK Tutorials available with your SDK documentation. Tutorial 2 provides an example of how to use the `Style` metadata tag and Tutorial 3 provides an example of how to use skins.

Compiling with Adobe Flex 2.0

To work with Dashboard Design, add-ons must be compiled using Adobe Flex SDK 2.0.1 Hotfix 3.

Configure your Adobe Flex project to compile with Adobe Flex SDK 2.0.1 Hotfix 3.

5.1 To compile Your Adobe Flex Project

1. Open the Adobe Flex project in Adobe Flex Builder.
2. Compile using Adobe Flex SDK 2.0.1 Hotfix 3.
   a. Select **Project** > **Properties**.
   b. In the "Properties" dialog box, select **Flex compiler**.
   c. In **Flex SDK version** select **Use a specific SDK** and from the list, select **Flex 2.0.1 Hotfix 3**.
3. Click **OK**.
4. If the warning message, "Overwriting Files", opens, click **OK**.

Adobe Flex SDK 2.0.1 Hotfix 3 is now selected to compile your add-on.

Compile your project for use in the Dashboard Design Default Property Sheet.

Creating Property Sheets

Visual components and custom data connections are configured in component property sheets. In general, you should create custom property sheets for your visual components, but to get started, you may want to use the Dashboard Design default property sheet. However, the default property sheet does not work with custom connections, so you must create a custom property sheet for custom connections.

6.1 To use the default property sheet

Although it is a good idea to create custom property sheets for configuring custom components, you can use the default property sheet as a starting point.

For examples of using the Dashboard Design default property sheet with a visual component, see the *Tutorial 2 Styling a Basic Horizontal Slider* and *Tutorial 3 Skinning a Basic Horizontal Slider* guides available with your SAP BusinessObjects Dashboard Design Component SDK documentation. For example add-on components using the Dashboard Design default property sheet, see the `BasicHorizontalSlider`, `BasicHorizontalSliderwithStyling`, or `BasicHorizontalSliderwithSkinning` samples. For more information about installing the add-on samples, see Installing the Sample Add-on Components.

Note:
Before you use the Dashboard Design default property sheet with your components refer to the additional information about the default property sheet API in Appendix A: Dashboard Design Default Property Sheet API Reference.

6.2 To create custom property sheets

Create a new Adobe Flex project for your custom property sheet. You can use the top-level application MXML to create your property sheet.

Note:
Do NOT use `xcelsius.propertySheets.interfaces.PropertySheetFunctionNames`. It is not supported as part of the API.

For additional information about creating custom property sheets for visual components, see the *Dashboard Design SDK Tutorial 4* available with the SAP BusinessObjects Dashboard Design SDK documentation and the sample add-on component, `CustomPropSheetHorizontalSlider`.

For an example of a custom property sheet for a connection, see the the sample add-on component, `RSSConnectorWithTrigger`.

For more information about using the Sample add-on components, see Installing the Sample Add-on Components.

Related Topics

- Appendix B: Custom Property Sheet API Reference

Testing Your Component

After you create an Adobe Flex Project, test it in Dashboard Design in design mode, in a preview, and at run-time after exporting it as as a SWF.

Note:
You can test the property sheets for custom components in design mode, but since SWF source files are not executed until the model is run, you cannot test custom connections and functions in design mode. To test custom connections and functions, you can use the **Preview** button to run the model. This way, you can test the connections and functions within Dashboard Design without having to export the model.

1. Package the Debug version of your component for testing

   Before you can test your component in Dashboard Design you need to package your component as a Dashboard Design add-on (add-ons have an `.xlx` file extension) using the Dashboard Design Add-On Packager.

   Note:
   For testing purposes, use the debug version of your SWF so the additional metadata for your component is available to the Dashboard Design default property sheet.

   Package your debug SWF (for testing purposes) along with the Dashboard Design default property sheet unless you created a custom property sheet.

   For more information about using the Dashboard Design Add-On Packager, see To create Dashboard Design add-on packages (.xlx files) Refer to "Package your component as a Dashboard Design Add-On" .

2. In Dashboard Design, use the Add-on Manager, to install your packaged debug component.

   For more information about using the Add-on Manager, see the “Customizing Dashboard Design with Add-On Components” section in the *SAP BusinessObjects Dashboard Design User Guide*.

3. Debug your component using trace statements
   - Using Mozilla Firefox
     a. To view trace statements without the Adobe Flex Builder debugger there is a Firefox add-on called Flash Tracer which displays ALL trace statements from SWF files.
     a. Download the Mozilla Firefox Flash Tracer add-on, available from: `https://addons.mozilla.org/en-US/firefox/addon/3469`
     b. From within Mozilla Firefox, enable the Flash Tracer. Select **Tools** > **Add-ons** > **Flash Tracer**. The Flash Tracer window is opened.
     c. Select **Options**. The options window opens.

