Comspec

Component Writer's Guide

1999-10-28, Peter Norrhall (Cell Network) Göteborg
(Sightly edited by Mats Lundälv, DART 1999-2000)

1. Introduction

To get a better understunding of how to create new components in Comlink we will go through all steps to create a component. We will create a component, which can select one of several Switch Input - Switch Scan couples and use it as output to a Selection Set component.
It shall be possible to change the Switch Scan during run time as well.

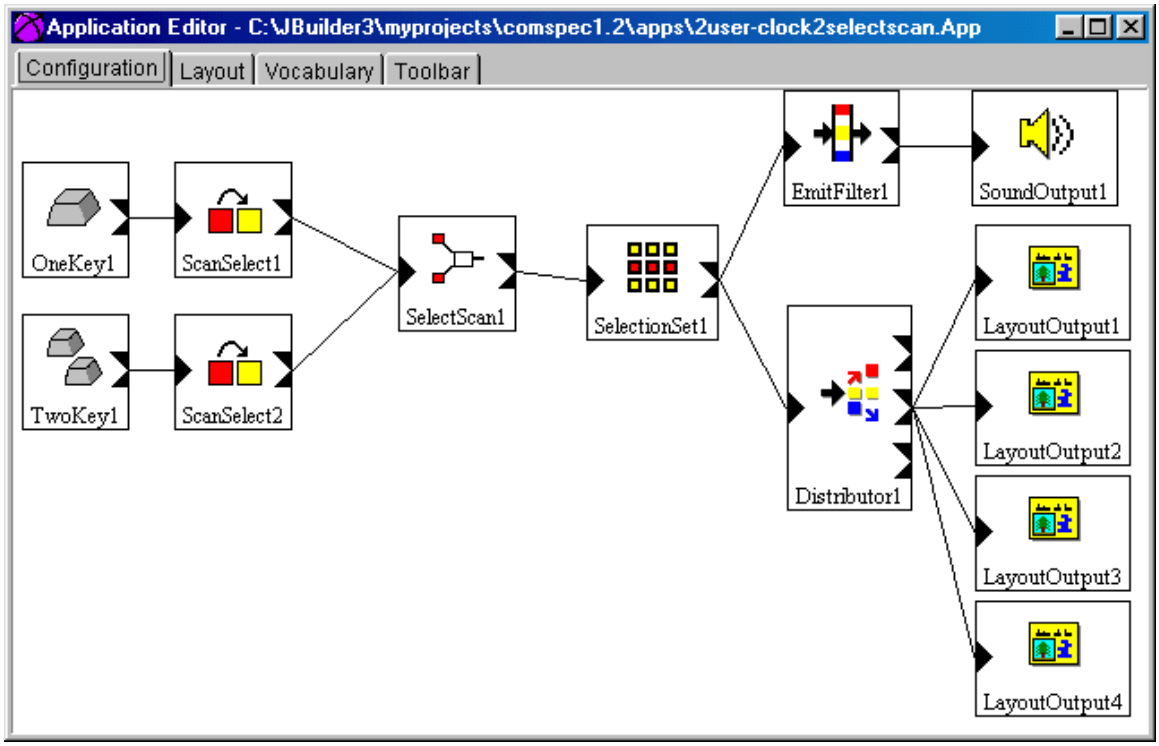
1.1. Purpose

The purpose of this document is to describe how to write components for Comlink

1.2. Prerequisites

As a component developer you are expected to be familiar with Java, have basic knowledge about JavaBeans and about the use and architecture of Comlink. For the latter we refer to [1] "Comspec on Java", [3] "ComLink User Manual" and [4] "ComLink Sample Applications".

Comlink components are designed to mimic the structure of JavaBeans with a few extensions. Because of this, it is essential to be well aquainted to the rules for creating JavaBeans. There are numerous books about the JavaBeans technology, and which describe how components should be designed and implemented to be suitable for "Visual" development tools, such as IBM Visual Age for Java, Visual Symantec Café and Borland Jbuilder.
An example of such a book is [2] "JavaBeans by Example", where it is recommendable to read chapters 1-3 at least

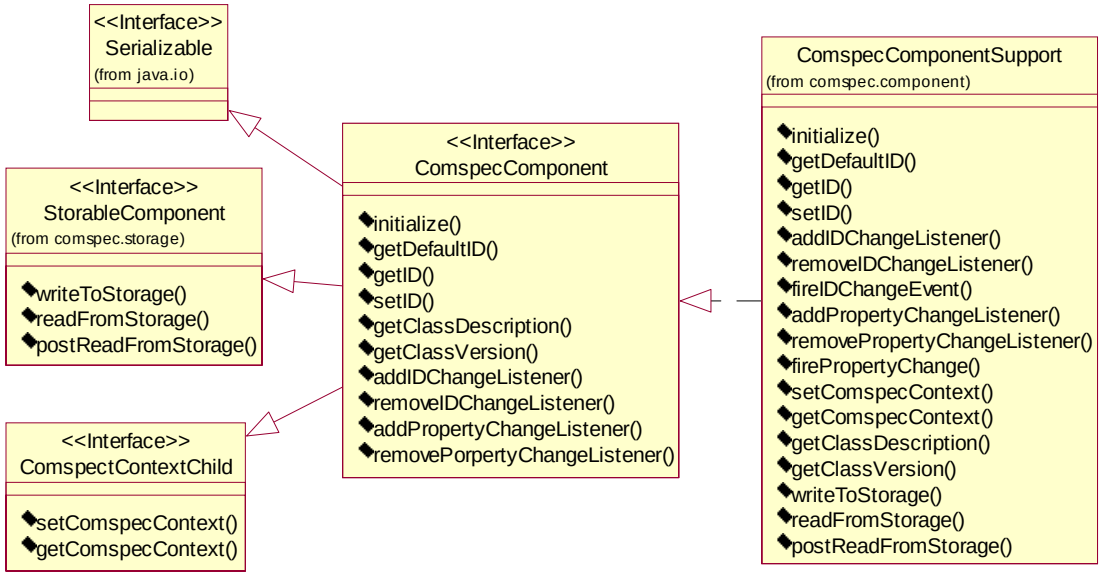


2. SelectScan

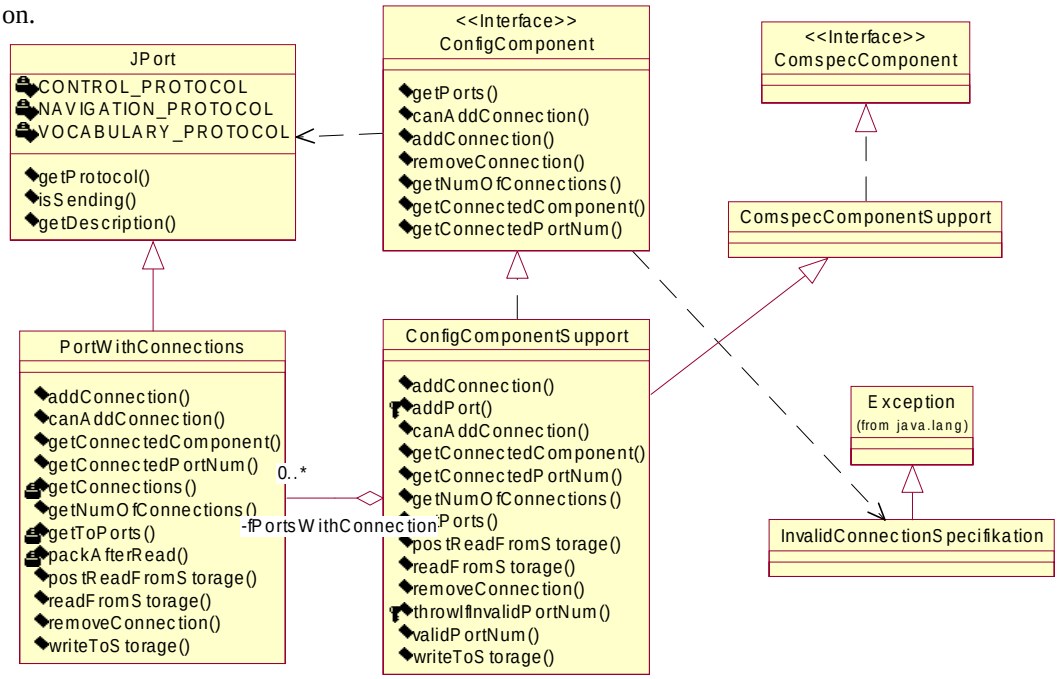
2.1. Configuration

Configuration components belong to the Configuration subsystem. SelectScan is such a component.

A configuration component implements the interface ConfigComponent or it can extend the ConfigComponentSupport. ConfigComponentSupport implements the ConfigComponent interface and it extends the ComspecComponent interface which all components in Comlink have to support.



Picture ComspecComponent – The interfaces Seializable and StorableComponent will be explained in the Storage section below. The ComspecConextChild is another interface which has to be supported to let the component have access to different services (see [1], chapter 4.3-4.4). We will use ComspecContextChild later on.



Picture ConfigComponent and ConfigComponentSupport

2.2. Port

Specification – Port

The Port class implements the neccessary functionality for keeping track of information regarding the ports used in connections.

Specification - ConfigComponentSupport

The ConfigComponentSupport class is an abstract class that implements the ConfigComponent interface. This class can be used as a base class for configuration components that do not have any special requirements on the implementation of the ConfigComponent interface.

ConfigComponentSupport uses the private class PortsWithConnections, from which it inherits, to implement most of the methods in ConfigComponent by means of delegation.

2.3. Protocol

SelectScan has one port for incoming NAVIGATION signals from the Switch Scan components and one port for outgoing NAVIGATION signals. The receiving port allows multiple connections, but the sending port only allows one NavigationReceiver (i.e. Selection Set) to be connected. To create ports we use the protected methods *addPort*, which comes in two flavours

```
protected void addPort(String protocol, boolean sending, boolean
multipleConnectionsAllowed)

protected void addPort(String protocol, boolean sending, String description,
boolean multipleConnectionsAllowed)
```

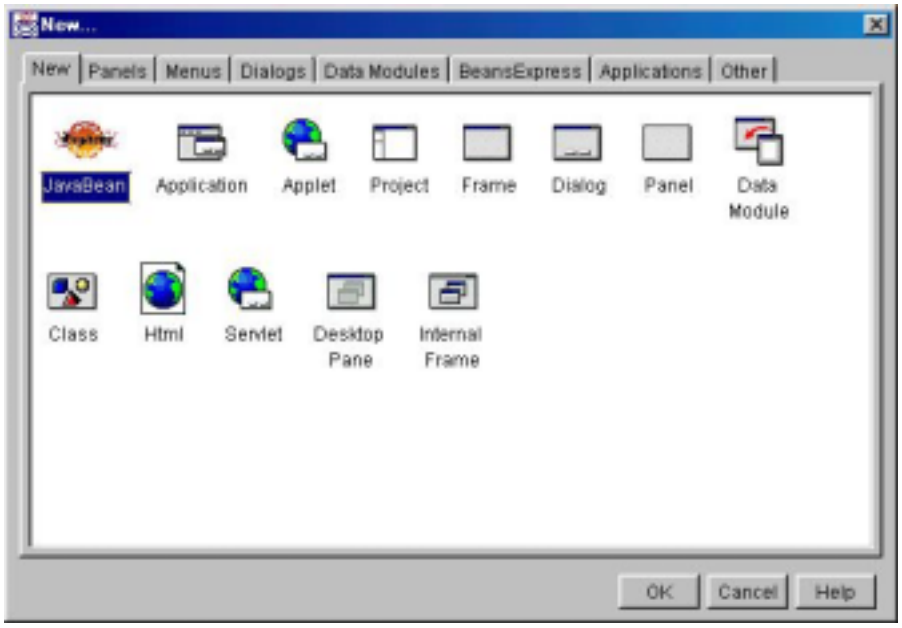
`Protocol`	Should be one of the predefined CONTROL_PROTOCOL, NAVIGATION_PROTOCOL or VOCABULARY_PROTOCOL, but it is allowed to define your own. SelectScan uses the NAVIGATION_PROTOCOL.
`Sending`	is whether the port is receiving or sending ”signals”
`MultipleConnectionsAllowed`	is True if this port can have multiple connections, false if only a single connection is allowed
`Dscription`	if there is a special description for this port

2.4. JavaBean

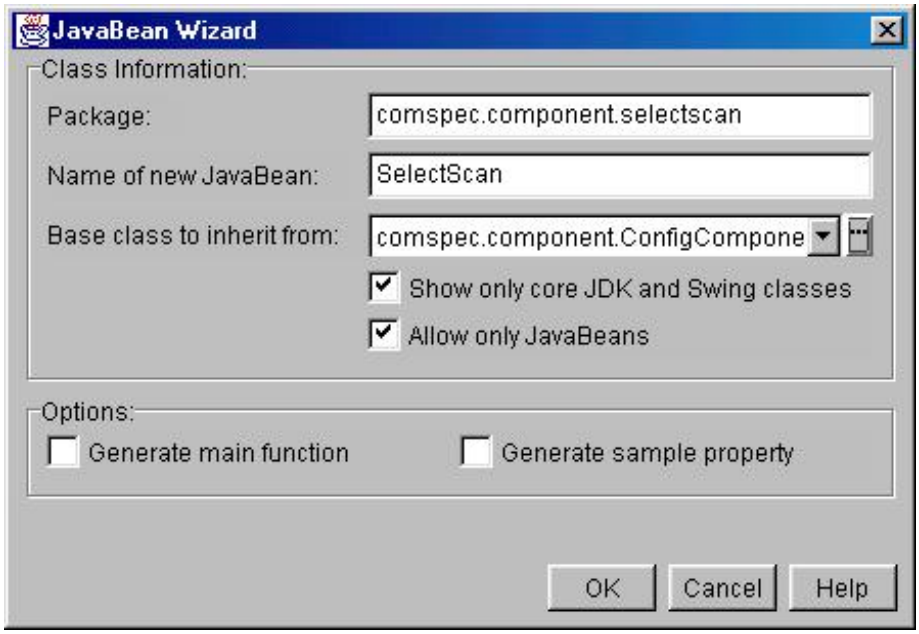
We shall now create the SelectScan class. You can do this by hand, but we are going to use Borland JBuilder's different kinds of wizards.

