

How to

Use Custom Controls

Tools: AltioLive Studio,
AltioLive Designer.

Skill Level: Advanced

Summary: This document explains how custom-written Java controls can be plugged into the AltioLive client. Java programming knowledge is required for this module.

Integra SP

88 Wood Street London
EC2V 7RS
United Kingdom

www.altio.com
tel: +44 (0) 20 8528 1045

Contents

Introduction	3
Configuration	4
Required jar files.....	4
Control Interface.....	4
Practical Exercise: Swing Label Control	5
File Directories.....	5
Adding a Constructor.....	5
Generating a Control Descriptor.....	7
Compiling and Deploying the Control.....	8
Creating and displaying the JLabel	9
Example of other features associated with Custom Controls	12
Properties	12
Run Time Properties	12
Design Time Properties.....	13

Inherited Properties.....	16
Making Properties Available to Other Controls.....	16
Triggers and Events.....	17
Mouse Events	17
Triggers	18
Live Data	19
Control Dependencies	21
Defining Default Values	21

Introduction

AltioLive provides many of the controls required for a developer to produce a functional screen for their client. Occasionally a control may be required that is not available in out of the box AltioLive, to accommodate this AltioLive has a Control API which allows developers to write their own controls to be used in the AltioLive IDE.

A custom control:

- Runs within the Altio client,
- May be visual or non-visual,
- Is written in Java and implements the **com.altio.client.Control** interface,
- Can be distributed as part of a control archive,
- Integrates with the Altio Designer for building Altio applications.

Once created the custom control can be deployed as either a JAR and loaded to the client when the Altio Smart Client is downloaded or as a class file and downloaded on an as required basis.

Please note: Each custom control adds to the size of the client footprint on the user machine.

This section will provide an introduction to creating AltioLive custom controls by creating a HTML label control.

Configuration

Required jar files

To be able to build a custom control the Altio client jar will need to be available in the class path. The client jar can be found in the client directory of the Altio web application e.g. **altio54/client**. For tomcat on a windows machine the directory structure will be something like:

C:\Program Files\AltioLiveStudio5.4\Enterprise\tomcat\webapps\altio54\client.

Control Interface

A custom control may make use of many classes, but there must be a main class that implements the **com.altio.client.control.Control** interface. The class **com.altio.client.control.BaseControlImpl** provides default implementations for the methods in the Control interface whilst the class **com.altio.client.control.SwingControlImpl** provides a default implementation of the methods in the Control interface for a Swing based control so extending one of these classes is the simplest way of creating a custom control that uses a Java Swing component.

Practical Exercise: Swing Label Control

This exercise introduces you to creating a custom control, further examples which also show how events are handled can be found in the controls/samples directory of the Altio application.

File Directories

To do this tutorial you will need to create some directories to place our created files in.

1. Create a directory, for example called **altiotutorial** in **C:** (e.g. **c:\altiotutorial**).
2. In your **altiotutorial** directory, create another directory called **controls**.
3. Create further directories until you have the following directory structure:
c:\altiotutorial\controls\com\mycontrols\swing.

Adding a Constructor

The custom control must contain a constructor that accepts **Window** and **XmlElement** arguments. **Window** is the Altio window which contains the controls. **ConfigElem** is an XML element that describes the instance of the control; the element may contain attributes and child elements with values specific to the instance.

As no control properties have been defined the control only needs to construct the **BaseControlImpl** super class, which sets up basic properties including position and size of the control.

1. Create a java file called **SimpleSwingLabel** in which to put your code in.

2. Save it in: `c:\altiotutorial\controls\com\mycontrols\swing\SimpleSwingLabel.java`

3. In the file enter:

```
package com.mycontrols.swing;
import com.altio.xml.XmlElement;
import com.altio.client.control.BaseControlImpl;
import com.altio.client.Window;
/**
 * swing control wrapper
 *
 */
public class SimpleSwingLabel extends BaseControlImpl {
    /** Constructor
     */
    public SimpleSwingLabel(Window window, XmlElement configElem) {
        super(window, configElem, null);
    }
}
```

This is all that is needed to run the control, even though it performs no useful purpose.

Generating a Control Descriptor

To use a control in AltioLive we need to create a control archive descriptor. The descriptor specifies a set of controls, their properties and events (triggers) they can handle, any images they use and the classes and jars needed to use them.