Set the output option to `C:\Documents and Settings\${YOUR_USER_NAME}\Application Data\Macromedia\Flash Player\Logs\flashlog.txt`.

For example: `C:\Documents and Settings\bobj\Application Data\Macromedia\Flash Player\Logs\flashlog.txt`

d. Select **OK** to close the options window.

Mozilla Firefox Flash Tracer displays trace information from any SWF file that is running including SWF files running in Dashboard Design.

- Using Java 1.6 (version 6) or later

a. To view trace statements without the Adobe Flex Builder debugger there is a Google code Java utility called Flash Tracer which displays ALL trace statements from SWF files.

a. Download the Google code Flash Tracer, available from: `http://code.google.com/p/flash-tracer/`

b. Install the Flash Tracer files.

Copy `Tracer_v1.1.17.jar` and `Tracer.bat` files to: `C:\Documents and Settings\${YOUR_USER_NAME}\Application Data\Macromedia\Flash Player\Logs`

For example: `C:\Documents and Settings\bobj\Application Data\Macromedia\Flash Player\Logs`

c. Edit `Tracer.bat` to point to Java 1.6 (version 6) or later.

For example:

```
"C:\Program Files\Java\jre1.6.0_03\bin\java.exe" -jar Tracer_v1.1.7.jar
```

d. Run `Tracer.bat`.

Java Flash Tracer displays trace information from any SWF file that is running including SWF files running in Dashboard Design.

Packaging Adobe Flex Projects as Add-Ons

The Dashboard Design Add-On Packager provides an easy way to package your component as an add-on `.xlx` file for distribution to Dashboard Design users. Dashboard Design users can simply use the Dashboard Design Add-on Manager to install or remove your add-on.

For more information about using the **Add-on Manager**, see the “Customizing Dashboard Design with Add-Ons” section in the *SAP BusinessObjects Dashboard Design User Guide*.

8.1 To create a release version of the SWF

While you are testing your component, you can use the debug version of your SWF.

If you created a custom property sheet, in Adobe Flex Builder 3, use the **File** > **Export** > **Release Build** option to create the release version.

Note:
If you update your add-on package `.xlx` before you distribute it, make sure you use the release version of the SWF instead of the debug version.

If your component uses the default property sheet, before you package your component, complete the following steps to use the Adobe Flex MXML compiler to build a release version of the SWF that includes the metadata required for use with the Dashboard Design Default Property Sheet.

1. Locate your Adobe Flex MXML command-line compiler.

   The MXML compiler (`mxmlc.exe`) is located in the Adobe Flex SDK 2 `bin` folder.

   For example: `C:\Program Files\Adobe\Flex Builder 3\sdks\2.0.1\bin\mxmlc.exe`
2. Locate the top-level application MXML file for your component.

   For example: `SalesFunnelSource.mxml`
3. Create an MXML compiler configuration file for your application MXML and save it in the same directory as your top-level application MXML file.

   Note:
   To ensure the MXML compiler uses the configuration file when you build your MXML application, name the configuration file with the same name as your application MXML file, but replace `.mxml` with `-config.xml`.

   For example, for an application MXML file called `SalesFunnelSource.mxml`, name the configuration file `SalesFunnelSource-config.xml`.

4. In the MXML compiler configuration file, add configuration information to include metadata for the Dashboard Design Default Property Sheet.

   For example,

```
<?xml version="1.0"?>
<flex-config xmlns="http://www.adobe.com/2006/flex-config">
    <compiler>
        <!-- Add the Xcelsius 2008 Component SDK framework to the Adobe Flex classpath. -->
        <library-path append="true">
            <path-element>${flexlib}\libs\xcelsiusframework.swc</path-element>
        </library-path>

        <!-- Keep additional metadata needed for use with the Xcelsius 2008 Default Property Sheet. -->
        <keep-as3-metadata>
            <name>Inspectable</name>
            <name>Style</name>
            <name>CxInspectableList</name>
        </keep-as3-metadata>
    </compiler>
</flex-config>
```

5. Open a Command Prompt.
6. Navigate to the directory that contains your top-level application MXML file.

   For example, `cd C:\xcelsius\sdk\samples\SalesFunnel\SalesFunnelSource\src`
7. Use the Adobe Flex MXML command-line compiler to compile your top-level application MXML file into a release SWF.