First of all we have to decide where we should put our classes. Since our component is called SelectScan (which follows the naming conventions suggested in [1], chapter 4.2 Naming Conventions) we create a directory comspec\component\selectscan where we put our component(s).

We start by using the JavaBeans Express Wizard by selecting File|New.

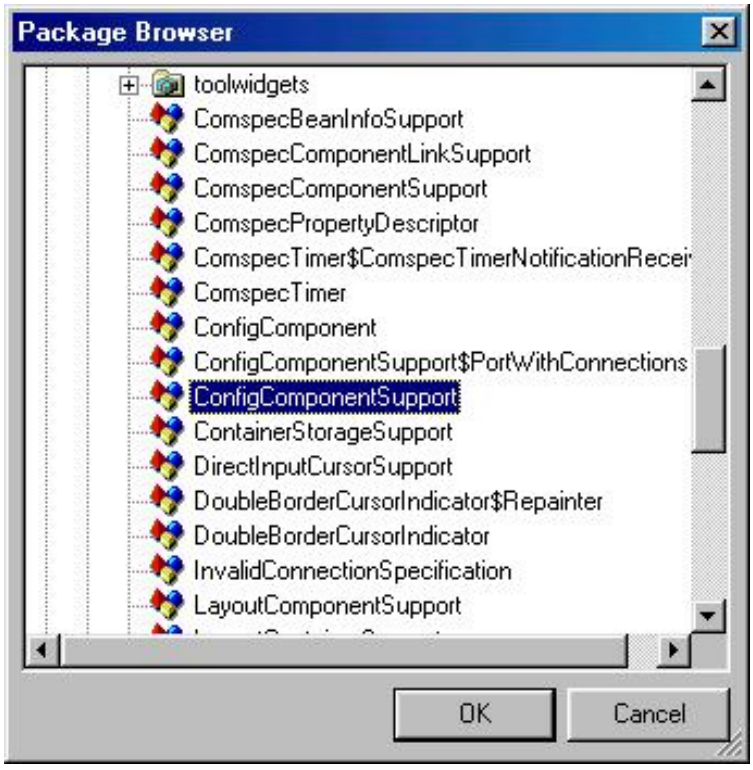


Select JavaBean on the New panel, and then the OK button.



Enter comspec.component.selectscan as the package name and SelectScan as the name of the JavaBean. Do not select the Generate main function. You can let the Generate sample property remain if you want to. We are going to remove this property later on anyway.

Instead of using the java.swing.JPanel we are going to use the ConfigComponentSupport class as our base class. Enter it manually or use the package browser.



A chunk of code is created, which looks like the listing below. The copyright and version information has been changed to comply with the Comspec-standard.

```
package comspec.component.selectscan;

import comspec.component.ConfigComponentSupport;

/**
 * <P><B>© Copyright 1999, the Comspec consortium.</B><P>
 *
 * This class implements the select scan component
 *
 * <pre>
 * Change History (most recent first):
 *
 * nr   DD/MM/YY  who  reason
 * (1)  17/09/99  PN   created
 * </pre>
 *
 * @author Peter Norrhall (PN), Linné Göteborg AB
 * @version 0.1
 *
 */

public class SelectScan extends ConfigComponentSupport {
  private String sample = "Sample";

  public SelectScan() {
  }

  public String getSample() {
    return sample;
  }
```

```
  public void setSample(String newSample) {
    sample = newSample;
  }

}
```

Listing – Select Scan

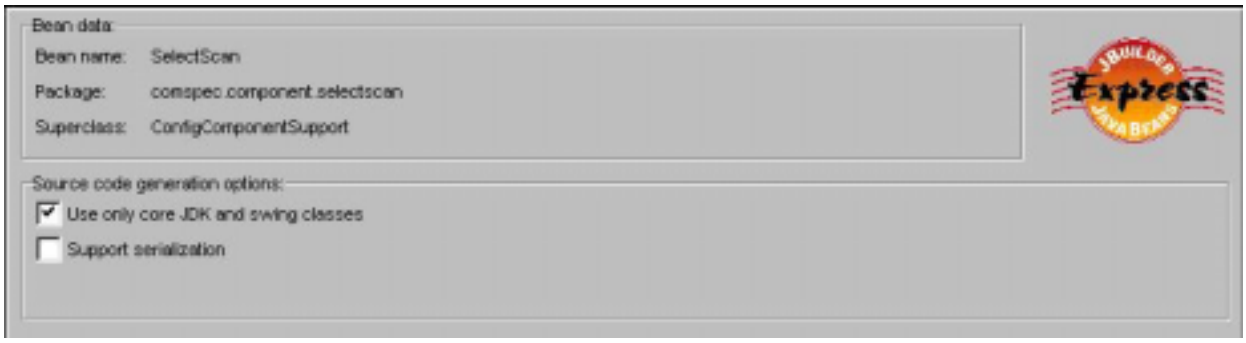
2.5. BeanInfo

For the component we also create a BeanInfo class following the naming convention for beans: <componentname>BeanInfo. SelectScanBeanInfo will be placed in the comspec\component\selectscan directory.
You can use JBuilder's BeanInfo wizard, but since the ComspecBeanInfoSupport class implements the most important methods, we use that class as base class for our SelectScanBeanInfo.

```
package comspec.component.selectscan;

import comspec.component.*;

/**
 * <P><B>© Copyright 1999, the Comspec consortium.</B><P>
 *
 * This class implements the bean info of the select scan component
 *
 * <pre>
 * Change History (most recent first):
 *
 * nr   DD/MM/YY  who  reason
 * (1)  17/09/99  PN   created
 * </pre>
 *
 * @author Peter Norrhall (PN), Linné Göteborg AB
 * @version 0.1
 *
 */

public class SelectScanBeanInfo extends ComspecBeanInfoSupport {

}
```

Listing – Select Scan Bean Info

Since the SelectScan is a true JavaBean we can inspect the SelectScan using JBuilder's Bean designer. Select the SelecScan.java file and select the Bean-tab to the right. At first the general tab is displayed showing some general data about the SelectScan bean.



Picture – SelectScan General Tab

2.6. Properties

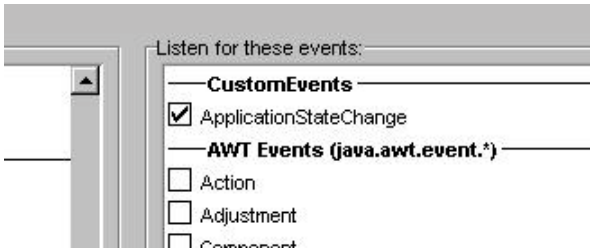
Selecting the Properties tab we can see that the bean has a property *sample* with both a get and a set method.

Picture – SelectScan Property Tab

Since we are not intending to use the sample property you could select the Remove Property-button to remove the property. Hopefully both the private field sample and the getSample and setSample methods are removed. If not, select the Source tab and do it yourself in the source code.

2.7. Events

Selecting the Bean tab again and the Events tab all events that the bean fires and listens for are displayed. Since all Comspec components has to listen for the ApplicationStateChange event you can select the ApplicationStateChange event to the right in the Listen for these events list.



Picture – SelectScan Events

JBuilder should by now have added the following

```
import comspec.system.*;
…
public class SelectScan extends ConfigComponentSupport implements
ApplicationStateChangeListener {

public void applicationStateChanging(ApplicationStateChangeEvent e) throws
…
```

```
java.beans.PropertyVetoException {
  }

  public void applicationStateChanged(ApplicationStateChangeEvent e) {
  }
}
```

Listing – SelectScan ApplicationStateChange support

2.8. Component linking

Before we start to implement these methods we shall add a property to select a switch scanner. Since we are dealing with another component we have to do this using the class ComspecComponentLinkSupport that implements the *ComspecComponentLink* interface.

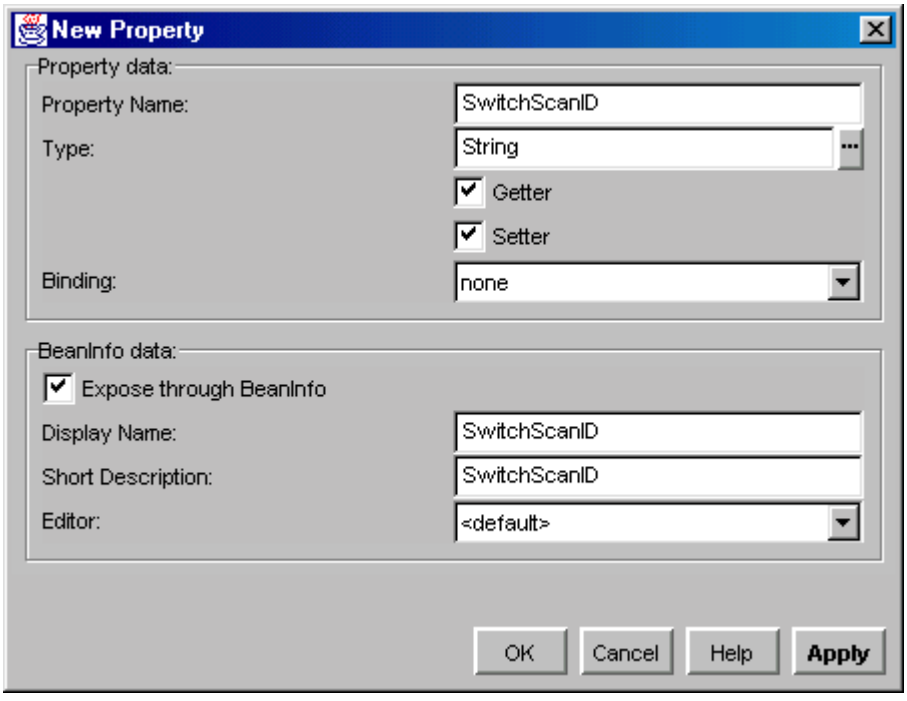
`Public ComspecComponent getComponent()`	`get the Component this link points to`
`Public void setComponent(ComspecComponent component)`	`set the Component this link points to`
`Public void resolveRef(ComspecInputStream in)`	`resolve the link to the Comspec component that belongs to the read reference`

Specification - ComspecComponentLinkSupport

Instances of ComspecComponentLinkSupport class keep references to other components. The class handles externalization/internalization of the reference, and it also will support remove and undo of remove operations on the referenced component from its container.

We name the property SwitchScanID. It is intended to be ID/name of the selected Switch Scan component and NOT the reference to it, since we must be able to restore the selection after we have loaded the application from storage.

To add the property we once again use the BeanExpress.



