UILOGIC®

DUIL Reference Manual

Ver1.2

UILOGIC, Inc.

March 2005

1 Introduce

DUIL (Dynamic User Interface logic) is an easy and rapid JAVA UI solution provided by uilogic, which will help you to reduce the development cost and period of your product.

DUIL can generate the UI from the XML-based UI definition file, that is like the role of windows application's *.rc file, furthermore, DUIL provide a mechanism for the interactive-support of components, that will enable you build relationship among components easier than the traditional way.

Feature:

- Dynamically relationship among components.
- Abstract the UI information from traditional java code, it enable you may build your GUI and action implementation independently.
- I18n support.
- Simplify the I/O process; you can get/set the value of UI component very easy.
- Support user-defined component, you can integrated your java component into DUIL.
- Support user-defined tag.
- UI defined file support include primitive.
- UI Bridge support, you can build relation deferent individual GUI, such as there are two individual dialogs that create with DUIL, then the sub components of the two dialogs can be related.

DUIL apply the beanShell as the java script interpreter, for more detail about beanshell please refers to the http://www.beanshell.org

2 Usage

DUIL have integrated JDK's swing component, you can use them directly, and also you can extend DUIL to support user-defined components.

The following JDK components are integrated in DUIL already:

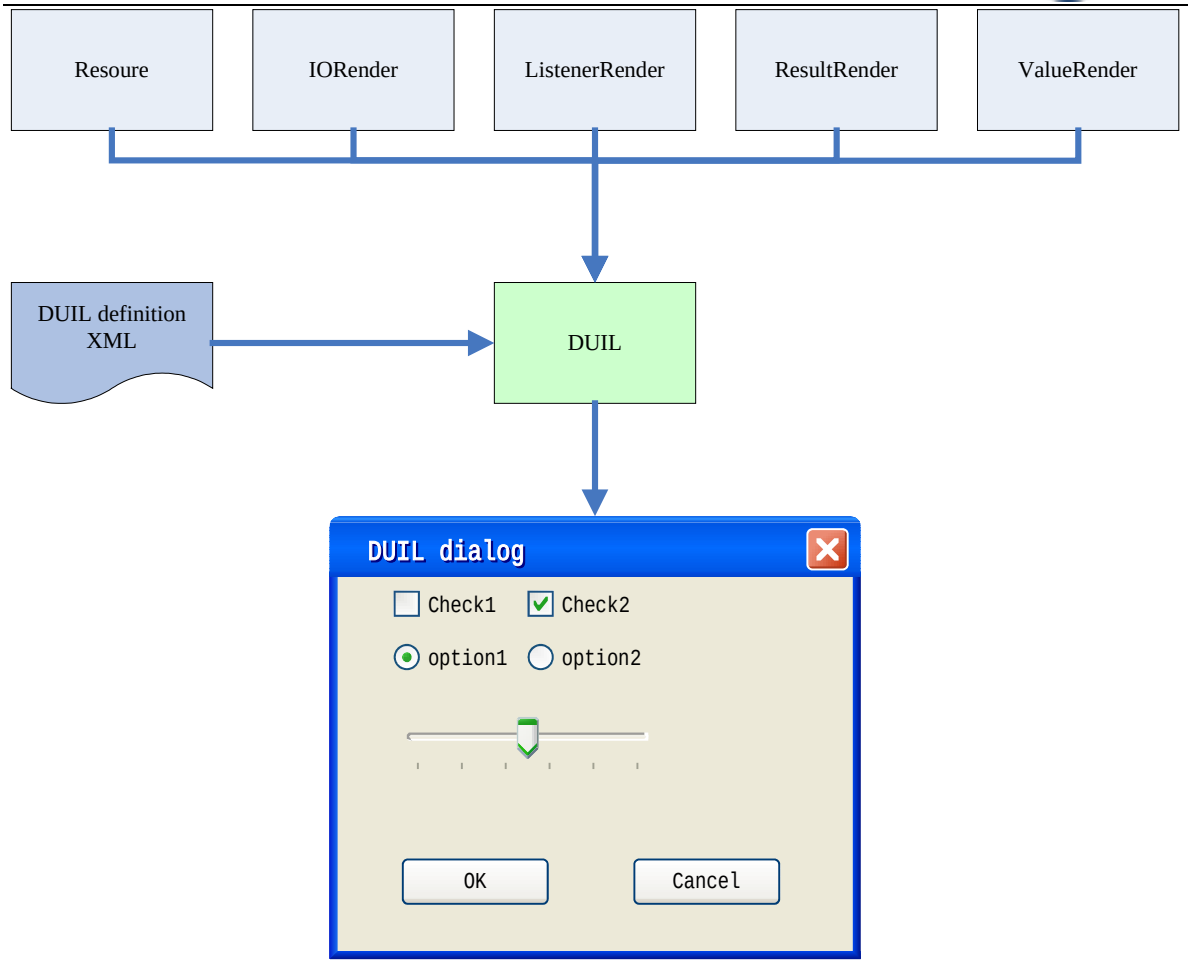
- JButton
- JCheckBox
- JColorChooser
- JCombo
- JDesktopPane
- JEditorPane