   For example, `C:\Program Files\Adobe\Flex Builder 3\sdks\2.0.1\bin\mxmlc.exe" SalesFunnelSource.mxml -output SalesFunnelSource.swf`

A release version of the SWF is created and includes the metadata required to use the Dashboard Design Default Property Sheet.

8.2 To create Dashboard Design add-on packages (.xlx files)

Although add-ons can contain multiple visual components, connections, and functions, the Add-on Manager can only install or remove the entire package at once. It cannot add or remove individual items within the package. If you expect that users might want to add or remove specific items in the add-on individually, create an separate package for each item.

1. Start **Dashboard Design Add-On Packager**.
2. Click the **Details** tab and enter the following information:

Option	Description
Name	Enter the name of your add-on. For example, `Sales Funnel` **Note:** For packages that contain only a set of functions, include the names of the functions in this field. Since custom functions are not visible in design mode, the only way to see which custom functions have been installed is through the Add-On Manager.
Publisher	The name of the add-on creator. For example, `SAP BusinessObjects`
Website	The name of the website where the package is available. For example, `http://www.sap.com/solutions/sapbusinessobjects/index.epx`
Publish Date	The date the package was published.
Version	The add-on version.
Description	A brief description of the add-on. For example, `Sales Funnel Xcelsius 2008 Component SDK sample add-on.` **Note:** For packages that contain only a set of functions, include the names of the functions in this field. Since custom functions are not visible in design mode, the only way to see which custom functions have been installed is through the Add-On Manager.

3. Select one of the following tabs:

Tab	Description
Visual Components	Allows you to add a visual component to the package. Continue with step 4.
Connections	Allows you to add custom connections to the package. Continue with step 5.
Function	Allows you to add custom Excel functions to the package. Continue with step 6.

4. To add a visual component to the add-on, click **Add Component** and, on the "Add New Component" dialog box, enter the following information:

Option	Description
Class Name	The fully qualified class name that identifies the item in the component SWF. For example: `com.businessobjects.xcelsius.sdk.samples.Sales Funnel` **Note:** Visual components are referenced by their **Class Name**. If you enter the**Class Name** incorrectly, when you place the add-on component on the Dashboard Design canvas, it will not be visible.
Display Name	The name that appears in the "Component" and "Object" browsers by default to represent your component. For example: `Sales Funnel`
Component SWF	The SWF file that you compiled from your Adobe Flex project. You can browse to the location of this file.
Category	The heading that your component appears under in the "Component" browser. For example: `Add-Ons`.
Large Icon	Specify a bitmap file (48 by 48 pixels) that will represent your component in the "Component" browser.
Small Icon	Specify a bitmap file (16 by 16 pixels) that will represent your component in the "Object" browser.
Property Sheet SWF	Enter the path to the property sheet users can use to edit your component properties and styles. You can set this to either the Dashboard Design default property sheet SWF or your own custom property sheet SWF. **Note:** The Dashboard Design default property sheet (`PropertyInspector.swf`) is installed as part of the Dashboard Design Component SDK Installation. For example,`C:\Program Files\Business Objects\Xcelsius\SDK\bin\PropertyInspector.swf`
Version	A version number for your component. (Optional)

.

5. To add a custom connection to the add-on, click **Add Connection** and, on the "Add New Connection" dialog box, enter the following information:

Option	Description
Class Name	The fully qualified class name that identifies your connection in the connection SWF. For example, `com.businessobjects.xcelsius.sdk.samples.RSS Connector`
Display Name	The name that represents your component in the Dashboard Design Data Manager. For example: `Basic RSS Connector`.
Connection SWF	The SWF file that you compiled from your Adobe Flex project. You can browse to the location of this file.
Small Icon	A bitmap file (16 by 16 pixels) that represents your connection in the Data Manager list of available connections.
Large Icon	A bitmap file (32 by 32 pixels) that will represent your connection in the Data Manager list of added connections.
Property Sheet SWF	The location of the property sheet SWF you created for users to edit the connection properties. **Note:** This must be a custom property sheet SWF.
Version	A version number for your connection (optional).

6. To add a custom functions to the package, click **Add Function** and, on the "Add New Function" dialog box, enter the following information:

Option	Description
Function Name	A name for your function (optional). The name helps users determine which functions have been included in the add-on. It is a good practice to include the names of the functions here.
Function SWF	The SWF file that you compiled from your Adobe Flex project. You can browse to the location of this file.
Version	A version number for your function (optional).

7. Click **OK.**
8. To add more items to the package, repeat steps 3 to 7, as appropriate.

   Note:
   Two components in the same package cannot have the same class name.
9. Click the **Build** tab.
10. Click **Build Package**.
11. In the dialog box, type a filename for the add-on `.xlx` package and click **Save**.