Picture SwitchScanID property

We also manually add the private field switchScan:

```
private ComspecComponentLinkSupport switchScan;
```

switchScan is the reference container to the selected SwitchScan component. We also add the private setter and getter methods setSwitchScan and getSwitchScan for this property to be used in the getSwitchScandID and setSwitchScanID methods.

```
  //*************************************************************
  // SwitchScanID Property
  //*************************************************************

  /** This method is used to get the ID of the switch scan
    * If no element is set NONE_SELECTED_STR
    * is returned
    */
  public String getSwitchScanID() {
    if (switchScan.getComponent() == null)
      return NONE_SELECTED_STR;
    else
      return switchScan.getComponent().getID();
  }

  /** This method is used to set the input device of the
    * selection set by ID.
    * Use NONE_SELECTED_STR to specify that no element
    * should be set.
    */
  public void setSwitchScanID(String theID) {
    if (theID == NONE_SELECTED_STR)
      setSwitchScan(null);
    else
      if (getComspecContext() == null)
        return;

      ObjectRegistry reg = (ObjectRegistry)getComspecContext().
         getService(ObjectRegistry.class, this);
      if (reg != null)
       setSwitchScan((comspec.component.ConfigComponent)reg.findObject(theID));
  }
```

2.9. BeanInfo Property Description

We are now ready to complete the SelectScanBeanInfo class. We override the getPropertyDescriptors method in the following way:

```
public PropertyDescriptor[] getPropertyDescriptors() {
    try {
      PropertyDescriptor[] props = new PropertyDescriptor[2];
      int x = 0;
      props[x++] = property(
        "ID",
        "Component ID",
        SelectScan.class,
        UserLevel.INTEGRATOR);
      props[x++] = property(
        "switchScanID",
        "Switch Scan",
```

```
        SelectScan.class,
        ScannerSelectionPE.class);

      return props;
    }
    catch (IntrospectionException e) {return super.getPropertyDescriptors();}
  }
```

Listing – SelecScanBeanInfo.getPropertyDescriptors

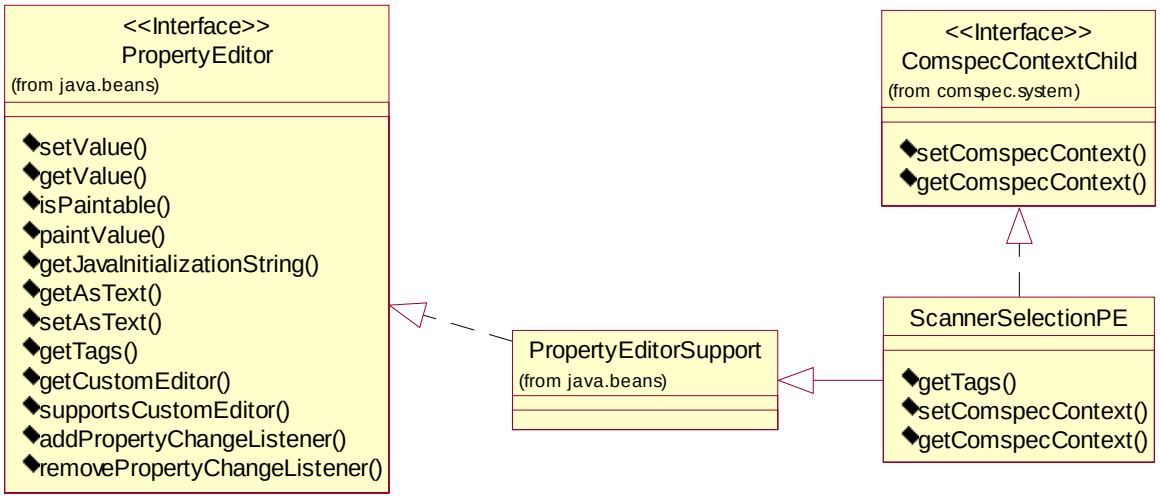
When the component is selected in the configuration environment, only two properties are displayed in the property sheet *Component ID* and *Switch Scan*. The property method is implemented in the ComspecBeanInfoSupport class with different kinds of parameters

```
public static PropertyDescriptor property(String name, String description,
                Class beanClass)
                throws IntrospectionException

public static PropertyDescriptor property(String name, String description,
                Class beanClass, Class editorClass)
                throws IntrospectionException

public static PropertyDescriptor property(String name, String description,
                Class beanClass, int maxUserLevel)
                throws IntrospectionException

public static PropertyDescriptor property(String name, String description,
                Class beanClass, Class editorClass, int maxUserLevel)
                throws IntrospectionException
```

Name	the property name
Description	the name / description to display
BeanClass	the class which the property belongs to
EditorClass	The PropertyEditor class for this property
MaxUserLevel	The maximum user level at which this property is visible

2.10. Property Editor

As you can see the Component ID property is only visible to the Integrator. For the Switch Scan property we have a editorClass, ScannerSelectionPE. The purpose of this class is to display all possible Switch Scan components in a list when the Switch Scan property is selected in the property sheet.



ScannerSelectionPE implements the PropertyEditor.getTags method to return all components that support the ISwitchScanner interface. To get those components ScannerSelectionPE uses the Object Registry service in the Comspec Context, by implementing the ComspecContextChild interface.

```
public String[] getTags() {
    if (fContext == null)
      return new String[] {"Context not found"};

    ObjectRegistry reg =
            (ObjectRegistry)fContext.getService(ObjectRegistry.class, this);
    if (reg == null)
      return new String[] {"Object registry not found"};

    // Find all the control senders
    Vector v = reg.objects(comspec.component.switchscan.ISwitchScanner.class);
    if (v == null)
      return new String[] {"Null vector of control senders"};

    // sort the senders
    comspec.support.Sorter.sort(v, new
             comspec.support.ComspecComponentComparator());

    // copy names into String[]
    int count = v.size() + 1;
    String[] strs = new String[count];
    // add the "- None Selected -" string
    strs[0] = SelectScan.NONE_SELECTED_STR;
    // copy names
    for (int i = 1; i < count; i++) {
      strs[i] = ((ComspecComponent)v.elementAt(i-1)).getID();
    }

    return strs;
  }
```

Listing – ScannerSelectionPE.getTags

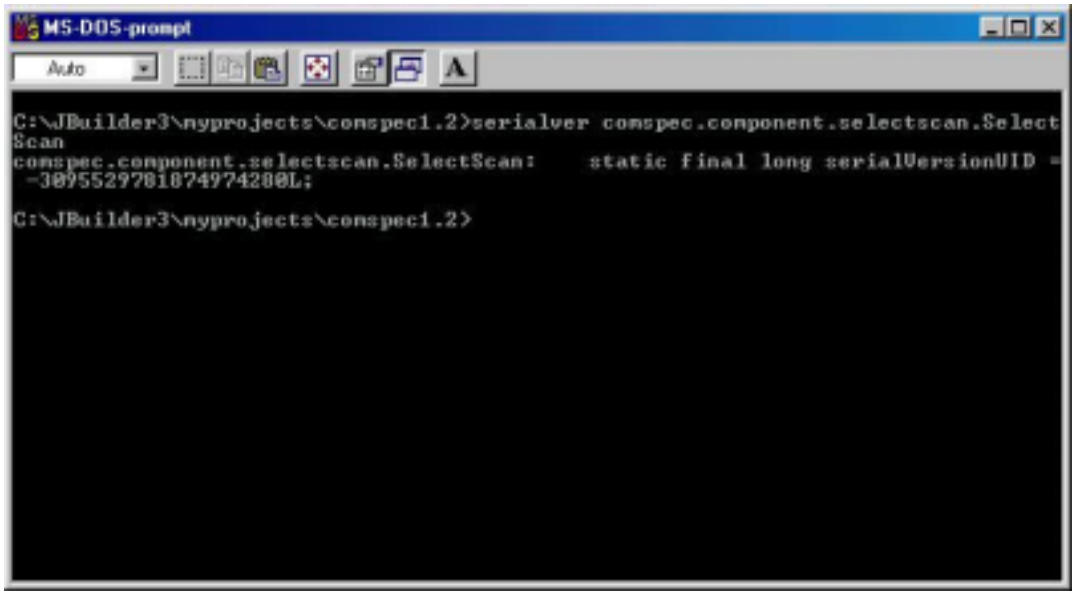
Note : ConfigComponentSupport allows all configuration components to connect to it, if the sending protocol is the same as the receiving (CONTROL_PROTOCOL, NAVIGATION_PROTOCOL, VOCABULARY_PROTOCOL, etc).

2.11. ObjectRegistry

The ObjectRegistry is a Comlink service for components to find other components by name or by type.

To be found and stored each component must have a unique serial number. This number is set as a static field in the class. The number is generated with the JDK utility Serialver from the command line as explained in [2] "JavaBeans by Example", pages 94-95.

You could either use the command line version or the graphical interface version using the -show parameter. Using the graphical interface you are able to copy & paste the serial number into your code in a more convenient way.



Picture – Serialver [classname]

Picture – Serialver [–show]

Copy the serial number into the SerialScan.java file. It should now look like this

```
public class SelectScan extends ConfigComponentSupport implements
ApplicationStateChangeListener {

  static final long serialVersionUID = -3095529781874974280L;

  private ComspecComponentLinkSupport switchScan;
```

Listing – serialVersionUID

2.12. Description

In addition to the ID the component shall have a default name, description and a version number. Overriding the three methods getDefaultID, getClassDescriptions and getClassVersion does this.

```
private static final String defID = "SelectScan1";
private static final String desc = "Config element: Select Scan";
private static final ComspecVersion vers = new ComspecVersion(0, 1);
…
/**
   * Get the default ID this component class wants to use.
   * @return the default ID, like "Component1"
   */
public String getDefaultID() {
    return defID;
```

```
}

/** Get a string describing this component class in a way
    * understandable to the integrator/facilitator
    * @return a string containing a description of the component class.
    */
public String getClassDescription() {
    return desc;
}

/** Get the current version of this component class
    */
public ComspecVersion getClassVersion() {
    return vers;
  }
```

2.13. Storage

According to [1], chapter 3.4 Storage and Clipboard handling and chapter 4.5 Storage, each component must be able to store itself in the Comspec Storage system. All components are *serializable*, but that is only used for temporary storage such as Copy & Paste.

The component must implement the *comspec.storage.StorableComponent* interface.

```
public interface StorableComponent {
  /** Write the content of the component using the specified
    * storage writer
    */
  public void writeToStorage(StorageWriter writer);

  /** Read the content of the component using the specified
    * storage reader
    */
  public void readFromStorage(StorageReader reader);

  /** Do any initialization and reference resolvement that
    * should be done after all components has been read
    * from storage
    */
  public void postReadFromStorage(StorageRefResolver resolver);
}
```

These methods are implemented in the ConfigComponentSupport class. But, they are only storing the ports of the configuration components. So therefore we have to override them in SelectScan since we want to save the SwitchScan property.

```
  //*****************************************************************
  // StorableComponent interface implementation
  //*****************************************************************

  public void writeToStorage(StorageWriter writer) {
    try {
      super.writeToStorage(writer);

      ByteArrayOutputStream bst = new ByteArrayOutputStream();
      DataOutputStream dst = new DataOutputStream(bst);

      // write properties
      switchScan.writeToStorage(writer, dst);

      dst.flush();
```

```
      dst.close();

      writer.addPropertyVersion("SelectScan", "v1.0", bst.toByteArray());
    }
    catch (Exception e) {}
  }

  public void readFromStorage(StorageReader reader) {
    super.readFromStorage(reader);
    byte[] propData = reader.getPropertyVersion("SelectScan", "v1.0");
    if (propData == null)
      return ;
    try {
      ByteArrayInputStream bst = new ByteArrayInputStream(propData);
      DataInputStream dst = new DataInputStream(bst);

      // read properties, read links as references (int values)
      switchScan.readFromStorage(reader, dst);

      dst.close();
    }
    catch (Exception ex) {
    }
  }

  public void postReadFromStorage(StorageRefResolver resolver) {
    super.postReadFromStorage(resolver);
    // resolve reference links
    switchScan.postReadFromStorage(resolver);
  }
```

Listing – Storage implementation

Note how easy it is to store the reference to the SwitchScan component using the ComspecComponentLinkSupport object switchScan using the methods readFromStorage and writeToStorage. To store properties of basic types (int, String etc) the DataInputStream and DataOutputStream method's read/writeInt, read/writeChars, …

2.14. Component Registration

To add the SelectScan component into the ComLink environment the class has to be registered. This is done according to [1], section 3.2.3 Description, adding the class to the Configuration section in the class.reg.txt file.

```
[Configuration]
…
*comspec.component.selectscan.SelectScan
-Label=Select Scan
```

Before we can use the SelectScan component in Comlink we also have to create icons according to [1], section 4.10.2. We create two icons SelectScan_16x16c.gif and SelectScan_32x32c.gif and place them in the images directory.

2.15. Test and Deploy

SelectScan is now ready for prime time. If we have done everything correct in our coding, and registered it correctly, the SelectScan component should appear on the Configuration Toolbar if we select Integrator User Level.