1. Create a minimal descriptor called **controls.xml** that just references our new class.
2. Save it in: **C:\altiotutorial\controls\controls.xml**.
3. In the file enter:

```
<ALTIO>

  <ID>com.mycontrols.swing</ID>
  <DESC>My Altio custom controls</DESC>
  <CONTROLS>
    <CONTROL CLASS='com.mycontrols.swing.SimpleSwingLabel'>
      <NAME>Simple Swing Label</NAME>
      <DESC>Swing Controls</DESC>
      <VERSION>1.0</VERSION>
      <HELPPURL></HELPPURL>
      <DEFAULT>W='60' H='30'</DEFAULT>
      <PROPERTIES>
        <PROPERTY INHERIT="BASIC_CONTROL" />
```

```
        </PROPERTIES>
        <TRIGGERS/>
    </CONTROL>
</CONTROLS>
<CLASSFILES>
    <CLASSFILE NM='com/mycontrols/swing/SimpleSwingLabel.class' />
</CLASSFILES>
</ALTIO>
```

Compiling and Deploying the Control

1. Compile the java using your preferred tool or using javac. Make sure to include the Altio client jar on your classpath e.g.

```
>javac -classpath ".;C:\Program
Files\AltioLiveStudio5.4\tomcat\webapps\altio54\client\clientRelease.jar"
com\mycontrols\swing\SimpleSwingLabel.java
```

2. Now create a deployment file by archiving the control and descriptor into a jar file.

```
>jar cf mycontrols.jar *
```

3. Copy the control archive to the AltioLive **controls** directory to deploy it. On a windows system this will be something like:

C:\Program Files\AltioLiveStudio5.4\tomcat\webapps\altio54\WEB-INF\classes\controls

4. On the **AltioLive Studio Explorer** window, right-click on the **Custom Controls** node.
5. Select **Install Archives**.

A new node should be added to the Custom Controls node with the title of your control.

While we now have a control which is available within Altio, it still does nothing, to have a functional control we now need to add the swing **JLabel**.

Creating and displaying the JLabel

Creating and displaying a Swing component is made easier from Altio 5.0 as most of the default functionality for a Swing based control are provided in the **SwingControlImpl** class. Displaying our JLabel is as simple as changing our class to extend **SwingControlImpl**, creating an instance of a JLabel and calling the **setRealControl** method with that instance as the parameter.

1. Open the **SimpleSwingLabel.java** file you have created.
2. Enter:

```
package com.mycontrols.swing;

import com.altio.xml.XmlElement;
import com.altio.client.control.SwingControlImpl;
```

```
import com.altio.client.Window;
import javax.swing.JLabel;

/**
 * swing control wrapper
 *
 */

public class SimpleSwingLabel extends SwingControlImpl {

    private JLabel label;

    /** Constructor
     */
    public SimpleSwingLabel(Window window, XmlElement configElem) {
        super(window, configElem, null);

        label = new JLabel("Hello World");
        setRealControl(label);
    }
}
```

```
        }  
    }
```

You now have a working label control which uses the caption property for the label text. As this is a swing control it is possible to display text formatted using HTML tags however it won't be visible in the **Designer** tool but it will appear when you run a regular Altio 5.4 application. Recompile and re-deploy the control, you will need to restart your browsers to be able to use the control.

This is the end of the exercise for the custom control. The rest of the sections provide examples of the other features associated with creating a custom control.

Example of other features associated with Custom Controls

Properties

Run Time Properties

As with standard AltioLive controls, a custom control can have runtime properties that can be set through the event action **Set property**.

Below is an example of a **setProperty** method, which sets the text displayed in a Swing Label control:

```
public void setProperty(String propertyName,
                        String propertyValue,
                        Hashtable lookup,
                        String componentName) {
    if (propertyName.equalsIgnoreCase("VALUE") ||
        propertyName.equalsIgnoreCase("CAPTION")) {
        setValue(propertyValue);
        label.setText(propertyValue);
    }
    else {
        super.setProperty(propertyName, propertyValue,
```

```
        lookup, componentName);  
    }  
}
```

Design Time Properties

Custom controls can have properties set at design time by publishing the property in the deployment descriptor for the control. The properties are displayed and set through the **Designer** tool as if the control was an AltioLive control.