    If the build is successful, the **Build Results** dialog box displays the message Package built successfully!
12. For easier updating in the future, click **File** > **Save As** to save your project settings.

13. In the "Save As" dialog box, enter a name for your project settings.

    The project settings are saved with a `.xlp` file extension.

    Note:
    After you save your `.xlp` file, you can use relative paths (relative to the `.xlp`) for your SWF and Bitmap files. For an example of how to do this, see the sample XLP projects.

8.3 To update add-on packages

If you saved your add-on package project as an .xlp file when you created the package, you can update the package, as necessary.

Note:
If you didn't save the package project, you must create the add-on package before you can update it.

1. Start **Dashboard Design Add-On Packager**.
2. Click **File** > **Open** and open your add-on project (`.xlp` file).
3. Click each of the **Details**, **Visual Components**, **Connections** and **Functions** tabs and update the information as required.

   Note:
   You can delete, modify, or add components as you did when you created the package.

4. When you have finished updating the information, click the **Build** tab.
5. Click **Build Package**.
6. On the dialog box, type a filename for the modified `.xlx` file and click **Save**.

   If the build is successful, the "Build Results" dialog box displays Package built successfully!

7. To save the updated add-on project, click **File** > **Save**.

Distribute your add-on

Distribute your Dashboard Design Add-On `.xlx` to Dashboard Design users. Dashboard Design users can install and remove your add-on using the Add-On Manager in Dashboard Design or the command-line version.

For more information about using the Dashboard Design Add-On Packager, see To create Dashboard Design add-on packages (.xlx files) Refer to "Package your component as an Dashboard Design Add-On" .

Appendix A: Dashboard Design Default Property Sheet API Reference

The Dashboard Design Default Property Sheet is provided so you can quickly get your visual component into Dashboard Design and provide users with a way to edit properties and styles of the component. However, to provide the best experience for the Dashboard Design users, create a property sheet that is customized for your add-on.

Note:
The default property sheet is not localized. If you want a localized property sheet or to use more complex data types and 2d arrays, create a custom property sheet. For more information about custom property sheets, see Appendix B: Custom Property Sheet API Reference.

In Dashboard Design, when users select your visual component on the canvas, the default property sheet:

1. Retrieves the public properties (with a getter, setter, or both) and styles from the SWF file.

   Note:
   To hide unsupported Adobe Flex properties in your add-on, you can use the `CxInspectableList` metadata tag.

2. Displays the public properties and styles in a grid where users can configure them:
   - Property (except Array properties) and style values can be edited in the grid.
   - Property values including Array properties can be bound to a range in the spreadsheet.

Note:
Only simple data types are supported including Object, Number, and Array. 2d arrays are not supported.

10.1 Hiding unsupported Adobe Flex properties

Certain Adobe Flex component properties (including `x`, `y`, `width`, `height`) are not supported in Xcelsius. Changing the property values for unsupported properties will cause unexpected behavior when the model is previewed or exported from Dashboard Design. When you create your add-on, you must hide unsupported properties so they don't appear on the property sheet.

To show only properties that you want the user to view and edit in the property sheet, add the Dashboard Design Default Property Sheet API `CxInspectableList` metadata to your component source code (above the class name) .

Note:
For an MXML file, add the `CxInspectableList` metadata in an <mx:Metadata> tag.

For example, to restrict the list of properties to `title` and `showTitle` for the `BasicHorizontal Slider`, add the following metadata to your source code:

```
package com.businessobjects.xcelsius.sdk.samples
{
    ...

    [CxInspectableList ("title", "showTitle")]
    public class BasicHorizontalSlider extends HSlider
    {
        ...
    }
}
```

For an MXML file, use the following format:

```
<mx:Metadata>
      [CxInspectableList ("title", "showTitle")]
</mx:Metadata>
<mx:Script>
      <![CDATA[
             ...
      ]]>
</mx:Script>
```

10.2 Binding properties to the spreadsheet

In the default property sheet, each property in the grid that has a getter or setter function can be bound to a cell or range of cells in the spreadsheet using the **Bind** button.

Note:
If multiple cells are selected for a property that is not an `Array` then only the first cell in the range is used when the range value changes.

The effect of changing a property value depends on whether the property has a getter function, a setter function, or both:

- Getter function only

  When the property changes the value is written to the spreadsheet range. By default, you cannot declare a getter without a setter to be bindable. You have to be explicit and declare the `property Changed` event name in the `Bindable` tag and then notify any listeners when the property changes using `dispatchEvent`.