The complete code is listed in Appendix B

2.16. Enhancement

As you might see when you use the SelectScan is that all configuration components, supporting the ISwitchScanner interface, are displayed for the Switch Scan property, and not just those that is connected to the SelectScan component. This is because the ScannerSelectionPE.getTags returns all ISwitchScanner components in the editor. To only select only those connected to the component the property editor must know the target component. Implementing the *ComspecPropertyEditor* interface does this

`public void setTarget(Object target)`	`set the target the PropertyEditor is editing a property for`

We add a protected field fTarget and implement the method

```
protected Object fTarget;
…
public void setTarget(Object target) {
    fTarget = target;
  }
```

setTarget is called when the property is selected in the property sheet.
The next thing to do is to change the getTags method. If fTarget is set and it is a ConfigComponent (i.e. SelectScan) we can get all connected components using the getConnectedComponent method.

```
public String[] getTags() {
    if (fContext == null)
      return new String[] {"Context not found"};

    ObjectRegistry reg =
          (ObjectRegistry)fContext.getService(ObjectRegistry.class, this);
    if (reg == null)
      return new String[] {"Object registry not found"};

    Vector v = null;

    if ((fTarget != null) && (fTarget instanceof ConfigComponent)) {
      ConfigComponent cc = (ConfigComponent)fTarget;
      ConfigComponent connected;
      v = new Vector();
      try {
        int count = cc.getNumOfConnections(ConfigComponentSupport.ReceivePort);
        for (int i = 0; i < count; i++) {
          connected =
             cc.getConnectedComponent(ConfigComponentSupport.ReceivePort, i);
          if (connected instanceof comspec.component.switchscan.ISwitchScanner)
             v.add(connected);
        }
      } catch (InvalidConnectionSpecification e) {
      }
```

```
    } else {
      // Find all the control senders
      v = reg.objects(comspec.component.switchscan.ISwitchScanner.class);
    }

    if (v == null)
      return new String[] {"Null vector of control senders"};
    // sort the senders
    comspec.support.Sorter.sort(v,
                new comspec.support.ComspecComponentComparator());

    // copy names into String[]
    int count = v.size() + 1;
    String[] strs = new String[count];
    // add the "- None Selected -" string
    strs[0] = SelectScan.NONE_SELECTED_STR;
    // copy names
    for (int i = 1; i < count; i++) {
      strs[i] = ((ComspecComponent)v.elementAt(i-1)).getID();
    }

    return strs;
  }
```

3. Toolbar

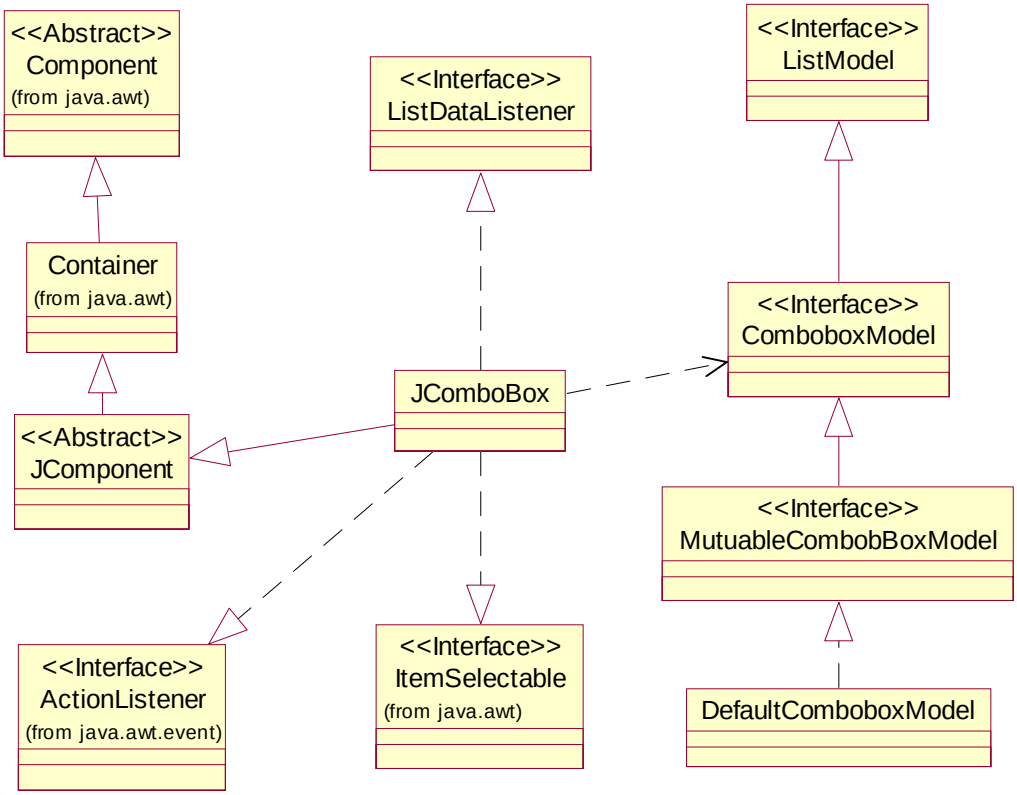
One of the requirements was to be able to select another SwitchScan component during runtime. This is done from the Toolbar.

We have to create a component that can select a SwitchScan component. Since there can be an infinit number of components connected to the SelectScan a good choice would be to create a combobox that contains the connected and selectable SwitchScan components.

3.1. Combobox Control

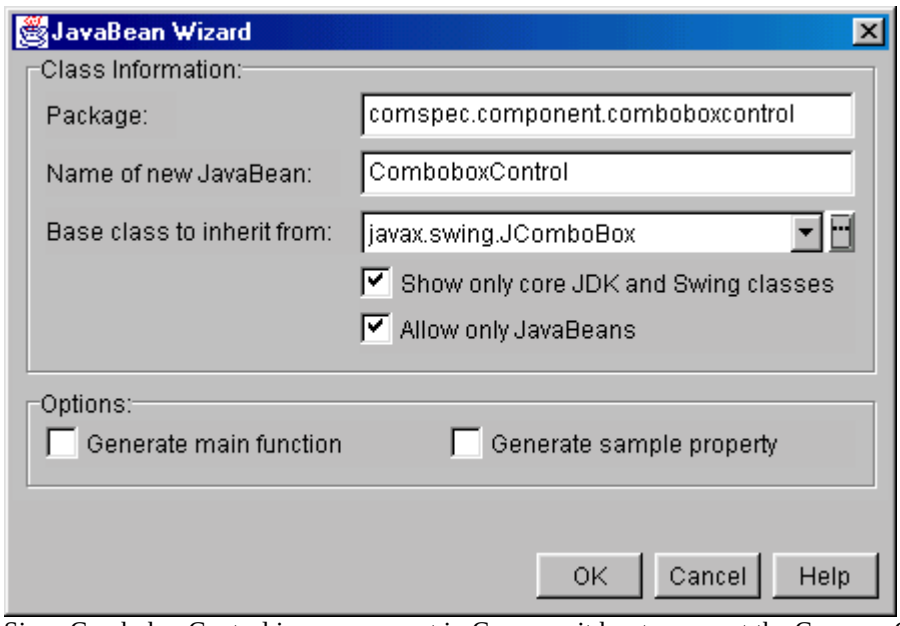
Since Comlink is mainly built using Swing we create our ComboboxControl based on javax.swing.JCombobox.

3.2. JComboBox



Classdiagram - JComboBox

Once again we use the Bean wizard to create our component.

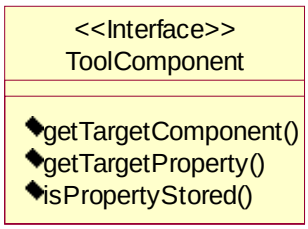


Since ComboboxControl is a component in Comspec it has to support the ComspecComponent interface. As before we use delegation and the class ComspecComponentSupport.

3.3. ToolComponent interface

A component on the toolbar must support the ToolComponent interface.

The ToolComponent interface defines the way the Run-time environment access the Toolbar-like components to determine which properties are to be stored in a user profile. It also unifies the Property names for the TargetComponent and the TargetProperty the control links to.

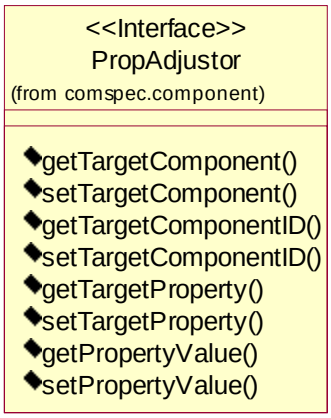


`ComspecComponent getTargetComponent()`	`Get the ComspecComponent this control accesses a property of`
`String getTargetProperty()`	`Get the name of the property in the target ComspecComponent this control accesses`
`Boolean isPropertyStored()`	`Whether the property should be stored in the user profile or not. This should default to true, but the integrator should be able to turn this off`

Note: if multiple ToolComponents link to the same TargetComponent and TargetProperty, the Run-time environment may use the setting of another control.

3.4. PropAdjustor interface

A toolbar control is supposed to set/get another components property. The user has to select the component and the property to set/get. The *PropAdjustor* interface defines all this.



The PropAdjuster interface defines all the methods to be implemented by Layout- and Toolbar components that want to change the property value of another component

GetTargetComponent	Get the component that owns the property to change.
SetTargetComponent	Set the component that owns the property to change.
GetTargetComponentID	Get the name of the component that owns the property to change
SetTargetComponentID	Set the component that owns the property to change by name.
GetTargetProperty	Get the name of the property to change.
SetTargetProperty	Set the name of the property to change.
GetPropertyValue	Get the current value of the property.
SetPropertyValue	Set a new value for the property.

And once again we delegate the implementation to another class, the *PropAdjustorSupport* class. Below is an example (getTargetComponent)

```
  /**
   * Get the ComspecComponent this control accesses a property of.
   * @returns the target ComspecComponent, or null if not linked.
   */
  public ComspecComponent getTargetComponent() {
    if (fPropAdjuster != null)
      return fPropAdjuster.getTargetComponent();
    return null;
  }
```

3.5. Bound Properties

Since we want the control to be notified when the property is changed, the property is defined as "bound". To be notified about changes the control has to implement the *PropertyChangeListener* interface and be registeredl as a listener to the target component.

PropertyChangeListener	A "PropertyChange" event gets fired whenever a bean changes a "bound" property. You can register a PropertyChangeListener with a source bean so as to be notified of any bound property updates.

PropertyChangeListener has one method

PropertyChange	This method gets called when a bound property is changed.

3.6. BeanInfo

We publish the properties for the combobox in the same way as for SelectScan by creating a BeanInfo class with the name ComboboxControlBeanInfo and place it in the same directory as ComboboxControl.
The BeanInfo class inherits this time as well from ComspecBeanInfoSupport and we override the method getPropertDescriptors as below.

```
public PropertyDescriptor[] getPropertyDescriptors() {
    try {
      PropertyDescriptor[] props = new PropertyDescriptor[7];
      int x = 0;
      props[x++] = property(
        "ID",
        "Component ID",
        ComboboxControl.class,
        UserLevel.INTEGRATOR);
      props[x++] = property(
        "foreground",
        "Foreground Colour",
        ComboboxControl.class,
        ColorPropertyEditor.class);
      props[x++] = property(
        "background",
        "Background Colour",
        ComboboxControl.class,
        ColorPropertyEditor.class);
      props[x++] = property(
        "font",
        "Font",
        ComboboxControl.class);
      props[x++] = property(
        "targetComponentID",
        "Target Component",
        ComboboxControl.class,
        TargetComponentPE.class,
        UserLevel.INTEGRATOR);
      props[x++] = property(
        "targetProperty",
        "Target Property",
        ComboboxControl.class,
        comspec.support.editor.BoundStringTagPropertyPE.class,
        UserLevel.INTEGRATOR);
      props[x++] = property(
        "propertyStored",
        "Store Property",
        ComboboxControl.class,
        UserLevel.INTEGRATOR);

      return props;
    }
    catch (IntrospectionException e) {
      return super.getPropertyDescriptors();
    }
```

Foreground, Background and Font are used to set the visual appearance of the combobox.

3.7. Property Editors

3.7.1. BoundStringTagPropertyPE

TargetComponentPE is a property editor to select a component and BoundStringTagPropertyPE is an editor that returns all properties that has at least one selection, i.e. implementing the getTags method, for the selected target component.

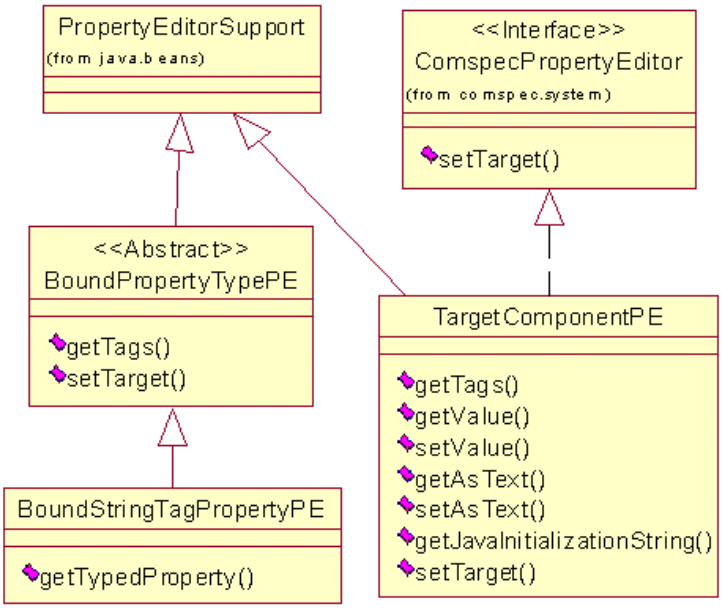


Diagram – BoundStringTagPropertyPE and TargetComponentPE

GetTypedProperty returns those properties of a component that support the getTags method.

