Java Bean Property Binding

This chapter covers the following topics:

- Class Binding
- Method Binding
- Property Binding
- Access Path Restrictions

Class Binding

The page binding is defined in the PAGE tag of your page definition. The PAGE tag points to a class supporting the interface `com.softwareag.cis.server.IModel`. There is a class `com.softwareag.cis.server.Adapter` which implements this interface - which should be used to build adapter classes as subclasses of `Adapter`.

Example:

```
<page model="com.softwareag.cis.demo.DemoAdapter" ...>
  ...
  ...
  ...
</page>
```

The above definition points to a class which looks as follows:

```
package com.softwareag.cis.demo;

import com.softwareag.cis.server.*;

public class DemoAdapter
    extends Adapter
{
    // constructor - either no constructor or a constructor
    // without any parameters
    public DemoAdapter()
    {
    }
    ...
    ...
}
```

Note that the adapter class has at least a default constructor (without any parameters).

Method Binding

Controls triggering a method inside the adapter are bound to a method name of the adapter. The method implementation itself must be a method without any parameters.

Example:

```
<button name="Save" method="doSave" ...>
</button>
```

The above button definition points to a method inside the adapter class which looks as follows:

```
public void doSave()
{
    ...
    ...
}
```

Property Binding

Controls presenting or manipulating data of the adapter are bound to properties of the adapter. There is a flexible concept available that makes it possible for you to use the following:

- Simple Properties which are Provided Directly by the Adapter
- Simple Properties which are Provided by Embedded Objects of the Adapter
- Array Properties which are Provided Directly by the Adapter
- Array Properties which are Provided by Embedded Objects of the Adapter

Simple Properties which are Provided Directly by the Adapter

This is the easiest way of binding: the property name which you specify in the definition of the control is provided directly by the adapter object - by a corresponding set and get method. It depends on the control whether you have to provide both set and get methods or just one of them.

Following the Java Bean conventions, the first character of the property name is written as a capital letter inside the corresponding set or get method.

The get method must return a value which is either a simple data type or a "simple" object. A list of supported return values is shown in *Appendix C - Data Types to be Used by Adapter Properties*. The set method must offer one parameter to update its value at runtime. The parameter type must either be a simple data type or one of the classes that are listed in appendix C.

Example:

```
<field valueprop="name" ...></field>
<field valueprop="age" ...></field>
<field valueprop="weight" ...></field>
<field valueprop="birthday" ...></field>
```

The above field definitions are bound to the following set/get methods:

```
public void setName(String value) { ... }
public String getName() { ... }

public void setAge(int value) { ... }
public String getAge() { ... }

public void setWeight(float value) { ... }
public float getWeight() { ... }

public void setBirthday(Cdate value) { ... }
public Cdate getBirthday() { ... }
```

The correct property name starts with a lowercase letter, because the first letter is always converted to lowercase. Example:

```
<field valueprop="cAPITAL" ...></field>
```

The above field definition.is bound to the following set/get method:

```
public void setCAPITAL(String value) { ... }
public String getCAPITAL() { ... }
```

Simple Properties which are Provided by Embedded Objects of the Adapter

Properties can also be provided by an embedded object of the adapter. The embedded object itself must be accessible by a corresponding get method.

Example:

```
<field valueprop="address.street"></field>
```

The above field definition points to a value which is provided in the following way:

```
public class XYZAdapter
    extends com.softwareag.cis.server.Adapter
{
    // access in the adapter to the address object
    public Address getAddress() { ... }
}

public class Address
{
    public String getStreet() { ... }
    public void setStreet(String value) { ... }
}
```

You can build any chaining of properties you desire.

As shown in the example, embedded objects need not be adapter objects. Only the root object is required to be an adapter.

Array Properties which are Provided Directly by the Adapter

You can use array properties and can access them directly within your binding definitions. An array property always returns an array of objects, each object providing either simple properties or array properties. The type of the object array is not relevant for the Application Designer runtime. If you just return "Object[]" as a result of the method, this is sufficient.

Example:

```
<field valueprop="addresses[0].street" ...></field>
```

The above field definition points to a property which is implemented in the following way:

```
public class XYZAdapter
    extends com.softwareag.cis.server.Adapter
{
    // access in the adapter to the address object
    public Address[] getAddresses() { ... }
}

public class Address
{
    public String getStreet() { ... }
    public void setStreet(String value) { ... }
}
```

Note that the name used inside the control definition for binding (`addresses[0].street` in the our example) can either be entered manually or is implicitly created by some controls. Example: in a TEXTGRID control, specify an array property for the entire control and a simple property inside the COLUMN definition:

```
<textgrid arrayprop="addresses" ...>
    <column property="street" ...></column>
    <column property="city" ...></column>
</textgrid>
```

The TEXTGRID control itself uses these definitions to ask for the properties `addresses[0].street`, `addresses[0].city`, `addresses[1].street`, `addresses[1].city` etc. at runtime.

Note that it is not possible to access an array of simple objects directly. It is not possible to define a field in the following way

```
<field valueprop="streets[0]"></field>
```

having a method:

```
public String[] getStreets() { ... }
```

You always have to go through an array of objects where each element itself provides access to simple properties.

Array Properties which are Provided by Embedded Objects of the Adapter

You can use any combination of *Simple Properties which are Provided by Embedded Objects of the Adapter* and *Array Properties which are Provided Directly by the Adapter*.

Example: define access to array properties in the following way:

```
<field valueprop="person.addresses[0].street" ...></field>

<textgrid arrayprop="person.addresses" ...>
    <column property="street" ...></column>
    <column property="city" ...></column>
</textgrid>
```

Access Path Restrictions

At runtime, Application Designer transfers the data from the adapter to the client. For accessing the data, it uses the following strategy:

- It asks the adapter object for all properties. This means, it calls all get methods that are defined as public methods.
- If the get method returns a simple value, is marked to be transferred. (Whether it is really transferred, depends also on the delta management between the client and the server.)
- If the get method returns an object (e.g. an address object as used in the previous sections) or an array of objects, these objects are used for further drill down.

This mechanism is flexible on the one side, but dangerous on the other side: the Application Designer runtime will load *all objects* by following up the get methods.

Consequently, there is a certain access path restriction inside the Application Designer environment: if you generate a page (either by the Layout Painter or by the logical interfaces to the HTML generator) an access path restriction file is generated in addition. The HTML generator parses all tags of a page; the controls themselves are bound to properties. This information is collected and written to a file.

This file is stored in the directory */accesspath* below the project directory. Please have a look at the files generated implicitly with your pages: the file contains a list of all access paths that are valid to be followed by runtime.

The name of the access path file is the same as the name of the page, but has the extension *.access*. Be aware of the fact that this access path file is inevitably important to avoid "mass loading" of data. Therefore, it must be a part of your software deployment.