Composing New Controls Out of Existing Controls

Composing New Controls Out of Existing Controls

This chapter covers the following topics:

- Concept
- Programmed Macro Control - Example
- Configured Macro Control - Example

Concept

The concept is quite simple: you provide a macro control that tells how a given XML tag is transferred into an XML string representing the internally used Application Designer controls.

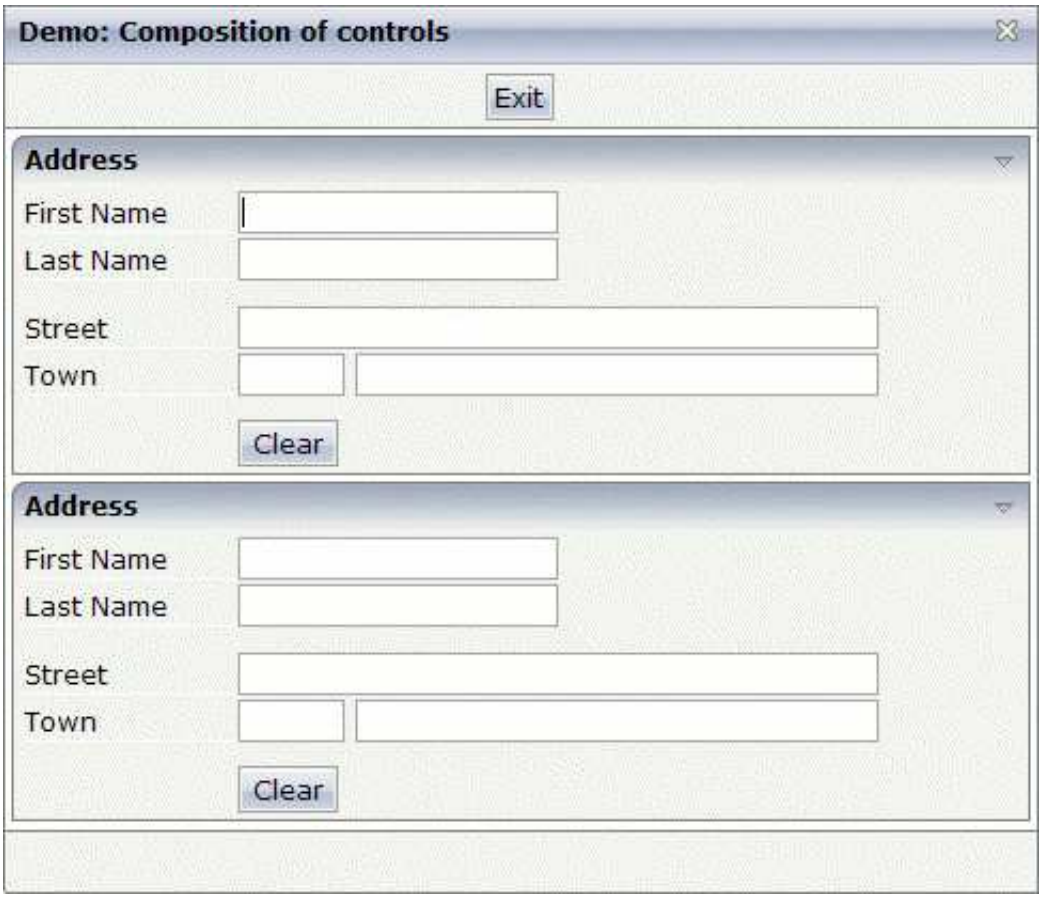
There are two ways to provide a macro control:

- Either you program a control on your own,
- or you configure the control using an XML definition.

The first way is the most flexible way - you create a piece of code translating the macro XML tag into other controls' XML tags. The second way is the easier way by which you can use a certain XML definition to define macro controls without having to code at all.

Programmed Macro Control - Example

Let us have a look at the following page:



This page contains two address areas. Now let us look at the corresponding XML layout definition:

```
<page model="com.softwareag.cis.test14.ControlLibraryControlCompositionAdapter">
    <titlebar name="Demo: Composition of controls">
    </titlebar>
    <header withdistance="false">
        <button name="Exit" method="endProcess">
        </button>
    </header>
    <pagebody>
        <demo:addressrowarea2 addressprop="address">
        </demo:addressrowarea2>
        <demo:addressrowarea2 addressprop="addressWife">
        </demo:addressrowarea2>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

You see that there is a control `demo:addressrowarea2` which is used two times - once per address area. The control is responsible for arranging all its inner controls. Each tag has an `addressprop` property - we will see later how this property is treated.

Control Handler Code

Let us have a look at the corresponding control handler code:

```
package com.softwareag.cis.demolibrary;

import org.xml.sax.AttributeList;

import com.softwareag.cis.gui.generate.IMacroTagHandler;
import com.softwareag.cis.gui.protocol.Message;
import com.softwareag.cis.gui.protocol.ProtocolItem;

public class ADDRESSROWAREA2Handler
    implements IMacroTagHandler
{
    // ------------------------------------------------------------------------
    // members
    // ------------------------------------------------------------------------