The first step is to get all registered properties of the target class. This is achieved using the *java.beans.Introspector* class and the *java.beans.BeanInfo* class.

Specification Introspector

The Introspector class provides a standard way for tools to learn about the properties, events, and methods supported by a target Java Bean.

```
// get all properties of the target component
    PropertyDescriptor properties[];
    try {
      BeanInfo bi =
           Introspector.getBeanInfo(target.getTargetComponent().getClass());
      properties = bi.getPropertyDescriptors();
    }
    catch (IntrospectionException ex) {
      return null;
    }
```

The next step is to iterate through all properties and use the PropertyDescriptor of the property.

```
for (int i = 0; i < properties.length; i++) {
        PropertyDescriptor pd = properties[i];
}
```

We try to find out whether the property has a PropertyEditor registered to it. If so we instanciates that property editor:

```
Class pe = pd.getPropertyEditorClass();
PropertyEditor propEditor = null;
if (pe != null) {
   try {
      propEditor = (PropertyEditor)pe.newInstance();
   } catch (Exception ex) { // Drop through. }
…
}
```

Then we examine if the getTags method is implemented for the Property Editor:

```
if (propEditor != null) {
                String[] tags;
                tags = propEditor.getTags();
                if (tags.length > 0)
                  v.addElement(pd.getName());
              }
```

3.7.2. BoundConfigComponentPE

BoundConfigComponentPE is a property editor used to select all configuration components in the Comspec context.

Like BoundStringTagPropertyPE BoundConfigComponentPE inherits from BoundPropertyTypePE and it implements the getTypedProperty method.

It iterates through all properties to find those properties that is of type ConfigComponent interface:

```
// check if there are any int-properties
for (int i = 0; i < properties.length; i++) {
  PropertyDescriptor pd = properties[i];
  if (pd.isExpert() || pd.isHidden())
    continue;
  java.lang.reflect.Method getter = pd.getReadMethod();
  Object value = null;
  try {
    Object args[] = { };
    value = getter.invoke(target.getTargetComponent(), args);
    if (value instanceof ConfigComponent)
      v.addElement(pd.getName());
  }
  catch (Exception ex) {
  }
}
```

4. Layout Components

This guideline does not include an example of how to create a layout component. However, we can point out some things to remember when you want to make one yourself.

4.1. Swing

Make sure that your component is a Swing component. Most components of the current ComLink version are based on the java.awt.Component or java.awt.Container. You should use javax.swing.Jcomponent, Jpanel or equivalents instead. You can of course use some other Swing based component, JButton for example.

There are at least two reasons to use Swing:

100% pure Swing

The next version of ComLink will hopefully be a 100% Swing based application. Today AWT and Swing components are mixed, and that creates some problems. You can read about this on Sun's webpage.

Accessibility

All Swing component support the Java Accessibility interface. They are not implementing any full accessibility support but, at least, a good foundation is provided.

4.2. Interfaces to support

There are some special interfaces that a Layout Component should support, in addition to those that are mandatory for all components in ComLink

LayoutReceiver

The LayoutReceiver interface should be implemented by scannable layout components. The interface enables other components to check, get and set the navigation links of the component, and to find the vocabulary signals that should be output when the component is scanned or selected.

LayoutWithSingleRepr

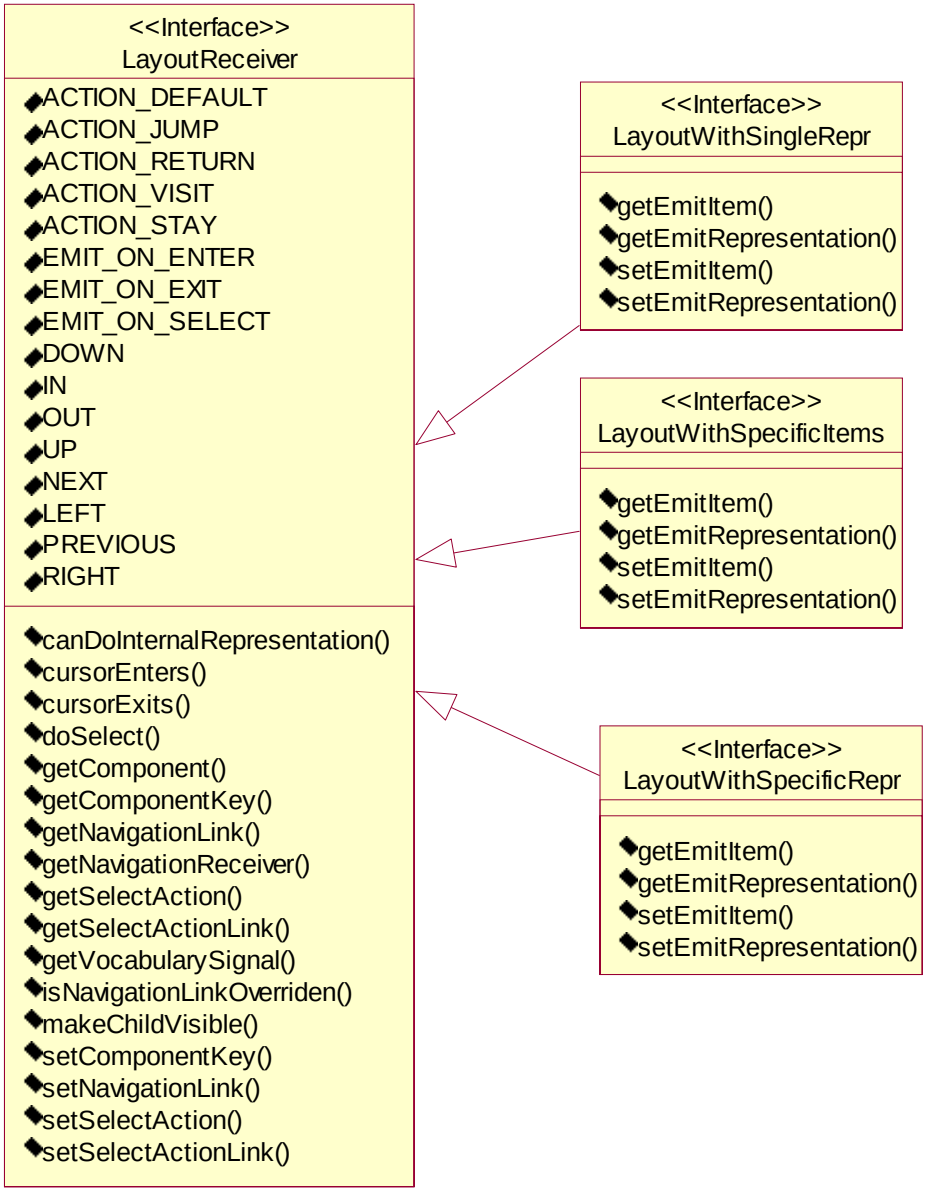
The LayoutWithSingleRepr extends the LayoutReceiver to give access to editing of a single vocabulary item and representation to emit

LayoutWithSpecificItems

The LayoutWithSpecificItem extends the LayoutReceiver to give access to editing of the vocabulary item and representations to emit. Implementors of this interface can emit different items and representations on enter, exit and select

LayoutWithSpecificRepr

The LayoutWithSpecificRepr extends the LayoutReceiver to give access to editing of the vocabulary item to use for emit and which representations to emit for it



4.3. Support Classes

There are some support classes that implement the layout interfaces above.

LayoutComponentSupport

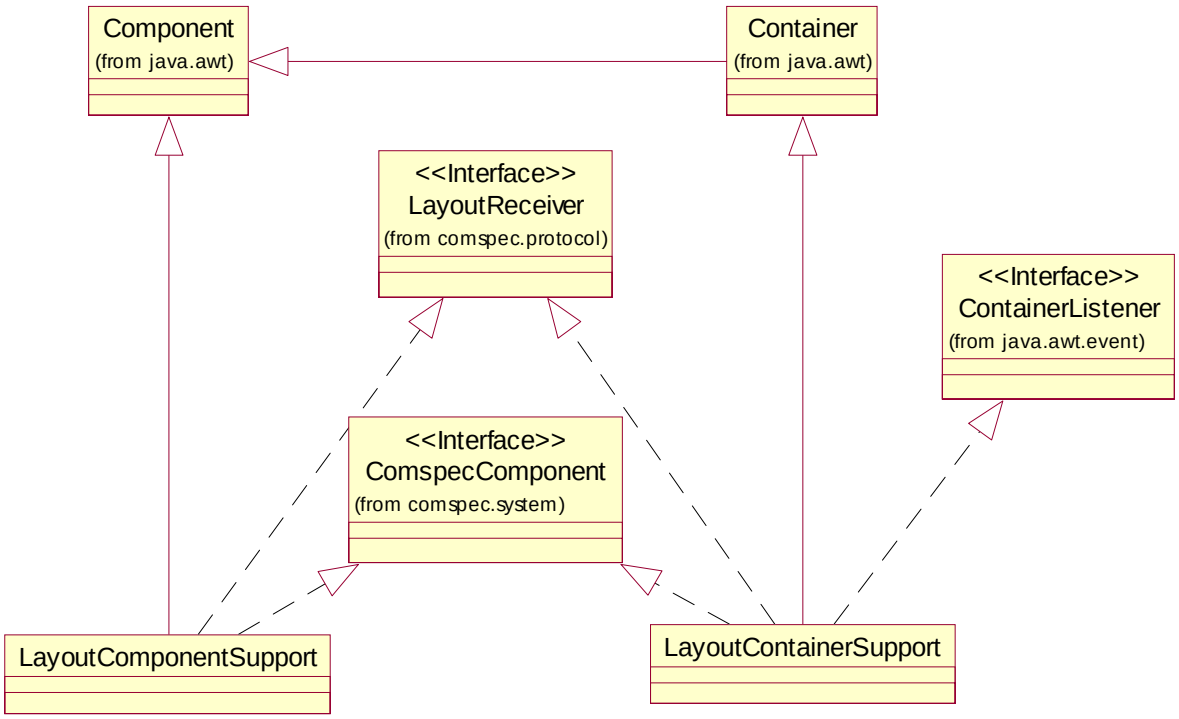
The LayoutComponentSupport class is an abstract class that implements the LayoutReceiver interface. This class can be used as a base class for layout components that do not have any special requirements on the implementation of the LayoutReceiver interface.

The implementation takes care of the getting and setting of navigation links and actions, and also the coordination of the painting of any CursorIndicators currently being displayed on this component. Subclasses are responsible for implementing getVocabularySignal()

See also LayoutReceiver for more detailed comments on most methods implemented by this class.

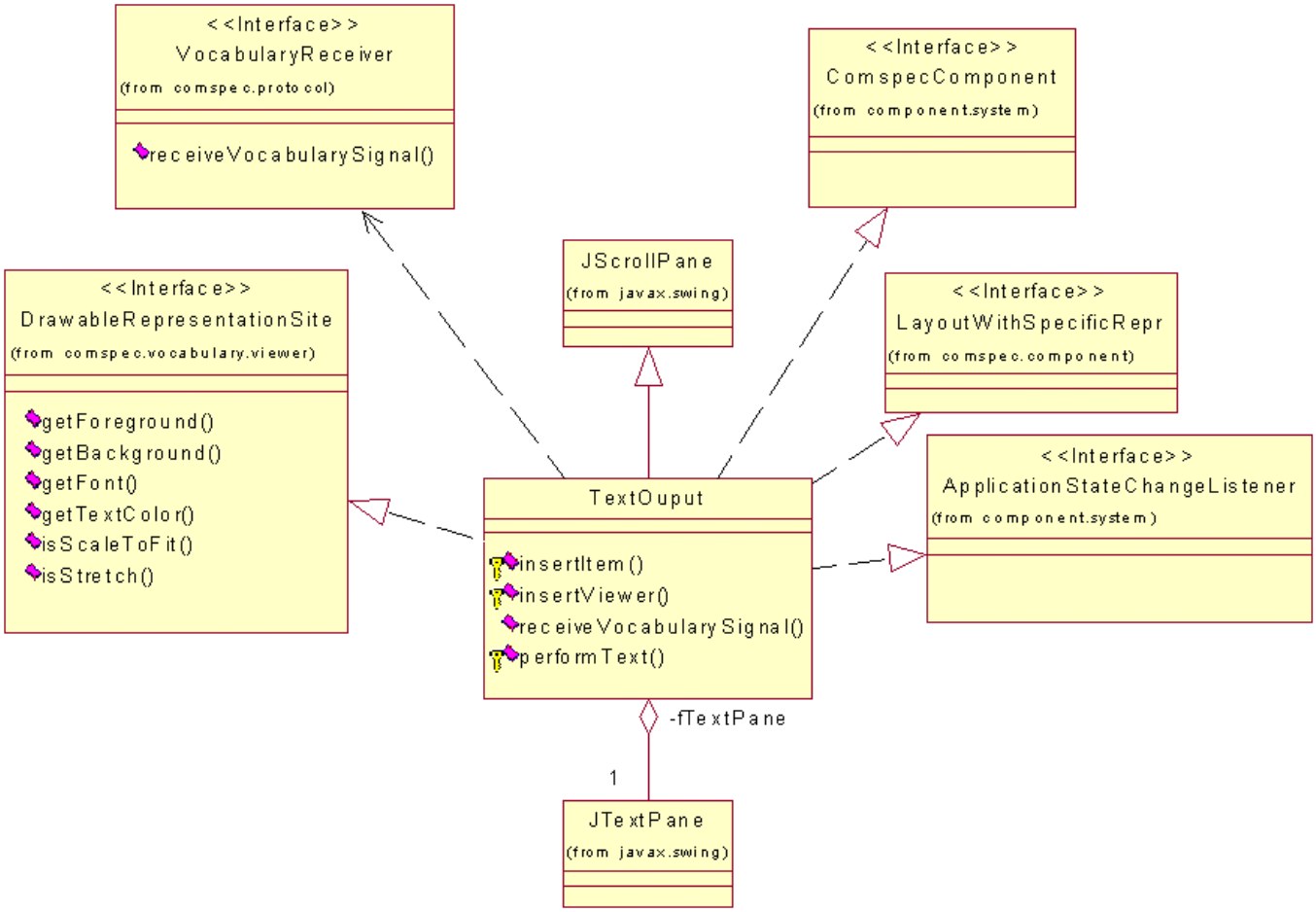
LayoutContainerSupport

The LayoutContainerSupport class is an abstract container class that implements the LayoutReceiver interface. This class can be used as a base class for layout containers that do not have any special requirements on the implementation of the LayoutReceiver interface.
The implementation takes care of the getting and setting of navigation links and actions, and also the coordination of the painting of any CursorIndicators currently being displayed on this component.
See also LayoutReceiver for more detailed comments on most methods implemented by this class, and LayoutComponentSupport for a corresponding class for non-container components