  For example:

  ```
  [Bindable("propertyChange")]
  public function get value():Number
  {
      ...
  }
  /**
   * Example of how to notify when the
  ```

```
 "value" property changes.
 */
public function notifyValueChanged():void
{
   var event:Event = PropertyChangeEvent
   .createUpdateEvent(this, "value", null,
   value);
   dispatchEvent(event);
}
```

- Setter function only

  When the spreadsheet range value changes the value is passed to the property setter function.

  For example:

  ```
  [Bindable]
  public function set value(val:Number):void
  {
      ...
  }
  ```

- Both Getter and Setter functions

  When the spreadsheet range value changes the value is passed to the property setter function, and when the property value changes the value is written to the spreadsheet range.

  For example:

  ```
  [Bindable]
  public function get value():Number
  {
      ...
  }
  public function set value(val:Number):void
  {
      ...
  }
  ```

10.3 Customizing styles and properties

To determine which properties and styles to make available to users, the default property sheet looks for specific metadata tags defined in the SWF, including `CxInspectableList`, which hides properties. You can also use the standard Adobe Flex `Inspectable` and `Style` tags to further customize the default property sheet for certain types of properties or styles including `Boolean` and `Color`.

Customize the style or property value editor in the default property sheet with additional metadata (the default editor type is `String`):

Tag	Description
Style: Font Family (a font name including `Verdana`, `Arial`) For example, `[Style(name="titleFontFamily", defaultValue="Verdana", type="String", format="fontFamily")]`	A combo box in the property sheet allows users to select a font from the list of fonts available on their system.
Style: Font Weight (`normal` or `bold`) For example: `[Style(name="titleFontWeight", defaultValue="normal", type="String", format="fontWeight")]`	A combo box in the property sheet allows users to choose a normal or bold font weight.
Style: Font Style (`normal` or `italic`) For example: `[Style(name="titleFontStyle", defaultValue="normal", type="String", format="fontStyle")]`	A combo box in the property sheet allows users to choose an italic or normal font style .
Style: `Color` For example: `[Style(name="titleFontColor", defaultValue="0x000000", type="uint", format="Color")]`	A Color Selector in the property sheet allows users to pick a color.
Style: `Boolean` (`true` or `false`) For example: `[Style(name="invertThumbDirection", defaultValue="false", type="Boolean")]`	A combo box in the property sheet allows users to select `true` or `false`
Style: Known list of values (an enumerated list). For example: `[Style(name="dataTipPlacement", defaultValue="top", type="String", enumeration="top, left, right, bottom")]`	A combo box in the property sheet allows users to pick from a list of values
Style: `Number` For example: `[Style(name="tickThickness", defaultValue="1", type="Number")]`	A text box in the property sheet allows the user to type a number (restricted to numeric characters) .

Tag	Description
Inspectable: `Boolean` (`true` or `false`) For example: `[Inspectable(type="Boolean")]`	A combo box in the property sheet allows users to pick `true` or `false`.

10.4 Naming component properties

For consistency with other Dashboard Design property sheets, use the following style conventions when naming component properties:

Type	
`Array`	Use a plural ending to imply that you can have more than one value. For example (for an `Array` of labels to show on a chart): `chartLabels`
`Color`	Include the word "color" at the end of your style or property name. For example (for a border color): `borderColor`

10.5 Retain metadata required for the default property sheet (required)

Before you compile your project, you must add compiler arguments to ensure the Adobe Flex compiler keeps the metadata required by the default property sheet:

From within your Adobe Flex project in Flex Builder:

1. Click **File** > **Properties**.
2. In the "Properties" dialog box, select **Flex Compiler**.
3. Add the following additional compiler arguments to keep metadata for the `Inspectable` and `Style` tags, as well as the `CxInspectableList` tag.

   For example: `-locale en_US -keep-as3-metadata+=Inspectable,Style,CxInspectableList`

Compile your release SWF to keep the metadata. To use the default property sheet.

10.6 Displaying and hiding properties in the default property sheet

If the default property sheet is displaying more properties than you want, use the CxInspectableList tag to restrict the list of properties. Also, make sure you add the extra compiler arguments to keep the additional metadata required by the default property sheet and compile the release version of your SWF using MXMLC to keep the extra metadata.

Related Topics

- Hiding unsupported Adobe Flex properties
- Retain metadata required for the default property sheet (required)

Appendix B: Custom Property Sheet API Reference

Although you can use the default property sheet to get visual components working quickly in Xclesius, to provide the best user experience, you can create a customer property sheet for your add-on. Custom connections cannot use the default property sheet, so you must create a custom property sheet for connection add-ons.

When you build a custom property sheet, you have full control over the layout and the properties available for users to view, edit, and bind to the spreadsheet.

Note:
Only simple data types are supported including Object, Number and Array.