    String m_addressprop;

    // ------------------------------------------------------------------------
    // public usage
    // ------------------------------------------------------------------------

    /** */
    public void generateXMLForStartTag(String tagName,
                                       AttributeList attrlist,
                                       StringBuffer sb,
                                       ProtocolItem pi)
    {
        readAttributes(attrlist);
        fillProtocol(pi);
        // build XML that consists out of contained controls
        sb.append("<rowarea name='Address'>");
        sb.append(  "<itr>");
        sb.append(    "<label name='First Name' width='100'/>");
        sb.append(    "<field valueprop='"+m_addressprop+".firstName' width='150'/>");
        sb.append(  "</itr>");
        sb.append(  "<itr>");
        sb.append(    "<label name='Last Name' width='100'/>");
        sb.append(    "<field valueprop='"+m_addressprop+".lastName' width='150'/>");
        sb.append(  "</itr>");
        sb.append(  "<vdist height='10'/>");
        sb.append(  "<itr>");
        sb.append(    "<label name='Street' width='100'/>");
        sb.append(    "<field valueprop='"+m_addressprop+".street' width='300'/>");
        sb.append(  "</itr>");
        sb.append(  "<itr>");
        sb.append(    "<label name='Town' width='100'/>");
        sb.append(    "<field valueprop='"+m_addressprop+".zipCode' width='50'/>");
        sb.append(    "<hdist width='5'/>");
        sb.append(    "<field valueprop='"+m_addressprop+".town' width='245'/>");
        sb.append(  "</itr>");
        sb.append(  "<vdist height='10'/>");
        sb.append(  "<itr>");
        sb.append(    "<hdist width='100'/>");
        sb.append(    "<button name='Clear' method='"+m_addressprop+".clearAddress'/>");
        sb.append(  "</itr>");
        sb.append("</rowarea>");
    }

    /** */
    public void generateXMlForEndTag(String tagName,
                                     StringBuffer sb)
    {
```

```
    }

    // ------------------------------------------------------------------------
    // private usage
    // ------------------------------------------------------------------------

    /** */
    private void readAttributes(AttributeList attrlist)
    {
        for (int i=0; i<attrlist.getLength(); i++)
        {
            if (attrlist.getName(i).equals("addressprop"))
                m_addressprop = attrlist.getValue(i);
        }
    }

    /** */
    private void fillProtocol(ProtocolItem pi)
    {
        // check
        if (m_addressprop == null)
            pi.addMessage(new Message(Message.TYPE_ERROR,"Attribute ADDRESSPROP is not set"));
        // properties
        pi.addProperty(m_addressprop,"ADDRESSInfo");
        // no further properties to be proposed in code assistant
        pi.suppressFurtherCodegenEntries();
    }

}
```

Let us have a look at the building blocks of the code:

- The class has a member `m_addressprop`. This member is filled directly at the beginning of the `generateHTMLForStartTag` method - inside the `readAttributes()` method. The attribute list is walked through and checked for the attribute `addressprop`.

- As the next step, the protocol item is filled. On the one hand, you can put there any information with a certain `severity` attribute - in the example, an error message is written to protocol if no attribute `addressprop` is defined. On the other hand, you have to tell the protocol item which properties you are accessing from your control.

  Pay attention that the properties which are accessed inside the access control are a composition out of the `m_address` value and some fix names which are defined by the control.

- In the processing of the method `generateXMLForStartTag()`, all the XML is created that per control instance creates the container representing an address area.

Control Adapter Code

The control adapter is not required to be written for a control - it is just an option which is extremely useful for structuring you server side code.

In principle, the control definition says that it refers to an address property (`ADDRESSPROP` value). The inner controls take their information out of sub-properties of this control. For example, if the `ADDRESSPROP` value is defined to be "wifeAddress", then the fields and buttons are bound to:

- `wifeAddress.firstName`

- `wifeAddress.lastName`
- `wifeAddress.street`
- `wifeAddress.zipCode`
- `wifeAddress.town`
- `wifeAddress.clearAddress` (method)

You now can provide for a server side control adapter class which provides for all this data. For example, the implementation is:

```
package com.softwareag.cis.demolibrary;

import com.softwareag.cis.server.*;

/**
 * This is the logic-class behind the control ADDRESSROWAREA.
 */
public class ADDRESSInfo
{
    // -------------------------------------------------------------------------
    // members
    // -------------------------------------------------------------------------

    String m_firstName;
    String m_lastName;
    String m_street;
    String m_zipCode;
    String m_town;