TextOutput

Below is an overview of the TextOutput component. TextOutput is a scrollpane, but the drawing is taken care of by the TextPane component.



5. Appendix A

5.1. Reference list

[1] Comspec on Java, TIDE Project No 1169, A. Hekstra, H. Klunder, E. Stav
[2] JavaBeans by Example, Prentice Hall, Henri Jubin and the Jalapeño Team
[3] ComLink User Manual – for Integrators and Facilitators, v 1.2, TP 1169, P. Head, D. Hekstra
[4] ComLink Sample Applications, TP 1169, M. Lundälv

6. Appendix B

6.1. SelectScan.java

```
package comspec.component.selectscan;

import comspec.component.ConfigComponentSupport;
import comspec.protocol.NavigationReceiver;
import comspec.protocol.ComspecCursor;
import comspec.protocol.VocabularySignal;
import comspec.component.InvalidConnectionSpecification;
import comspec.system.ApplicationStateChangeListener;
import comspec.system.ComspecVersion;
import comspec.component.ComspecComponentLinkSupport;
import comspec.system.*;
import comspec.component.Port;
import java.io.*;
import comspec.storage.StorageReader;
import comspec.storage.StorageWriter;
import comspec.storage.StorageRefResolver;

/**
 * <P><B>© Copyright 1999, the Comspec consortium.</B><P>
 *
 * This class implements the bean info of the select input component
 *
 * <pre>
 * Change History (most recent first):
 *
 * nr   DD/MM/YY  who  reason
 * (1)  20/09/99  PN   created
 * </pre>
 *
 * @author Peter Norrhall (PN), Linné Göteborg AB
 * @version 0.1
 *
 */


public class SelectScan extends ConfigComponentSupport implements
ApplicationStateChangeListener, NavigationReceiver {

  static final long serialVersionUID = -3095529781874974280L;

  private static final String defID = "SelectScan1";
  private static final String desc = "Config element: Select Scan";
  private static final ComspecVersion vers = new ComspecVersion(0, 1);
  public static final String NONE_SELECTED_STR = "- None Selected-";

  private ComspecComponentLinkSupport switchScan;
  private NavigationReceiver fSelectionSet;

  public SelectScan() {
    super();

    // add port definition: sending, single connections allowed SendPort
    addPort(Port.NAVIGATION_PROTOCOL, true, false);
    // add port definition: receiving, multiple connections allowed ReceivePort
    addPort(Port.NAVIGATION_PROTOCOL, false, true);
```

```
    switchScan = new ComspecComponentLinkSupport();
    fSelectionSet = null;

  }

  //*********************************************************************
  // ComspecComponent implementation
  // note: the Subsystem interface is an extension of the
  // ComspecComponent interface
  //*********************************************************************

  /*********************************************
  /* ComspecComponent method implementations
   */

  public void setComspecContext(ComspecContext bc) {
    ApplicationState appState;
    if (getComspecContext() != null) {
      // Unregister from application state found through context
      appState = (ApplicationState)
        getComspecContext().getService(ApplicationState.class, this);
      if (appState != null) {
        appState.removeApplicationStateChangeListener(this);
      }
    }
    // Set context through superclass's method
    super.setComspecContext(bc);

    if (bc != null) {
      // Register this component as listener to changes of application state
      appState = (ApplicationState) bc.getService(ApplicationState.class,
this);
      if (appState != null) {
        appState.addApplicationStateChangeListener(this);
      }
    }
  };

  /**
   * Get the default ID this component class wants to use.
   * @return the default ID, like "Component1"
   */
  public String getDefaultID() {
    return defID;
  }

  /** Get a string describing this component class in a way
    * understandable to the integrator/facilitator
    * @return a string containing a description of the component class.
    */
  public String getClassDescription() {
    return desc;
  }

  /** Get the current version of this component class
    */
  public ComspecVersion getClassVersion() {
    return vers;
  }

  //*********************************************************************
  // ComspecComponent implementation
  //*********************************************************************
```

```
  public boolean canCreateCursor(Class kindOfCursor, String componentID) {
    return ((getSwitchScan() != null) && (getSwitchScanID() == componentID) &&
       (getSelectionSet() != null) &&
(getSelectionSet().canCreateCursor(kindOfCursor, componentID)));
  }

  public ComspecCursor createCursor(Class kindOfCursor) {
    return fSelectionSet.createCursor(kindOfCursor);
  }

  public void destroyCursor(ComspecCursor cursor) {
    fSelectionSet.destroyCursor(cursor);
  }

  public void handleOutput(VocabularySignal signal) {
    fSelectionSet.handleOutput(signal);
  }

  //*****************************************************************
  // ApplicationStateChange implemnation
  //*****************************************************************

  private void setSelectionSet(NavigationReceiver theSelSet)
  {
    fSelectionSet = theSelSet;
  }

  private NavigationReceiver getSelectionSet()
  {
    if (fSelectionSet == null) {
      try {
        Object tmpObj =
this.getConnectedComponent(ConfigComponentSupport.SendPort, 0);
        if (tmpObj != null) {
          setSelectionSet((NavigationReceiver)tmpObj);
        }
      } catch (InvalidConnectionSpecification e) {}
    }
    return fSelectionSet;
  }

  public void applicationStateChanging(ApplicationStateChangeEvent e) throws
java.beans.PropertyVetoException {
  }

  public void applicationStateChanged(ApplicationStateChangeEvent evt) {
    int newState = evt.getNewValue();
    if (newState == ApplicationState.STARTING ) {
      try {
        Object tmpObj =
this.getConnectedComponent(ConfigComponentSupport.SendPort, 0);
        if (tmpObj != null) {
          setSelectionSet((NavigationReceiver)tmpObj);
        }
        tmpObj = this.getConnectedComponent(ConfigComponentSupport.ReceivePort,
0);
      } catch (InvalidConnectionSpecification e) {}
    }
    else if (newState == ApplicationState.STOPPING)
    {
      setSelectionSet(null);
    }
  }
```

```
  //*****************************************************************
  // Properties
  //*****************************************************************

  //*****************************************************************
  // SwitchScan Property
  //*****************************************************************

  public void setSwitchScan(comspec.component.ConfigComponent newSwitchScan) {
    // Använd interface istället för en klass
    comspec.component.ConfigComponent comp = getSwitchScan();
    boolean running = false;
    if (comp instanceof comspec.component.switchscan.SwitchScan) {
      if (((comspec.component.switchscan.SwitchScan)comp).isScanning()) {
        ((comspec.component.switchscan.SwitchScan)comp).stopScanning();
        running = true;
      }
    }
    switchScan.setComponent((ComspecComponent)newSwitchScan);
    if ((newSwitchScan != null) &&
        (newSwitchScan instanceof comspec.component.switchscan.SwitchScan) /*&&
(running)*/)

((comspec.component.switchscan.SwitchScan)newSwitchScan).restartScanning();
  }

  public comspec.component.ConfigComponent getSwitchScan() {
    return (comspec.component.ConfigComponent)switchScan.getComponent();
  }

  //*****************************************************************
  // SwitchScanID Property
  //*****************************************************************

  /** This method is used to get the ID of the switch scan
    * If no element is set NONE_SELECTED_STR
    * is returned
    */
  public String getSwitchScanID() {
    if (switchScan.getComponent() == null)
      return NONE_SELECTED_STR;
    else
      return switchScan.getComponent().getID();
  }

  /** This method is used to set the input device of the
    * selection set by ID.
    * Use NONE_SELECTED_STR to specify that no element
    * should be set.
    */
  public void setSwitchScanID(String theID) {
    String oldValue = getSwitchScanID();
    if (theID == NONE_SELECTED_STR)
      setSwitchScan(null);
    else {
      if (getComspecContext() == null)
        return;

      ObjectRegistry reg = (ObjectRegistry)getComspecContext().
         getService(ObjectRegistry.class, this);
      if (reg != null)

setSwitchScan((comspec.component.ConfigComponent)reg.findObject(theID));
```

```
    }
    firePropertyChange("switchScanID", oldValue, theID);
  }

  //****************************************************************
  // StorableComponent interface implementation
  //****************************************************************

  public void writeToStorage(StorageWriter writer) {
    try {
      super.writeToStorage(writer);

      ByteArrayOutputStream bst = new ByteArrayOutputStream();
      DataOutputStream dst = new DataOutputStream(bst);

      // write properties
      switchScan.writeToStorage(writer, dst);

      dst.flush();
      dst.close();

      writer.addPropertyVersion("SelectScan", "v1.0", bst.toByteArray());
    }
    catch (Exception e) {}
  }

  public void readFromStorage(StorageReader reader) {
    super.readFromStorage(reader);
    byte[] propData = reader.getPropertyVersion("SelectScan", "v1.0");
    if (propData == null)
      return ;
    try {
      ByteArrayInputStream bst = new ByteArrayInputStream(propData);
      DataInputStream dst = new DataInputStream(bst);

      // read properties, read links as references (int values)
      switchScan.readFromStorage(reader, dst);

      dst.close();
    }
    catch (Exception ex) {
    }
  }

  public void postReadFromStorage(StorageRefResolver resolver) {
    super.postReadFromStorage(resolver);
    // resolve reference links
    switchScan.postReadFromStorage(resolver);
  }

}
```

6.2. SelectScanBeanInfo.java

```
package comspec.component.selectscan;

import comspec.component.*;
import comspec.system.UserLevel;
import java.beans.*;
/**
 * <P><B>© Copyright 1999, the Comspec consortium.</B><P>
 *
```

```
 * This class implements the bean info of the select input component
 *
 * <pre>
 * Change History (most recent first):
 *
 * nr   DD/MM/YY  who  reason
 * (1)  20/09/99  PN   created
 * </pre>
 *
 * @author Peter Norrhall (PN), Linné Göteborg AB
 * @version 0.1
 *
 */

public class SelectScanBeanInfo extends ComspecBeanInfoSupport {

  public SelectScanBeanInfo() {
  }

  public PropertyDescriptor[] getPropertyDescriptors() {
    try {
      PropertyDescriptor[] props = new PropertyDescriptor[2];
      int x = 0;
      props[x++] = property(
        "ID",
        "Component ID",
        SelectScan.class,
        UserLevel.INTEGRATOR);
      props[x++] = property(
        "switchScanID",
        "Switch Scan",
        SelectScan.class,
        ScannerSelectionPE.class);

      return props;
    }
    catch (IntrospectionException e) {return super.getPropertyDescriptors();}
  }
}
```