The example below shows a **controls.xml** control deployment descriptor for our Swing Label control. The descriptor has the control inheriting from the **BASIC_CONTROL** properties and adds a new property for the control.

```
<ALTIO>  
  
  <ID>com.mycontrols.swing</ID>  
  
  <DESC>My Altio custom controls</DESC>  
  
  <CONTROLS>  
  
    <CONTROL CLASS='com.mycontrols.swing.SimpleSwingLabel'>  
  
      <NAME>Simple Swing Label</NAME>  
  
      <DESC>Swing Controls</DESC>  
  
      <VERSION>1.0</VERSION>  
  
      <HELPURL></HELPURL>  
  
      <DEFAULT>W='60' H='30'</DEFAULT>  
  
      <PROPERTIES>
```

```
        <PROPERTY INHERIT="BASIC_CONTROL" />
        <PROPERTY NM="LABEL_TEXT" ATTRTYPE="General" CAPTION="Label Text"
CTRLTYPE="TEXT" GET="LabelText" SET="Y" />
    </PROPERTIES>
    <TRIGGERS/>
</CONTROL>
</CONTROLS>
<CLASSFILES>
    <CLASSFILE NM='com/mycontrols/swing/SimpleSwingLabel.class' />
</CLASSFILES>
</ALTIO>
```

Each **PROPERTY** element specifies the behavior of a particular control property. The attributes we have used are:

- **NM:** specifies the name of the property. This is used as the name of the attribute to store the property value in XML
- **GET:** specifies the name that can be used to get the property. For example: for the **LabelText** property above, we can now use **SimpleSwingLabel1.LabelText** inside an expression.
- **SET:** specifies whether this property can be set at runtime. Its value is either **N** or **Y**.

- **DESIGN:** specifies whether the property is shown in the design-time property list for the control. Its value is either **Y** or **N** and defaults to **Y**.
- **ATTRTYPE:** specifies how the property is to be grouped in the Designer property list. We have set all our properties to use the **General** group.
- **CAPTION:** specifies the display name to use for the property in the Designer property list
- **CTRLTYPE:** specifies that a text control will be used for editing the property value when it is selected in the property list.
- **CTRLMASK:** specifies what sort of values can be entered for the property value.

Please note: For more property options see the online help topic **Property Style Editing**.

For the custom control to use the Design time properties the **configure** method needs to be edited to get the property values when the control is generated. The design time properties are stored within the control details of the application view file.

Example constructor using design time properties:

```
public void configure(XmlElement configElem,
                    XmlElement preferences){
    super.configure(configElem, preferences);
    String text= configElem.getAttribute("LABEL_TEXT");
    label.setText(text);
}
```

Inherited Properties

It is not necessary to add every property available in your custom control to the deployment descriptor as there is a **INHERIT** attribute available in the **PROPERTY** element.

```
<PROPERTY INHERIT="BASIC_CONTROL" />
```

The definition of **BASIC_CONTROL** is held in the view for the designer, **altiodesigner.xml**. If there are properties in **BASIC_CONTROL** that we don't want to inherit then these can be overridden.

In the example below we remove the caption property from the list of available properties for the control.

```
<PROPERTY NM="CAPTION" OVERRIDE="REMOVE" />
```

Making Properties Available to Other Controls

To allow other controls to access the properties of a custom control override the method **getProperty**. If this is not done then expressions may not resolve correctly, for example, `#{SimpleSwingLabel1.Label_Text}` will do nothing.

Below is an example **getProperty** method for our **SimpleSwingLabel**:

```
public Object getProperty(String propertyName, String[] componentNames,  
Object[] arguments) {  
    if (propertyName.equalsIgnoreCase("label_text")) {  
        return label.getText();  
    } else {
```

```
        return super.getProperty(propertyName, componentNames,  
        arguments);  
    } }
```

Triggers and Events

Mouse Events

The base class will handle Swing mouse events and trigger corresponding Altio mouse events for you. You may have Swing components additional to the component you have set as the “real control” (for example a collection of buttons on a panel). If you would like to trigger Altio events on these controls you will need to call the **SwingControlImpl** **enrichSubcomponentmethod** for each component using that component as the parameter.