11.1 High-level steps for creating custom property sheets

This section provides a high-level description for each step required to create a custom property sheet. For a walkthrough on creating a custom property sheet tht includes more details for each steps, see the *SAP BusinessObjects Dashboard Design SDK Tutorial 4* available with your Dashboard Design SDK documentation.

To see an example custom property sheet for a visual component, refer to the `CustomPropSheetHorizontalSlider` sample add-on component.

1. Create an Adobe Flex project for your custom property sheet. You can use the top-level application MXML to create your property sheet..
2. Drag and drop Adobe Flex MXML visual components to create your custom property sheet user interface.
3. Create an instance of the custom property sheet proxy (`xcelsius.propertySheets.impl.PropertySheetExternalProxy`) that allows communication between the custom property sheet and Dashboard Design.

   For example:

   ```
   import xcelsius.propertySheets.impl.PropertySheetExternalProxy;
   import xcelsius.propertySheets.interfaces.IPropertySheetProxy;

   // Create an instance of the proxy between the custom property sheet and Xcelsius 2008.
   protected var proxy:IPropertySheetProxy = new PropertySheetExternalProxy();
   ```

4. The custom property sheet loads every time the user selects the component or the connection to view its properties in Dashboard Design. Add in initialization ActionScript code to initialize the custom property sheet when it is loaded and to notify Dashboard Design (via the proxy) when the custom property sheet has finished loading.

5. Handle localization based on Dashboard Design language setting.
6. Add a helper function called `getPropertyBindDisplayName` that returns null if the property is not bound to the spreadsheet, otherwise it returns the displayable text for the cell range that the property is bound to.
7. Add a helper function that lets the Dashboard Design user bind a property to the spreadsheet (`initiateBind`) and another helper function that handles the user cell selection (`continueBind`) to create the binding and update the custom property sheet.
8. Add calls to set the property via the proxy when a value changes in the custom property sheet user interface.

   For example (let the Dashboard Design user edit a number value in the custom property sheet):

   ```
   <mx:TextInput id="valueEditor"  ...  change="proxy.setProperty('value', Number(valueEditor.text));" />
   ```

9. Add calls to set a style via the proxy when a value changes in the custom property sheet user interface.

   For example (let the Dashboard Design user pick a color for a border style in the custom property sheet):

   ```
   <mx:ColorPicker id="colorEditor" ... change="proxy.setStyle('borderColor', colorEditor.selectedColor);"
    />
   ```

10. Add calls to let the user bind a property (styles cannot be bound) to the spreadsheet in the custom property sheet.

    For example (using an Adobe Flex `Button` to start the bind process for the `value` property):

    ```
    <mx:Button  ...  click="initiateBind('value')" />
    ```

11.2 Bindings to and from the spreadsheet

InputBindings and OutputBindings types

When you create a binding, you also specify how to handle passing values from the spreadsheet to the property set function, and how to pass values from the property get function to the spreadsheet:

- `InputBindings.SINGLETON` or `OutputBindings.SINGLETON`

  A single value is passed.

  Note:
  If the user selects multiple cells only the first cell is used.

- `InputBindings.ARRAY2D` or `OutputBindings.ARRAY2D`

  Multiple columns or rows of cells are passed as a 2D array.

  For example (user selects cells **A1:C2**): `[[A1,B1,C1],[A2,B2,C2]]`

- `InputBindings.ARRAY` or `OutputBindings.ARRAY`

  Multiple columns or rows of cells are passed as a flattened 1D array.

For example (user selects cells **A1:C2**): `[A1,B1,C1,A2,B2,C2]`

Binding directions

Binding directions between the spreadsheet and the component property can be:

- Two way between the spreadsheet and component property
- One way: spreadsheet to the component property
- One way: component property to the spreadsheet

You can also bind sub-elements and sub-properties to the spreadsheet.

Related Topics

- Two-direction bindings
- Spreadsheet to component bindings
- Component to spreadsheet bindings
- Sub-element and Sub-property Binding

11.2.1 Two-direction bindings

When you use bindings with two-way direction, the spreadsheet is updated when the property value changes and the property value is updated when the spreadsheet changes.

