Class next.wo.client.Association

```
java.lang.Object
   |
   +    next.wo.client.Association
```

public class **Association**
extends Object
implements KeyValueCoding

The Association class is an abstract class that defines the required behavior for objects that know how to get and set the values of applet keys (for state) and communicate that data to the AppletGroupController. These objects are also responsible for detecting actions triggered by their destination applets and sending **invokeAction(**String**)** to have the appropriate action method invoked in the server. Subclasses of Association must implement the KeyValueCoding interface methods **takeValueForKey(**Object, String**)** and **valueForKey**(String**)** as well as the **keys()** method. If you have access to the source code of an applet, consider implementing the SimpleAssociationDestination interface in the applet instead of creating a subclass of Association.

See Also:
KeyValueCoding, SimpleAssociationDestination

Method Index

Methods

o **valueForKey**

```
public abstract Object valueForKey(String key)
```

Because Association explicitly implements the KeyValueCoding interface but is an abstract class, it declares this method as abstract. Subclasses of Association must implement this KeyValueCoding method.

See Also:

KeyValueCoding

o **takeValueForKey**

```
public abstract void takeValueForKey(Object value,
                                     String key)
```

Because Association explicitly implements the KeyValueCoding interface but is an abstract class, it declares this method as abstract. Subclasses of Association must implement this KeyValueCoding method.

See Also:

KeyValueCoding

o **keys**

```
public abstract Vector keys()
```

Subclasses must implement this method to return the list (as a Vector object) of the keys for the state managed by the associated applet.

o **destination**

```
public Object destination()
```

Returns the destination (the applet) of the Association object.

o **invokeAction**

```
public void invokeAction(String action)
```

When the Association determines that an action has been triggered in the destination applet, it should send this message to itself to have the bound action method in the server side component invoked. The argument *action* should be the name of the action managed by the applet. for example:

```
this.invokeAction("selectionChanged");
```

Interface next.wo.client.SimpleAssociationDestination

public interface **SimpleAssociationDestination**
extends Object
extends KeyValueCoding

The SimpleAssociationDestination interface allows applets that implement it to be the destinations of SimpleAssociations. By implementing these methods, plus the KeyValueCoding methods, the applet takes on the responsibilities of the Association object. The SimpleAssociation class simply passes calls to **valueForKey(**String**)**, **takeValueForKey(**Object, String**)**, and **keys()** along to its destination. Associations are required for the exchange of state and action information with the AppletGroupController, which handles communication with the server.

See Also:
Association, KeyValueCoding

Method Index

o **keys**()
Applets must implement this method to return the list (as a Vector object) of the keys for the state that they manage.
o **setAssociation**(Association)
Implemented by applets so that they can store the Association object *assoc* (an instance of SimpleAssociation) so that later, when an action is triggered in the applet, they can send **invokeAction(**String**)** to that object.

Methods

o **keys**

```
public abstract Vector keys()
```

Applets must implement this method to return the list (as a Vector object) of the keys for the state that they manage.

o **setAssociation**

```
public abstract void setAssociation(Association assoc)
```

Implemented by applets so that they can store the Association object *assoc* (an instance of SimpleAssociation) so that later, when an action is triggered in the applet, they can send **invokeAction(**String**)** to that object. This results in the invocation of the associated action method in the server. Note that even applets that do not have actions must implement this method, even if as a "null" method.

See Also:

invokeAction

Interface next.util.KeyValueCoding

public interface **KeyValueCoding**
extends Object

Classes implement the KeyValueCoding interface to return and set the values of obect attributes as identified by a string *key*. In contrast to the Java "put" convention, this interface uses the more Objective C like **takeValueForKey(**Object, String**)**. The value set by the **takeValueForKey(** Object, String**)** method must be a property list type of object. In Java, this must be a String, Vector, Hashtable, or byte[] object; the corresponding Objective C object types are NSString, NSArray, NSDictionary, and NSData. In the client side components feature, Association objects must implement the methods of this interface to set and return the values of their associated applets. Applets can assume the responsibility of an Association object by implementing the SimpleAssociationDestination interface, which includes the methods of the KeyValueCoding interface. Key value coding in client side components must always deal with state keys, never with actions.

See Also:
SimpleAssociationDestination, Association

Method Index

Methods

o **valueForKey**

```
public abstract Object valueForKey(String key)
```

Returns the value (as a property list object) for the applet state or action identified by *key*.

- **takeValueForKey**

```
public abstract void takeValueForKey(Object value,
                                     String key)
```

Sets the value of the applet state or binding identified by *key* to *value*. The *value* must be a property list type of object: String, Vector, Hashtable, or byte[]. Note that you can have nested property lists, like a Vector containing Hashtables.

Class next.util.PropertyListUtilities

```
java.lang.Object
   |
   +    next.util.PropertyListUtilities
```

---

public class **PropertyListUtilities**
extends Object

The PropertyListUtilities class provides a collection of utility methods that deal with property lists. One method, **propertyListFromString(**String**)** converts string objects containing ASCII encoded property lists to property list objects. Another method, **stringFromPropertyList(**Object**)** does the opposite conversion, and the third **propertyListsAreEqual(**Object, Object**)** compares property list objects. These methods are useful for serializing and deserializing property lists in and out of string objects.

See Also:
KeyValueCoding

Method Index

o **propertyListFromString**(String)
Returns a property list object derived from interpreting the ASCII property list representation *stringRep*.
o **propertyListsAreEqual**(Object, Object)
Returns TRUE if *plist1* and *plist2* are the same, FALSE if they are not.
o **stringFromPropertyList**(Object)
Returns a string object, which has only characters from the ASCII subset of Unicode.

Methods

o **stringFromPropertyList**

```
public static String stringFromPropertyList(Object plist)
```

Returns a string object, which has only characters from the ASCII subset of Unicode. The *plist* object must be one of the other permitted property list types: Vector, Hashtable, or byte[].

- **propertyListFromString**

```
public static Object propertyListFromString(String stringRep)
```

Returns a property list object derived from interpreting the ASCII property list representation *stringRep*. The string object should encode only one top level object. The return object is one of Vector, Hashtable, String, or byte[].

Throws: RuntimeException

Throws a RuntimeException if there is an error parsing the property list representation.

- **propertyListsAreEqual**

```
public static boolean propertyListsAreEqual(Object plist1,
                                            Object plist2)
```

Returns TRUE if *plist1* and *plist2* are the same, FALSE if they are not. The compared objects must be of the same type (else FALSE is always returned). This method performs recursive compare operations on collections such as Vectors and Hastables. If anything is different, the method returns FALSE.