- JFormattedTextField
- JInternalFrame
- JLabel
- JLayeredPane
- JList
- JOptionPane
- JPanel
- JPasswordField
- JProgressBar
- JRadioButton
- JRadioContainer
- JRootPane
- JScrollBar
- JScrollPane
- JSlider
- JSpinner
- JSplitPane
- JTabbedPane
- JTable
- JTextArea
- JTextComponent
- JTextField
- JTextPane
- JToggleButton
- JToolBar
- JToolTip
- JTree
- JViewport
- JMenu
- JMenuBar
- JMenuItem
- JSeparator

Furthermore, DUIL provide additional components for user.

- JFileChooserButton
- ColorButton
- FontChooser
- RadioGroup

The following flow chart shows the DUIL's mechanism.



graph 1 DUIL mechanism diagram

DUIL will generate the JAVA GUI base on the DUIL definition XML and Renders. The Renders is used to provide information when create GUI.

- IORender: it is used to load/save UI component's value from/to external.
- ListenerRender: it is used to provide the Listener for specific component.
- ResultRender: it is used to provide the method to calculate the result value for specific component.
- Resource: it is used to specify the resource base for i18n support.
- ValueRender: it is used to provide the User Interface for specific component.

2.1 Get Start

First of all, let's start with a simple sample.

```
<JPanel layout="=new BorderLayout()" id="demoCommon">
    <JPanel layout="=new BoxLayout($(this),BoxLayout.Y_AXIS )">
        <JTextField id="welcome" text="" enabled="" background="#000000"/>
```

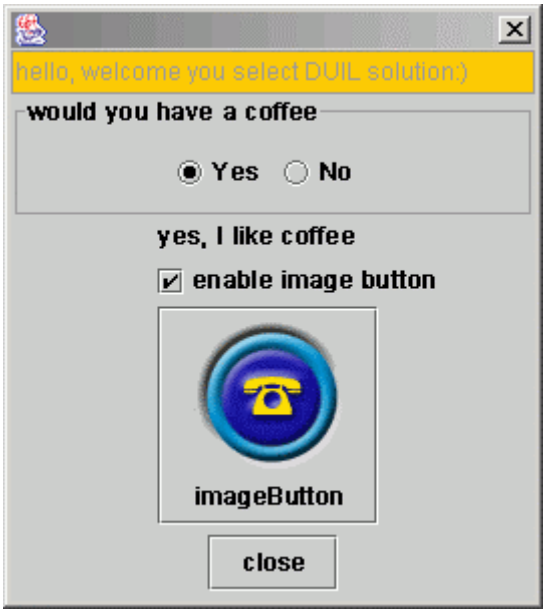
```
        <RadioGroup id="choice" name="choice" border="true" text="would you
have a coffee">
            <JRadioButton name="yes" text="Yes" toolTipText="yes, I like coffee"
selected="true" />
            <JRadioButton name="no" text="No" toolTipText="no, I like tea" />
        </RadioGroup>
        <JLabel
text="=$(parent.yes).isSelected()?$(parent.yes).getToolTipText():$(parent.no).ge
tToolTipText()"/>
        <JCheckBox name="chk" text="enable image button" selected="true"/>
        <JButton text="imageButton"  enabled="=$(parent.chk).isSelected()"
            verticalTextPosition="BOTTOM" horizontalTextPosition="CENTER"
            disabledIcon="images/b1d.gif"                    icon="images/b1.gif"
pressedIcon="images/b1p.gif" rolloverIcon="images/b1r.gif"/>
    </JPanel>
    <JPanel layoutContrains="SOUTH" >
        <JButton text="close" action="CLOSE_WINDOW"/>
    </JPanel>
</JPanel>
```

It is a piece of UI definition XML file, it contain a dialog, which id is “choice”, choice dialog contain the following components:

- A radio with three choices.
- Two buttons: ok and cancel.