Note:
The component property needs to have both getter and setter functions for this to work and they must be marked as `Bindable`.

```
[Bindable]
public function get value():Number
{
    ...
}
public function set value(val:Number):void
{
    ...
}
```

1. Bind a single value property.

   For example:

   ```
   proxy.bind(propertyName, null, bindingId, BindingDirection.BOTH, InputBindings.SINGLETON, OutputBind
   ings.SINGLETON);
   ```

2. Bind an array as a 2D array.

   For example:

   ```
   proxy.bind(propertyName, null, bindingId, BindingDirection.BOTH, InputBindings.ARRAY2D, OutputBindings.AR
   RAY2D);
   ```

3. Bind an array as a 1D array.

   For example:

   ```
   proxy.bind(propertyName, null, bindingId, BindingDirection.BOTH, InputBindings.ARRAY, OutputBindings.ARRAY);
   ```

11.2.2 Spreadsheet to component bindings

If you use spreadsheet to component bindings, the property value is updated when the spreadsheet changes. Known as an output from the spreadsheet to the user (or in this case the component property).

Note:
The component property needs to have a setter function for this to work.

```
[Bindable]
public function set value(val:Number):void
{
    ...
}
```

1. Bind a single value property.

   For example:

   ```
   proxy.bind(propertyName, null, bindingId, BindingDirection.OUTPUT,"", OutputBindings.SINGLETON);
   ```

2. Bind an array as a 2D array.

   For example:

   ```
   proxy.bind(propertyName, null, bindingId, BindingDirection.OUTPUT, "", OutputBindings.ARRAY2D);
   ```

3. Bind an array as a 1D array.

   For example:

   ```
   proxy.bind(propertyName, null, bindingId, BindingDirection.OUTPUT, "", OutputBindings.ARRAY);
   ```

11.2.3 Component to spreadsheet bindings

If you use component to spreadsheet bindings, the spreadsheet is updated when the component property value changes. Known as an input from the user (or in this case the component property) to the spreadsheet.

Note:
The component property needs to have a getter function. By default, a getter function cannot be marked `Bindable` on its own. You need to specify the `propertyChange` event. Use the `dispatchEvent` call to dispatch a `propertyChange` event.

```
[Bindable("propertyChange")]
public function get value():Number
{
    ...
}
/**
 * Example of how to notify when the
 "value" property changes.
*/
public function notifyValueChanged():void
```

```
{
   var event:Event = PropertyChangeEvent
   .createUpdateEvent(this, "value", null,
   value);
   dispatch(event);
}
```

1. Bind a single value property.

   For example:

   ```
   proxy.bind(propertyName, null, bindingId, BindingDirection.INPUT, InputBindings.SINGLETON, "");
   ```

2. Bind an array as a 2D array.

   For example:

   ```
   proxy.bind(propertyName, null, bindingId, BindingDirection.INPUT, InputBindings.ARRAY2D, "");
   ```

3. Bind an array as a 1D array.

   For example:

   ```
   proxy.bind(propertyName, null, bindingId, BindingDirection.INPUT, InputBindings.ARRAY, "");
   ```

11.2.4 Sub-element and Sub-property Binding

Most connections allow binding property sub-elements or sub-properties that are of Array type to the Excel spreadsheet. For example of a sub-element binding, the RSS Connector has a property called `rssItems` that contains all of the feed items returned from the loading process. Each feed item has a title, a description, a link, and a published date. You can then bind an individual feed item or each element of a single feed, such as title, to the Excel spreadsheet. Refer to the sample "RSS Connector With Trigger" and its source code for an example.

To use sub-element or sub-property binding, first create a chain accessor, which provides a mechanism to access and assign a specific element or property within an Array, and pass it as the second parameter to the `bind` method of the property sheet proxy as follows:

Sub-element binding:

```
// create a chain to bind to rssItems[0]
var chain:Array = [0];                    //access first feed item: an array of 4 elements
// binding to rssItems
proxy.bind("rssItems", chain, bindingID, BindingDirection.INPUT, InputBindings.ARRAY, "");
```

or

```
// create a chain to bind to rssItems[0][2]
var chain:Array = [0, 2];                 //access the third element of the first feed item
// binding to rssItems
proxy.bind("rssItems", chain, bindingID, BindingDirection.INPUT, InputBindings.SINGLETON, "");
```

Sub-property binding:

```
// create a chain to bind to sub-property data[0].xValue
var chain:Array = [0, "xValue"];                 //access xValue property of the first object
// binding to property "data"
proxy.bind("data", chain, bindingID, BindingDirection.INPUT, InputBindings.SINGLETON, "");
```

Note:
The binding type on either InputBindings or OutputBindings should match the type of the end point (data[0].xValue) and not the property (data).

11.3 Localizing property sheets

To get the language setting from Dashboard Design, create a variable to hold the locale string:

```
var locale:String = proxy.getLocale();
```

`locale` variable can then hold one of these values:

- de (German)
- ko (Korean)
- es (Spanish)
- en (English)
- pt (Portugese)
- sv (Swedish)
- fr (French)
- zh_CN (Chinese - simplified)
- zh_TW (Chinese - traditional)
- ru (Russian)
- nl (Dutch)
- it (Italian)
- ja (Japanese)

The property sheet should load corresponding strings based on the locale variable. The default locale should be in English. For an example of localizing the property sheet, see the `CustomPropSheetHorizontalSlider` sample add-on source code.

