Definition of dynamic models using XML descriptors

Definition of dynamic models using XML descriptors

Introduction

It is possible to define the underlying data model without having to write java code. To do this, you must use an XML descriptor which describes the DataNodes and their relationships. An skeleton XML descriptor has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<zkib-model>
    <datanode name="my-data">
        <finder name="title" type="title">
        </finder>
        <finder name="country" type="country">
            ...
        </finder>
    </datanode>
    <datanode name="title"/>
    <datanode name="country">
        <finder name="city" type="city">
            ...
        </finder>
    </datanode>

    <datanode name="city">
            ...
    </datanode>
</zkib-model>
```

Within the XML tag whose root is always zk-ib, you can specify one or more DataNodes. Each DataNode has a unique name. Within each DataNode, you can define multiple finders. Each finder specifies a name and a type. The name will be used to build xpaths, whlie the type identifies the type of DataNode this xpaths refers to.

Within each finder you can define multiple search handlers. They will be responsible for retrieving data from persistent storage, just like the find method on the finder interface. Additionally, you can define one or new instance handlers. They will be responsible for creating new business objects on user request.

Finally, each DataNode can have many persistence handlers. They will act just lik the doInsert, doUpdate and doDelete methods on DataNode class. Each type of handler can be executed conditionally, depending on expressions to be evaluated at run time. These expressions can use the following predefined variables:

Additionally, EL expressions may refer to all variables defined within the DataSource. Those variables are accessed via JXPathContext.getVariables() method. To use of this type of data models, simply create a datamodel component on the ZUL page and assign the src attribute the path to the XML descriptor. The path can be a web component or a class path resource.

Variable	Value
self	Current DataNode
instance	Business object wrapped into current DataNode
parent	Parent DataNode
parent.instance	Business object wrapped into parent DataNode
datasource	DataSoruce the current DataNode belongs to

Ejb-handler Handler

It is responsible for persisting the object via a stateless session bean. The following attributes are supported:

Attribute	Usage
jndi	JNDI path to EJB Home interface
if	EL expression that must be evaluated to true prior to handler action
unless	EL expression that must be evaluated to false prior to handler action

Inside the handler, you should specify the suitable insert-method, delete-method and update-method tags. There is a mandatory attribute named method. This attribute must contain the name of the method to invoke. Additionally, the parameters to use can be specified. The following example shows an ejb-handler that uses an EJB whose only parameter is the data object:

```
<datanode name="network">
    <ejb-handler jndi="com.soffid.sample/NetworksBean">
        <insert-method method="insert"/>
        <delete-method method="delete"/>
        <update-method method="update"/>
    </ejb-handler>
        ...
</datanode>
```

This second example shows how to call a method with slightly complex parameters:

```
<datanode name="acl">
    <ejb-handler jndi="com.soffid.sample/NetworksBean" >
        <insert-method method="grant">
            <parameter value="${parent.instance}"/>
            <parameter value="${instance}"/>
        </insert-method>
        <delete-method methos="revoke">
            <parameter value="${instance}"/>
        </delete-method>
    </ejb-handler>
```

```
</datanode>
```

Script-handler Handler

It ca be used to persist the business objects using BSH scripts. Supports the following attributes:

Attribute	Usage
if	EL expression that must be evaluated to true prior to handler action
unless	EL expression that must be evaluated to false prior to handler action

Inside the handler, you can use the insert-script, delete-script and update-script tags. Each contains the BSH script that the engine will execute to perform inserts, deletes or updates. Within the BSH script you can refer to the same EL expressions predefined variables:

Variable	Value
self	Current DataNode
instance	Business object wrapped into current DataNode
parent	Parent DataNode
parent.instance	Business object wrapped into parent DataNode
datasource	DataSoruce the current DataNode belongs to

The following example shows how to save objects in file Country:

```
<datanode name="country">
    <script-handler >
        <insert-script>
            import java.io.*;
            f = new FileOutputStream ("country."+instance.abbreviation);
            oos = new ObjectOutputStream (f);
            oos.writeObject (instance);
            oos.close ();
            f.close ();
        </insert-script>
        <update-script>
            import java.io.*;
            f = new FileOutputStream ("country."+instance.abbreviation);
            oos = new ObjectOutputStream (f);
            oos. writeObject (instance);
            oos. close ();
            f. close ();
        </update-script>
        <insert-script>
            import java. io. *;
            f = new File ("country. "+instance. abbreviation);
            f. delete ();
        </insert-script>
    </script-handler>
        ...
</datanode>
```

Collection-handler Handler

This handler is applicable when the persistence of this object is managed by the parent dataNode. The allowed attributes are:

Attribute	Usage
collection	EL expression that identifies the collection onto which the business object must be added or removed
if	EL expression that must be evaluated to true prior to handler action
unless	EL expression that must be evaluated to false prior to handler action

The following example shows how you could manage objects City as a collection of objects within the object Country, which in turn is saved on a disk arhivo:

```
<datanode name="city">
    <collection-handler collection="${parent.instance.cities}">
</datanode>
```

Custom-handler Handler

The custom-handler provides coverage for situations where you need a more sophisticated handler and it is not worth to use a bsh script., In this case the persistence must be done be a java class that implements the PersistenceHandler interface, and the custom-handler specifying the name of the class used. The attributes are:

Attribute	Usage
className	Name of the java class to be used
if	EL expression that must be evaluated to true prior to handler action
unless	EL expression that must be evaluated to false prior to handler action

Data validation

The validation tag is responsible for performing basic checks regarding mandatory attributes and valid attribute values before being submitted to the persistence handler. The validation tag may contain one or more attribute-validation and script-validation tags. The verification will be performed before running the insert or update handler.

attribute-validator contains the following attributes:

Attribute	Usage
expr	EL expression pointing to the attribute to validate
friendlyName	Text to be presented to the user on validation failure. If there is a ZK label with this text, it will be localized based on current user language preference.
notNull	true is the attribute is mandatory
maxLength	Maximum length of the attribute
minValue	Minimum value in case of numeric attributes
maxValue	Maximum value in case of numeric attributes

attribute-script contains a script that will be executed to validate the business object.

Example:

```
<datanode name="country">
    <validation>
        <attribute-validation expr=”${instance.abbrevisation}”
                notNull=”true” friendlyName=”Two letter abbrv.”>
        <script-validation>
            if (instance.abbreviaton.equals(“CT”))
            {
                throw new RuntimeException (“Catalonia is not a country yet”);
            }
        </script-validation>
    </validation>
        ...
</datanode>
```

EJB find handler: ejb-finder

Handles the method to retrieve business objects via a stateless session bean. Supports the following attributes:

Attribute	Usage
jndi	JNDI path to EJB Home interface
method	EJB Bean method to get business objects
if	EL expression that must be evaluated to true prior to handler action
unless	EL expression that must be evaluated to false prior to handler action

Additionally, it can specify one or more parameters in a way similar to the ejb-handler methods

ejb-finder example

```
<datanode name="country">
        ...
    <finder name="cities" type="city" >
        <ejb-finder jndi="com.soffid.sample/CitiesBean" method="findByCountry">
            <parameter value="${instance.abbreviation}"/>
        </ejb-finder>
    </finder>
        ...
</datanode>
```

script-finder handler

It is responsible for retrieve business objects from the persistence layer using BSH scripts. The following attributes are supported

Attribute	Usage
if	EL expression that must be evaluated to true prior to handler action
unless	EL expression that must be evaluated to false prior to handler action

The contained bsh script must return a collection of business objects. If the returned object is not a collection object, the engine will treat the returned object as a singleton.

In the following example, the script finder retrives all countries saved to disk in the previous example:

```
<datanode name="root">
    ...
    <finder name="country" type="country" >
        <script-finder >
            import java.io.*;
            files[] = new File(".").listFiles ( new FilenameFilter () {
                public boolean accept (File dir, String name) {
                    return name.startsWith("coutnry.");
                }
            );
            v = new java.util.Vector();
            for (int i = 0; i < files.length; i++)
            {
                f = new FileInputStream (files[i]);
                ois = new ObjectInputStream (f);
                v.add (ois.readObject ();
                ois.close ();
                f.close();
            }
            return v;
        </script-finder>
    </finder>
    ...
</datanode>
```

collection-finder handler

Similar to collection-handler, this handler retrieves the list objects contained on a parent collection.

Attribute	Usage
collection	EL expression that contains the objects collection
if	EL expression that must be evaluated to true prior to handler action
unless	EL expression that must be evaluated to false prior to handler action

The following example shows how you could retrieve City objects as a collection of objects within the County object:

```
<datanode name="country">
        ...
    <finder name="cities" type="city" >
        <collection-finder collection="${instance.cities}">
    </finder>
        ...
</datanode>
```

custom-finder handler

The custom-finder provides coverage for situations where you need a more sophisticated handler and is not worth implement it using a script. In this case a class that implements the FinderHandler interface must be developed, and the custom-finder specifying the name of the class must be used

Attribute	Usage
className	Name of the FinderHandler class
if	EL expression that must be evaluated to true prior to handler action
unless	EL expression that must be evaluated to false prior to handler action

new-instance-script handler

It is responsible for instantiating new objectswithin a finder on user request .

Attribute	Usage
if	EL expression that must be evaluated to true prior to handler action
unless	EL expression that must be evaluated to false prior to handler action

With the following example the model will alow to create a new city within a country:

```
<datanode name="country">
    ...
    <finder name="cities" type="city" >
        <collection-finder collection="${instance.cities}">
        <new-instance-script>
            c = new City();
            c.countryAbbreviation = instance.abbreviation;
            return c;
        </new-instance-script>
    </finder>
    ...
</datanode>
```

new-instance-bean handler

This handler allows the craetion of a new business object and assign default attribute values. The value of the bean attributes is specified using multiple instances of the bean-attribute tag

Attribute	Usage
className	Name of the business object class
if	EL expression that must be evaluated to true prior to handler action
unless	EL expression that must be evaluated to false prior to handler action
bean-attribute/name	Name of the attribute
bean-attribute/value	EL expression with the value to assign

With the following example would create a new city within a country:

```
<datanode name="country">
    ...
    <finder name="cities" type="city" >
        <collection-finder collection="${instance.cities}">
        <new-instance-bean  className="City">
            <bean-attribute name="countryAbbreviation" value="${instance.abbreviation}"/>
        </new-instance-bean>
    </finder>
    ...
</datanode>
```

custom-attribute handler

Generates virtual attributes derived from other attributes or external elements of the application. It can be applied to any DataNode to add attributes that were not originally present at the underlying business object. Those attributes will be presented at the JXPath interface just as if they were business objects attributes

Attribute	Usage
name	Name of the virtual attribute
expr	EL expression that evaluates de attribute value
if	EL expression that must be evaluated to true prior to handler action
unless	EL expression that must be evaluated to false prior to handler action
depends	XPath to a attribute or business object the expression depends on

It's important to properly set the depends attribute as long as the attribute will be reevaluated whenever a dependent attribute has been changed.

The custom attribute can have an empty EL expressions and use a BeanShell script instead. Here is an example of both aproaches:

```
<datanode name="network">
    <custom-attribute name=”networkMask1” expr=”${instance.network}/${instance.mask}”>
    <custom-attribute name=”networkMask2”>
        <depends>@network</depends>
        <depends>@mask</depends>
        return instance.network + “/” + instance.mask;
    </custom-attribute>
    ...
</datanode>
```

Using dynamic models

To use dynamic models XmlDataSource tag must be replaced by datamodel. The datamodel tag has the following attributes:

Attribute	Usage
id	ZK Identifier
className	root DataNode class name
src	XML resource name for XML dynamic data model
rootNode	Root node type for dynamic data model

className usage is not compatible with src and rootNode attributes.

rootNode attribute is mandatory when using XML dynamic data models.