6.3. ScannerSelectionPE.java

```
package comspec.component.selectscan;

import comspec.component.ConfigComponentSupport;
import comspec.protocol.NavigationReceiver;
import comspec.protocol.ComspecCursor;
import comspec.protocol.VocabularySignal;
import comspec.component.InvalidConnectionSpecification;
import comspec.system.ApplicationStateChangeListener;
import comspec.system.ComspecVersion;
import comspec.component.ComspecComponentLinkSupport;
import comspec.system.*;
import comspec.component.Port;
import java.io.*;
import comspec.storage.StorageReader;
import comspec.storage.StorageWriter;
import comspec.storage.StorageRefResolver;

/**
 * <P><B>© Copyright 1999, the Comspec consortium.</B><P>
 *
 * This class implements the bean info of the select input component
```

```
 *
 * <pre>
 * Change History (most recent first):
 *
 * nr   DD/MM/YY  who  reason
 * (1)  20/09/99  PN   created
 * </pre>
 *
 * @author Peter Norrhall (PN), Linné Göteborg AB
 * @version 0.1
 *
 */


public class SelectScan extends ConfigComponentSupport implements
ApplicationStateChangeListener, NavigationReceiver {

  static final long serialVersionUID = -3095529781874974280L;

  private static final String defID = "SelectScan1";
  private static final String desc = "Config element: Select Scan";
  private static final ComspecVersion vers = new ComspecVersion(0, 1);
  public static final String NONE_SELECTED_STR = "- None Selected-";

  private ComspecComponentLinkSupport switchScan;
  private NavigationReceiver fSelectionSet;

  public SelectScan() {
    super();

    // add port definition: sending, single connections allowed SendPort
    addPort(Port.NAVIGATION_PROTOCOL, true, false);
    // add port definition: receiving, multiple connections allowed ReceivePort
    addPort(Port.NAVIGATION_PROTOCOL, false, true);

    switchScan = new ComspecComponentLinkSupport();
    fSelectionSet = null;

  }

  //*****************************************************************
  // ComspecComponent implementation
  // note: the Subsystem interface is an extension of the
  // ComspecComponent interface
  //*****************************************************************

  /*********************************************
  /* ComspecComponent method implementations
   */

  public void setComspecContext(ComspecContext bc) {
    ApplicationState appState;
    if (getComspecContext() != null) {
      // Unregister from application state found through context
      appState = (ApplicationState)
        getComspecContext().getService(ApplicationState.class, this);
      if (appState != null) {
        appState.removeApplicationStateChangeListener(this);
      }
    }
    // Set context through superclass's method
    super.setComspecContext(bc);

    if (bc != null) {
```

```
      // Register this component as listener to changes of application state
      appState = (ApplicationState) bc.getService(ApplicationState.class,
this);
      if (appState != null) {
        appState.addApplicationStateChangeListener(this);
      }
    }
  };

  /**
   * Get the default ID this component class wants to use.
   * @return the default ID, like "Component1"
   */
  public String getDefaultID() {
    return defID;
  }

  /** Get a string describing this component class in a way
    * understandable to the integrator/facilitator
    * @return a string containing a description of the component class.
    */
  public String getClassDescription() {
    return desc;
  }

  /** Get the current version of this component class
    */
  public ComspecVersion getClassVersion() {
    return vers;
  }

  //*****************************************************************
  // ComspecComponent implementation
  //*****************************************************************

  public boolean canCreateCursor(Class kindOfCursor, String componentID) {
    return ((getSwitchScan() != null) && (getSwitchScanID() == componentID) &&
       (getSelectionSet() != null) &&
(getSelectionSet().canCreateCursor(kindOfCursor, componentID)));
  }

  public ComspecCursor createCursor(Class kindOfCursor) {
    return fSelectionSet.createCursor(kindOfCursor);
  }

  public void destroyCursor(ComspecCursor cursor) {
    fSelectionSet.destroyCursor(cursor);
  }

  public void handleOutput(VocabularySignal signal) {
    fSelectionSet.handleOutput(signal);
  }

  //*****************************************************************
  // ApplicationStateChange implemnation
  //*****************************************************************

  private void setSelectionSet(NavigationReceiver theSelSet)
  {
    fSelectionSet = theSelSet;
  }

  private NavigationReceiver getSelectionSet()
  {
```

```
    if (fSelectionSet == null) {
      try {
        Object tmpObj =
this.getConnectedComponent(ConfigComponentSupport.SendPort, 0);
        if (tmpObj != null) {
          setSelectionSet((NavigationReceiver)tmpObj);
        }
      } catch (InvalidConnectionSpecification e) {}
    }
    return fSelectionSet;
  }

  public void applicationStateChanging(ApplicationStateChangeEvent e) throws
java.beans.PropertyVetoException {
  }

  public void applicationStateChanged(ApplicationStateChangeEvent evt) {
    int newState = evt.getNewValue();
    if (newState == ApplicationState.STARTING ) {
      try {
        Object tmpObj =
this.getConnectedComponent(ConfigComponentSupport.SendPort, 0);
        if (tmpObj != null) {
          setSelectionSet((NavigationReceiver)tmpObj);
        }
        tmpObj = this.getConnectedComponent(ConfigComponentSupport.ReceivePort,
0);
      } catch (InvalidConnectionSpecification e) {}
    }
    else if (newState == ApplicationState.STOPPING)
    {
      setSelectionSet(null);
    }
  }


  //*****************************************************************
  // Properties
  //*****************************************************************

  //*****************************************************************
  // SwitchScan Property
  //*****************************************************************

  public void setSwitchScan(comspec.component.ConfigComponent newSwitchScan) {
    // Använd interface istället för en klass
    comspec.component.ConfigComponent comp = getSwitchScan();
    boolean running = false;
    if (comp instanceof comspec.component.switchscan.SwitchScan) {
      if (((comspec.component.switchscan.SwitchScan)comp).isScanning()) {
        ((comspec.component.switchscan.SwitchScan)comp).stopScanning();
        running = true;
      }
    }
    switchScan.setComponent((ComspecComponent)newSwitchScan);
    if ((newSwitchScan != null) &&
        (newSwitchScan instanceof comspec.component.switchscan.SwitchScan) /*&&
(running)*/)

((comspec.component.switchscan.SwitchScan)newSwitchScan).restartScanning();
  }

  public comspec.component.ConfigComponent getSwitchScan() {
    return (comspec.component.ConfigComponent)switchScan.getComponent();
```

```
  }

  //*************************************************************
  // SwitchScanID Property
  //*************************************************************

  /** This method is used to get the ID of the switch scan
    * If no element is set NONE_SELECTED_STR
    * is returned
    */
  public String getSwitchScanID() {
    if (switchScan.getComponent() == null)
      return NONE_SELECTED_STR;
    else
      return switchScan.getComponent().getID();
  }

  /** This method is used to set the input device of the
    * selection set by ID.
    * Use NONE_SELECTED_STR to specify that no element
    * should be set.
    */
  public void setSwitchScanID(String theID) {
    String oldValue = getSwitchScanID();
    if (theID == NONE_SELECTED_STR)
      setSwitchScan(null);
    else {
      if (getComspecContext() == null)
        return;

      ObjectRegistry reg = (ObjectRegistry)getComspecContext().
         getService(ObjectRegistry.class, this);
      if (reg != null)

setSwitchScan((comspec.component.ConfigComponent)reg.findObject(theID));
    }
    firePropertyChange("switchScanID", oldValue, theID);
  }

  //*************************************************************
  // StorableComponent interface implementation
  //*************************************************************

  public void writeToStorage(StorageWriter writer) {
    try {
      super.writeToStorage(writer);

      ByteArrayOutputStream bst = new ByteArrayOutputStream();
      DataOutputStream dst = new DataOutputStream(bst);

      // write properties
      switchScan.writeToStorage(writer, dst);

      dst.flush();
      dst.close();

      writer.addPropertyVersion("SelectScan", "v1.0", bst.toByteArray());
    }
    catch (Exception e) {}
  }

  public void readFromStorage(StorageReader reader) {
    super.readFromStorage(reader);
    byte[] propData = reader.getPropertyVersion("SelectScan", "v1.0");
```

```
    if (propData == null)
      return ;
    try {
      ByteArrayInputStream bst = new ByteArrayInputStream(propData);
      DataInputStream dst = new DataInputStream(bst);

      // read properties, read links as references (int values)
      switchScan.readFromStorage(reader, dst);

      dst.close();
    }
    catch (Exception ex) {
    }
  }

  public void postReadFromStorage(StorageRefResolver resolver) {
    super.postReadFromStorage(resolver);
    // resolve reference links
    switchScan.postReadFromStorage(resolver);
  }

}
```