11.4 ActionScript Documentation (ASDocs) and Class Diagram

Refer to the Dashboard Design Component SDK ActionScript Documentation (ASDocs) and the Dashboard Design Component SDK Class Diagram for more information on the classes and methods of the Dashboard Design Component SDK custom property sheet API.

Refer to Tutorial 4 for more information on how to create a custom property sheet using the custom property sheet API.

Refer to the `CustomPropSheetHorizontalSlider` sample for an example custom property sheet.

Appendix C: Add-On Manager Command Line Interface

This section provides an overview for using the command-line interface to install add-on files. For more information about using the integrated Add-On Manager in Dashboard Design, see the *Using Advanced Features in Dashboard Design User Guide*.

Caution:
Before installing or removing add-on packages, close Dashboard Design.

To use the command-line version of the Add-On Manager to install add-on (.xlx) files:

1. Open the command prompt and go to the directory where Dashboard Design Component SDK is installed. For example, `C:\Programs Files\BusinessObjects\Xcelsius\SDK`.
2. For usage, type AddOnMgr_CL and press **Enter**.
3. Use the following commands to add and remove add-ons:

   Note:
   If paths contain spaces, they must be enclosed in quotation marks.

Action	Command
To list all installed packages	`AddOnMgr_CL -l`
To install a package	`AddOnMgr_CL -i XLX_PATH`
To install all packages in a directory	`AddOnMgr_CL -i -dir XLX_DIR_PATH`
To remove a package	`AddOnMgr_CL -r XLX_GUID` or `AddOnMgr_CL -r -l PACKAGE_LIST_INDEX` **Note:** XLX_GUID and PACKAGE_LIST_INDEX are available when you list all installed packages.
To remove all installed packages	`AddOnMgr_CL -r -all`

More Information

Information Resource	Location
SAP BusinessObjects product information	http://www.sap.com
SAP Help Portal	Navigate to http://help.sap.com/businessobjects and on the "SAP BusinessObjects Overview" side panel click **All Products**. You can access the most up-to-date documentation covering all SAP BusinessObjects products and their deployment at the SAP Help Portal. You can download PDF versions or installable HTML libraries. Certain guides are stored on the SAP Service Marketplace and are not available from the SAP Help Portal. These guides are listed on the Help Portal accompanied by a link to the SAP Service Marketplace. Customers with a maintenance agreement have an authorized user ID to access this site. To obtain an ID, contact your customer support representative.
SAP Service Marketplace	http://service.sap.com/bosap-support > Documentation • Installation guides: https://service.sap.com/bosap-instguides • Release notes: http://service.sap.com/releasenotes The SAP Service Marketplace stores certain installation guides, upgrade and migration guides, deployment guides, release notes and Supported Platforms documents. Customers with a maintenance agreement have an authorized user ID to access this site. Contact your customer support representative to obtain an ID. If you are redirected to the SAP Service Marketplace from the SAP Help Portal, use the menu in the navigation pane on the left to locate the category containing the documentation you want to access.
Docupedia	https://cw.sdn.sap.com/cw/community/docupedia Docupedia provides additional documentation resources, a collaborative authoring environment, and an interactive feedback channel.
Developer resources	https://boc.sdn.sap.com/ https://www.sdn.sap.com/irj/sdn/businessobjects-sdklibrary

Information Resource	Location
SAP BusinessObjects articles on the SAP Community Network	https://www.sdn.sap.com/irj/boc/businessobjects-articles These articles were formerly known as technical papers.
Notes	https://service.sap.com/notes These notes were formerly known as Knowledge Base articles.
Forums on the SAP Community Network	https://www.sdn.sap.com/irj/scn/forums
Training	http://www.sap.com/services/education From traditional classroom learning to targeted e-learning seminars, we can offer a training package to suit your learning needs and preferred learning style.
Online customer support	http://service.sap.com/bosap-support The SAP Support Portal contains information about Customer Support programs and services. It also has links to a wide range of technical information and downloads. Customers with a maintenance agreement have an authorized user ID to access this site. To obtain an ID, contact your customer support representative.
Consulting	http://www.sap.com/services/bysubject/businessobjectsconsulting Consultants can accompany you from the initial analysis stage to the delivery of your deployment project. Expertise is available in topics such as relational and multidimensional databases, connectivity, database design tools, and customized embedding technology.