    // -------------------------------------------------------------------------
    // public access
    // -------------------------------------------------------------------------

        public String getFirstName() { return m_firstName; }
        public String getLastName() { return m_lastName; }

        public String getStreet() { return m_street; }
        public String getTown() { return m_town; }

        public String getZipCode() { return m_zipCode; }
        public void setFirstName(String firstName) { m_firstName = firstName; }

        public void setLastName(String lastName) { m_lastName = lastName; }
        public void setStreet(String street) { m_street = street; }

        public void setTown(String town) { m_town = town; }
        public void setZipCode(String zipCode) { m_zipCode = zipCode; }

        public void clearAddress()
       {
        m_firstName = null;
        m_lastName = null;
        m_street = null;
        m_zipCode = null;
        m_town = null;
    }
}
```

A page adapter class can now use this control adapter class and can automatically take over all its contained properties and methods. For example, the page adapter of the page of this example might look like:

```
package com.softwareag.cis.test14;

import com.softwareag.cis.demolibrary.*;
import com.softwareag.cis.server.Model;

public class ControlLibraryControlCompositionAdapter
    extends Model
{
    // ------------------------------------------------------------------------
    // members
    // ------------------------------------------------------------------------

    ADDRESSInfo m_address = new ADDRESSInfo();
    ADDRESSInfo m_addressWife = new ADDRESSInfo();

    // ------------------------------------------------------------------------
    // public access
    // ------------------------------------------------------------------------

    public ADDRESSInfo getAddress() { return m_address; }
    public ADDRESSInfo getAddressWife() { return m_addressWife; }
}
```

The page adapter just creates two instances of the control adapter `ADDRESSInfo` and publishes them as property `address` and property `wifeAddress`.

Be aware that you can use all Java possibilities on the server side to let the control adapter interact with your page adapter. Maybe you would like to be informed inside the page adapter every time the `clear()` method is invoked? Then just build some eventing functions into the control adapter - and the page adapter can register as event listener to its contained control adapter.

Configured Macro Control - Example

In the previous example, an explicit control handler class was written in order to transfer a short XML statement into a long one. For simple control arrangements without any sophisticated logic, you can do the same by just configuring the control - instead of programming it.

Let us now do the same as done with code in the previous section - this time without coding.

The configuration is done using an editor extension file (e.g. *editor_demo.xml* in the *cis/config* directory). When generating HTML pages, Application Designer looks into its configuration directory and searches for all *.xml* files starting with "editor_". Each of the files contains configuration information about controls and their usage.

Have a look at the *editor_demo.xml* file and you will see the following section:

```
<!-- DEMO:ADDRESSROWAREA3 -->
  <tag name="demo:addressrowarea3">
    <attribute name="addressprop" mandatory="true"/>
    <taginstance>
         <rowarea name="Address">
           <itr>
             <label name="First Name" width="100"/>
```

```
          <field valueprop="$addressprop$.firstName" width="150"/>
        </itr>
        <itr>
          <label name="Last Name" width="100"/>
          <field valueprop="$addressprop$.lastName" width="150"/>
        </itr>
        <vdist height="10"/>
        <itr>
          <label name="Street" width="100"/>
          <field valueprop="$addressprop$.street" width="300"/>
        </itr>
        <itr>
          <label name="Town" width="100"/>
          <field valueprop="$addressprop$.zipCode" width="50"/>
          <hdist width="5"/>
          <field valueprop="$addressprop$.town" width="245"/>
        </itr>
        <vdist height="10"/>
        <itr>
          <hdist width="100"/>
          <button name="Clear" method="$addressprop$.clearAddress"/>
         </itr>
       </rowarea>
  </taginstance>
  <protocolitem>
      <addproperty name="$addressprop$" datatype="ADDRESSInfo" useincodegenerator="true"/>
  </protocolitem>
</tag>
<tagsubnodeextension control="pagebody" newsubnode="demo:addressrowarea3"/>
```

The following is defined in this section:

- The new tag ADDRESSROWAREA3 is defined.
- A property `addressprop` is defined to exist.
- The XML macro is contained that is used for transferring the control into XML. Inside the XML you see that the value of `addressprop` is referred to by using "$addressprop$".
- The new tag is defined to be reachable below the PAGEBODY tag.

The result at the end is the same as produced with the ADDRESSROWAREA2 control of the previous section.

The XML configuration can be either done manually within the XML file or by using the Control Editor.

*editor_** File Concept

In the example above, the macro XML definition was part of a file *editor_demo.xml*. If you have a look at the */cis/config* directory of your web application, then you will see some files:

- *editor.xml*
- *editor_demo.xml*

  editor_report.xml

editor_pivot.xml

and other *editor_** files

- *editorextensions_template.xml*

Each file contains information about controls. When gathering the available controls, Application Designer reads all *editor_*.xml* files and builds one "big internal" control model.

editor_*.xml files are also mentioned later. Since they hold information on how to arrange controls, they are also used as control files for Application Designer’s Layout Painter. For more details, see the section *Bringing Controls into the Layout Painter*.

Note:
If you are using an old servlet engine of version 2.2 (e.g. Tomcat 3, Websphere 4), there is one additional file to be maintained: *editorextensions.xml*. For detailed information, see the *editorextensions_template.xml* file.