For example:

```
// Panel is a JPanel and the buttons are JButtons  
  
panel.add(button1);  
  
panel.add(button2);  
  
panel.add(button3);  
  
setRealControl(panel);
```

```
enrichSubcomponent(button1);
```

```
enrichSubcomponent(button2);
```

```
enrichSubcomponent(button3);
```

Triggers

To allow someone to fire an action against the control triggers need to be added to the control deployment descriptor.

```
<TRIGGERS>
<TRIGGER VALUE="ONCHANGE" />
</TRIGGER>
```

To fire the trigger and cause any associated actions to take place a call to **ActionRule.performActions** is required. In most cases this call will be instigated by a mouse event for example: **onMouseClicked**.

ActionRule.performActionRules(this, ActionRule.TRIGGER_ONCHANGE);

Triggers are not limited to the standard events used by Altio controls. If for example you want to trigger actions when a certain value is assigned to the custom control then you can add your own trigger.

```
<TRIGGERS>
...
<TRIGGER VALUE="ILLEGAL_TEXT">Illegal Text</TRIGGER>
```

```
...  
</TRIGGER>
```

And trigger any actions defined for this event by adding a call to **PerformActionRules** at the appropriate point in your code:

```
ActionRule.performActionRules(this, "ILLEGAL_TEXT");
```

Live Data

A custom control can be bound to data so that any changes in the data are automatically shown in the control. To achieve this, the custom control will require properties for the data source and data field. The following **PROPERTY** definition will provide the two required properties as if the control were an AltioLive control;

```
<PROPERTY ATTRTYPE="General" BUTTON="ELLIPSIS" CAPTION="Data source" PROPTYPE="DATABUILDER"  
CTRLTYPE="TEXT" NM="DATA" SET="Y" VALIDATE="Y" />  
<PROPERTY ATTRTYPE="General" BUTTON="ELLIPSIS" CAPTION="Data field" COMBO="Y" CTRLTYPE="SELECT"  
DATAPROP="DATA" LISTDATA="/DATATYPES//DATATYPE[@NM=${rowelement}]/@datatype[1]/ATTRIBUTE"  
LISTDATAFLD="@NM" NM="DATAFLD" PROPTYPE="DATAFLD" SET="Y" VALIDATE="Y">  
<OPTION/>  
</PROPERTY>
```

By default **SwingDataControlImpl** reads these properties and binds a dataset for you. So you need to change your class to extend **SwingDataControlImpl** instead of **SwingControlImpl**.

The **DataBinder** object will notify your control when the contents of the bound dataset change via **onDataUpdate()** of the **DataSetListener** interface. As **SwingDataControlImpl** already implements **DataSetListener**, all that is required is to override the listener's **onDataUpdate** method.

```
public void onDataUpdate(DataUpdateEvent event) {  
    super.onDataUpdate(event);  
    label.setText(getDataValue());  
}
```

Please note:

- If you wish to support any of the data change event triggers (**DATAADD**, **DATAUPDATE**, **DATADELETE**, **DATACHANGE**), these events will be automatically called by the **DataBinder**.
- If you override **SwingDataControlImpl's destroy()**, you must call **super.destroy()** to ensure that any registered datasets are cleared.
- When your control is constructed, the sequence of method calls is:
 1. constructor and **configure()**,
 2. **refreshData()** – which initializes your **DataBinder** and calls back to,
 3. **onDataUpdate()** – your datasets are read,
 4. **initializeData()** – to enable you to do any data specific initialization for your control.

- Once constructed you will get `onDataUpdate()` calls when your datasets change.

Control Dependencies

If other controls will be dependent upon your control, then you need to add some code at the point where you want the dependent control(s) to refresh.

```
// if you are triggering an ONCHANGE event, this automatically
```

```
// invokes the refreshDependent code
```

```
ActionRule.performActionRules(this, ActionRule.TRIGGER_ONCHANGE);
```

```
// or if not, you can call direct to the refreshDependentControls method
```

```
// refreshDependentControls();
```

Defining Default Values

Default value can be assigned to a custom control through the deployment descriptor the values being read in the constructor of the control as an attribute. The `DEFAULT` element is added under the `CONTROL` element of the deployment descriptor.

```
<DEFAULT>W= ' 60 ' H= ' 30 ' </DEFAULT>
```

Document Information

Integra SP – Altio

Telephone: +44 (0) 20 8528 1045

Internet: www.altio.com

Copyright © 2010 Integra SP

Copyright in this document is vested in Integra SP. The contents of the document (wholly or in part) must not be reproduced, distributed, used or disclosed without the prior written permission of Integra SP.

Integra recognizes the trademarks or registered trademarks of any third party product or company name referenced in this document at the time of its publication.