6.4. ComboboxControl.java

```
package comspec.component.comboboxcontrol;

import comspec.component.*;
import comspec.component.toolwidgets.*;

import comspec.system.IDAlreadyUsedException;
import comspec.system.ComspecContext;
import comspec.system.ComspecContextChild;
import comspec.system.IDChangeListener;
import comspec.system.ComspecVersion;
import comspec.system.ComspecComponent;
import comspec.system.ToolComponent;

import comspec.storage.*;

import java.awt.*;
import java.awt.Choice;
import java.io.*;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

import java.beans.Introspector;
import java.beans.BeanInfo;
import java.beans.IntrospectionException;
import java.beans.PropertyDescriptor;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.beans.PropertyEditor;

import javax.swing.JComboBox;
import javax.swing.ComboBoxModel;
import javax.swing.DefaultComboBoxModel;

import java.util.Vector;

/**
```

```
 * <P><B>© Copyright 1999, Comspec consortium.</B><P>
 *
 * ComboboxControl is a Comspec toolbar component for the adjustment
 * of tags properties.
 *
 * <pre>
 * Change History (most recent first):
 *
 * nr   DD/MM/YY  who  reason
 * (1)  20/09/99  PN   created
 * </pre>
 *
 * @author Peter Norrhall (PN), Linné Göteborg AB
 * @version 0.1
 *
 */

public class ComboboxControl
//       extends Choice
       extends JComboBox
       implements PropAdjuster,
                  ComspecComponent,
                  PropertyChangeListener,
                  ToolComponent,
                  ActionListener {

  static final long serialVersionUID = 4101302855308681596L;

  private static String defID = new String("ComboboxControl1");
  private static String desc = new String(
    "ComboboxControl: a toolbar component for adjusting multiple string
selection properties");
  private static ComspecVersion vers = new ComspecVersion(0,1);

  private PropAdjusterSupport fPropAdjuster; //todo: use service
  private ComspecComponentSupport fComspecComponent;
  private boolean fPropertyStored = true;

  public ComboboxControl(ComboBoxModel aModel) {
    super(aModel);
    this.init();
  }

  public ComboboxControl(final Object items[]) {
    super(new DefaultComboBoxModel(items));
    this.init();
  }

  public ComboboxControl(Vector items) {
    super(new DefaultComboBoxModel(items));
    this.init();
  }

  public ComboboxControl() {
    super(new DefaultComboBoxModel());
    this.init();
  }

  private void init() {
    fPropAdjuster = new PropAdjusterSupport(this);
    fComspecComponent = new ComspecComponentSupport(this);

    addActionListener(this);
    setLightWeightPopupEnabled(false);
```

```
  }

//*****************************************************************
//* Combobox overrides
//*****************************************************************

  public void setSelectedItem(Object anObject) {
    super.setSelectedItem(anObject);
    // send the new state to the linked component's property
    try {
      fPropAdjuster.setPropertyValue(anObject);
    }
    catch (Exception ex) {
    }
  }

//*****************************************************************
//* This section implements the PropAdjuster interface
//* The calls are delegated to the support component
//*****************************************************************

  /**
   * Whether the property should be stored in the user profile or not.
   * This should default to true, but the integrator should be able to
   * turn this off.<P>
   * Note: if multiple ToolComponents link to the same TargetComponent
   * and TargetProperty, the Run-time environment may use the setting
   * of another control.
   * @returns true of this needs to be stored.
   */
  public boolean isPropertyStored() {
    return fPropertyStored;
  }

  public void setPropertyStored(boolean stored) {
    boolean oldValue = fPropertyStored;
    fPropertyStored = stored;
    firePropertyChange("propertyStored", oldValue, stored);
  }

  /**
   * Get the ComspecComponent this control accesses a property of.
   * @returns the target ComspecComponent, or null if not linked.
   */
  public ComspecComponent getTargetComponent() {
    if (fPropAdjuster != null)
      return fPropAdjuster.getTargetComponent();
    return null;
  }

  public void setTargetComponent(ComspecComponent comp) {
    if (fPropAdjuster != null) {
      try {
        getTargetComponent().removePropertyChangeListener(this);
      }
      catch (Exception e) {
      }
      fPropAdjuster.setTargetComponent(comp);
      try {
        comp.addPropertyChangeListener(this);
      }
      catch (Exception ex) {
      }
    }
```

```
  }

  public String getTargetComponentID() {
    if (fPropAdjuster != null)
      return fPropAdjuster.getTargetComponentID();
    return "";
  }

  public void setTargetComponentID(String theID) {
    if (fPropAdjuster != null) {
      try {
        getTargetComponent().removePropertyChangeListener(this);
      }
      catch (Exception e) {
      }
      String oldValue = getTargetComponentID();
      fPropAdjuster.setTargetComponentID(theID);
      firePropertyChange("targetComponentID", oldValue, theID);
      try {
        getTargetComponent().addPropertyChangeListener(this);
      }
      catch (Exception ex) {
      }
    }
  }

  /**
   * Get the name of the property in the target ComspecComponent
   * this control accesses.
   * @returns the target property name, or null if not linked.
   */

  public String getTargetProperty() {
    if (fPropAdjuster != null)
      return fPropAdjuster.getTargetProperty();
    return "";
  }

  public void setTargetProperty(String prop) {
    String oldValue = getTargetProperty();
    fPropAdjuster.setTargetProperty(prop);
    firePropertyChange("targetProperty", oldValue, prop);
    updateValue();
  }


  protected void updateList() {
    try {
      super.removeAllItems();
    } catch (Exception e) {
    };
    //
    PropertyDescriptor properties[];
    try {
      if (getTargetComponent() == null)
        return;
      BeanInfo bi = Introspector.getBeanInfo(getTargetComponent().getClass());
      properties = bi.getPropertyDescriptors();
    }
    catch (IntrospectionException ex) {
      return;
    }

    String property = getTargetProperty();
```

```
    String[] tags = null;
    try {
      // check if there are any int-properties
      for (int i = 0; i < properties.length; i++) {
        PropertyDescriptor pd = properties[i];
        if (pd.isExpert() || pd.isHidden())
          continue;
        if (property.equals(pd.getName())) {
          Class pe = pd.getPropertyEditorClass();
          PropertyEditor propEditor = null;
          if ((pe != null)) {
//          if ((pe != null) && (pe instanceof PropertyEditorSupport)) {
            try {
              propEditor = (PropertyEditor)pe.newInstance();
            }
            catch (Exception ex) {
              // Drop through.
            }
            if (propEditor != null) {
              if ((propEditor instanceof ComspecContextChild) &&
                 (((ComspecContextChild)propEditor).getComspecContext() ==
null))

((ComspecContextChild)propEditor).setComspecContext(getComspecContext());
              tags = propEditor.getTags();
            }
          }
        }
      }
    }
    catch (Exception e) {
    }

    // Fill the list
    if (tags != null) {
      for (int i = 0; i < tags.length; i++) {
        addItem(tags[i]);
      }
    }
  }


  protected void updateValue() {
    try {
      // show current state of linked property
      Object o = getSelectedItem();
      Object n = (Object)fPropAdjuster.getPropertyValue();
      if (! n.equals(o)) {
        updateList();
        super.setSelectedItem(n);
      }
    }
    catch (Exception e) {
    }
  }

  public Object getPropertyValue() throws Exception {
    if (fPropAdjuster != null)
      return fPropAdjuster.getPropertyValue();
    return null;
  }

  public void setPropertyValue(Object o) throws Exception {
    if (fPropAdjuster != null)
```

```
      fPropAdjuster.setPropertyValue(o);
  }

//*************************************************************
//* This section implements the ComspecComponent interface
//* The calls are delegated to the support component
//*************************************************************

  public void setComspecContext(ComspecContext bc) {
    if (fComspecComponent != null) {
      fComspecComponent.setComspecContext(bc);
      if (bc != null)
        updateValue();
//        updateList();
    }
  }

  public ComspecContext getComspecContext() {
    if (fComspecComponent != null)
      return fComspecComponent.getComspecContext();
    return null;
  }

  public String getDefaultID() {
    return defID;
  }

  public String getClassDescription() {
    return desc;
  }

  public ComspecVersion getClassVersion() {
    return vers;
  }

  /**
   * Initialize the object. This method is called during
   * de-serialization.
   * This default implementation does nothing.
   */
  public void initialize() {
  }

  public void setID(String newID) throws IDAlreadyUsedException {
    if (fComspecComponent != null)
      fComspecComponent.setID(newID);
  }

  public String getID() {
    if (fComspecComponent != null)
      return fComspecComponent.getID();
    return null;
  }

  public void addIDChangeListener(IDChangeListener l) {
    if (fComspecComponent != null)
      fComspecComponent.addIDChangeListener(l);
  }

  public void removeIDChangeListener(IDChangeListener l) {
    if (fComspecComponent != null)
      fComspecComponent.removeIDChangeListener(l);
  }
```

```
  public void addPropertyChangeListener(PropertyChangeListener l) {
    if (fComspecComponent != null)
      fComspecComponent.addPropertyChangeListener(l);
  }

  public void removePropertyChangeListener(PropertyChangeListener l) {
    if (fComspecComponent != null)
      fComspecComponent.removePropertyChangeListener(l);
  }

  public void firePropertyChange(String propertyName, Object oldValue,
    Object newValue) {

    if (fComspecComponent != null)
      fComspecComponent.firePropertyChange(propertyName, oldValue, newValue);
  }

//***************************************************************
//* This section implements the PropertyChangeListener interface
//***************************************************************

  public void propertyChange(PropertyChangeEvent evt) {
    try {
      if (evt.getPropertyName().equalsIgnoreCase(getTargetProperty())) {
        Object o = evt.getOldValue();
        Object n = evt.getNewValue();
        if (o != n)
          // show new state of linked property
          super.setSelectedItem(n);
      }
    }
    catch (Exception e) {
    }
  }


  public void actionPerformed(ActionEvent evt) {
    try {
      if (getSelectedItem() != null)
        setPropertyValue(getSelectedItem());
    }
    catch (Exception ex) {
      ex.printStackTrace(System.out);
    }
  }

  //***************************************************************
  // StorableComponent interface implementation
  //***************************************************************

  private static final String storageName = "ComboboxControl";
  private static final String storageVersion = "v1.0";

  public void writeToStorage(StorageWriter writer) {
    StorageSupport.writeComponent(writer, this);
    try {
      ByteArrayOutputStream bst = new ByteArrayOutputStream();
      DataOutputStream dst = new DataOutputStream(bst);

      // write properties
//      StorageSupport.writeString(dst, getText());
      dst.writeBoolean(fPropertyStored);
      fPropAdjuster.writeToStorage(writer, dst);
```

```
      fComspecComponent.writeToStorage(writer, dst);

      dst.flush();
      dst.close();

      writer.addPropertyVersion(storageName, storageVersion,
bst.toByteArray());
    }
    catch (Exception e) {}
  }

  public void readFromStorage(StorageReader reader) {
    StorageSupport.readComponent(reader, this);
    byte[] propData = reader.getPropertyVersion(storageName, storageVersion);
    if (propData == null)
      return ;
    try {
      ByteArrayInputStream bst = new ByteArrayInputStream(propData);
      DataInputStream dst = new DataInputStream(bst);

      // read properties, read links as references (int values)
//      super.setText(StorageSupport.readString(dst));
      fPropertyStored = dst.readBoolean();
      fPropAdjuster.readFromStorage(reader, dst);
      fComspecComponent.readFromStorage(reader, dst);

      dst.close();
    }
    catch (Exception ex) {
    }
  }

  public void postReadFromStorage(StorageRefResolver resolver) {
    // resolve reference links
    fPropAdjuster.postReadFromStorage(resolver);
//    updateValue();
  }

}
```

6.5. ComboboxControlBeanInfo.java

```
package comspec.component.comboboxcontrol;

import comspec.component.*;
import java.beans.*;
import comspec.component.ComspecBeanInfoSupport;
import comspec.system.UserLevel;
import comspec.support.editor.*;

/**
 * <P><B>© Copyright 1999, Comspec consortium.</B><P>
 *
 * ComboboxControlBeanInfo is the BeanInfo for ComboboxControls. It
 * reports the properties for the various user levels and sets
 * property editors for some properties.
 *
 * <pre>
 * Change History (most recent first):
 *
 * nr   DD/MM/YY  who  reason
 * (1)  20/09/99  PN   created
 * </pre>
```

```
 *
 * @author Peter Norrhall (PN), Linné Göteborg AB
 * @version 0.1
 *
 */

public class ComboboxControlBeanInfo extends ComspecBeanInfoSupport {

//  public BeanDescriptor getBeanDescriptor() {
//    return new BeanDescriptor(ComboboxControl.class,
ComboboxControlCustomizer.class);
//  }


  public PropertyDescriptor[] getPropertyDescriptors() {
    try {
      PropertyDescriptor[] props = new PropertyDescriptor[7];
      int x = 0;
      props[x++] = property(
        "ID",
        "Component ID",
        ComboboxControl.class,
        UserLevel.INTEGRATOR);
      props[x++] = property(
        "foreground",
        "Foreground Colour",
        ComboboxControl.class,
        ColorPropertyEditor.class);
      props[x++] = property(
        "background",
        "Background Colour",
        ComboboxControl.class,
        ColorPropertyEditor.class);
      props[x++] = property(
        "font",
        "Font",
        ComboboxControl.class);
      props[x++] = property(
        "targetComponentID",
        "Target Component",
        ComboboxControl.class,
        TargetComponentPE.class,
        UserLevel.INTEGRATOR);
      props[x++] = property(
        "targetProperty",
        "Target Property",
        ComboboxControl.class,
        comspec.support.editor.BoundStringTagPropertyPE.class,
        UserLevel.INTEGRATOR);
      props[x++] = property(
        "propertyStored",
        "Store Property",
        ComboboxControl.class,
        UserLevel.INTEGRATOR);

      return props;
    }
    catch (IntrospectionException e) {
      return super.getPropertyDescriptors();
    }
  }


}
```

6.6. BoundConfigComponentPE

```
package comspec.component.comboboxcontrol;

import comspec.component.ConfigComponent;
import comspec.support.editor.*;
import java.beans.*;
import java.util.Vector;

/**
 * <P><B>© Copyright 1999, Comspec consortium.</B><P>
 *
 * The BoundConfigComponentPE edits String properties that represent
 * ConfigComponent properties. It only shows the properties of the
 * indicated target.
 *
 * <pre>
 * Change History (most recent first):
 *
 * nr   DD/MM/YY  who  reason
 * (1)  20/09/99  PN   created
 * </pre>
 *
 * @author Peter Norrhall (PN), Linné Göteborg
 * @version 0.1
 *
 */

public class BoundConfigComponentPE extends BoundPropertyTypePE {

  public BoundConfigComponentPE() {
  }

  protected Vector getTypedProperty() {
    // get all properties of the component to adjust
    PropertyDescriptor properties[];
    try {
      BeanInfo bi =
Introspector.getBeanInfo(target.getTargetComponent().getClass());
      properties = bi.getPropertyDescriptors();
    }
    catch (IntrospectionException ex) {
      return null;
    }

    Vector v = new Vector(5);
    try {
      // check if there are any int-properties
      for (int i = 0; i < properties.length; i++) {
        PropertyDescriptor pd = properties[i];
        if (pd.isExpert() || pd.isHidden())
          continue;
        java.lang.reflect.Method getter = pd.getReadMethod();
        Object value = null;
        try {
          Object args[] = { };
          value = getter.invoke(target.getTargetComponent(), args);
          if (value instanceof ConfigComponent)
            v.addElement(pd.getName());
        }
        catch (Exception ex) {
```

```
        }
      }
    }
    catch (Exception e) {
    }
    return v;
  }


}
```

6.7. BoundStringTagPropertyPE

```
package comspec.support.editor;

import java.beans.*;
import java.util.Vector;

/**
 * <P><B>© Copyright 1998, Handicom, NL, and the Comspec consortium.</B><P>
 *
 * The BoundStringTagPropertyPE edits String properties that represent
 * String properties with a property editor supporting the
 * PropertyEditor.getTags methods.
 * It only shows the properties of the indicated target.
 *
 * <pre>
 * Change History (most recent first):
 *
 * nr   DD/MM/YY  who  reason
 * (1)  22/09/99  PN   created
 * </pre>
 *
 * @author Peter Norrhall (PN), Linné Göteborg
 * @version 0.1
 *
 */
public class BoundStringTagPropertyPE
  extends BoundPropertyTypePE {

  public BoundStringTagPropertyPE() {
  }

  protected Vector getTypedProperty() {
    // get all properties of the component to adjust
    PropertyDescriptor properties[];
    try {
      BeanInfo bi =
Introspector.getBeanInfo(target.getTargetComponent().getClass());
      properties = bi.getPropertyDescriptors();
    }
    catch (IntrospectionException ex) {
      return null;
    }

    Vector v = new Vector(15);
    try {
      // check if there are any int-properties
      for (int i = 0; i < properties.length; i++) {
        PropertyDescriptor pd = properties[i];
        if (pd.isExpert() || pd.isHidden())
          continue;
```

```
        java.lang.reflect.Method getter = pd.getReadMethod();
        Object value = null;
        try {
          Object args[] = { };
          value = getter.invoke(target.getTargetComponent(), args);
//          if (value instanceof String) {
          if (value != null) {
            Class pe = pd.getPropertyEditorClass();
            PropertyEditor propEditor = null;
            if ((pe != null)) {
              try {
                propEditor = (PropertyEditor)pe.newInstance();
              }
              catch (Exception ex) {
                // Drop through.
              }
              if (propEditor != null) {
                String[] tags;
                tags = propEditor.getTags();
                if (tags.length > 0)
                  v.addElement(pd.getName());
              }
            }
          }
        }
        catch (Exception ex) {
        }
      }
    }
    catch (Exception e) {
    }
    return v;
  }

}
```