If you click the ok, the dialog value will be saved (refer to the IORender) to IORender and then close the dialog, click “cancel” to close the dialog without save result.

DUIL will generate it to the following dialog.



graph 2

Please refer to the demoCommon for detail.
You can use the duilBrowser to view your XML UI definition.

2.2 DUIL definition format

The resource definition file is used to define the dialogs; you can provide the **bean attribute** for component in resource file, and also you can specific a DUIL script for a bean attribute.
Format:

```
<comTag beanAttr1=" value1" beanAttr2=" Value2"/>
```

comTag: the component tag, it specific the java component, such as JRadioButton/JCombo.
beanAttr1: the attribute of comtag specific component.

Tips:

- The attribute must conform to the java bean standard or attribute that DUIL provided.
- The attribute's value must be wrapper in "", such as beanAttr1="value1", not beanAttr1=value1.
- If the attributes start with one '=', DUIL will regard it as DUIL script, for detail please refer to the Script.

2.3 Component attribute

2.3.1 General attributes

Each component has several attributes (java bean attribute) that can be used in the resource file.
In the next phase, I will show an example for JButton.

JButton have the following bean attribute (refer to the J2SE API Specification for detail).

ATTRIBUTE	TYPE
text	String
pressedIcon	Icon
rolloverIcon	Icon
ToolTipText	String

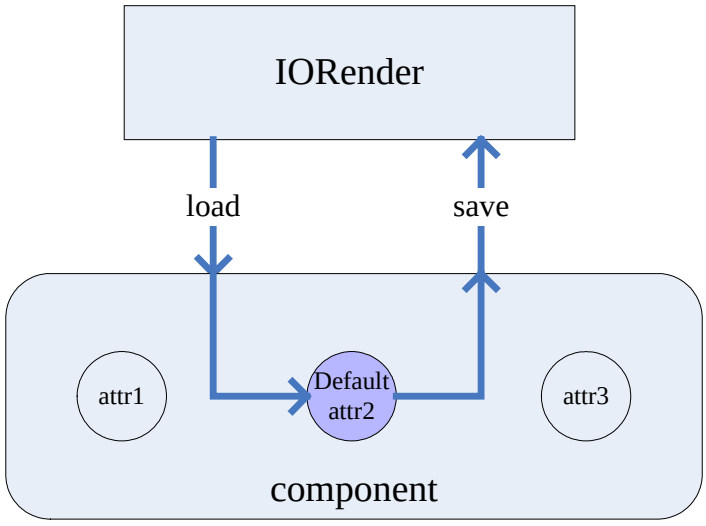
Then you can define it as following:

```
<JButton id="ok_btn" text="ok" pressedIcon="iamges/press.gif"
rolloverIcon="images/rollover.gif" toolTipText="I am a OK button"/>
```

2.3.2 Primary attribute

A component may have more than one attributes, but only one attribute can be set as primary attribute, the component's primary attribute will accept the default value from IORender, and store the primary attribute's value to IORender when save dialog.

The following diagram shows this logic.



graph 3 Load/save flow with primary attribute

For example:

JTextField's primary attribute is "text"

```
<JTextField id="userName" text="carfield" toolTipText="please input the username"/>
<JTextField id="password" text="" toolTipText="please input the password"/>
```

So we can use the following code to provide the initial value for userName.

```
CompositeRenders renders = new CompositeRenders();
DefaultIORender ioRender = new DefaultIORender();
renders.setIORender(ioRender);
render.save("userName","I am a cat");
```

Then the DUIL will assign the "I am a cat" to the userName's **"text"** attribute.

DUIL have specific some component's primary attribute to simplify the

COMPONENT	PRIMARY ATTRIBUTE
JButton	selected
JCheckBox	selected
JColorChooser	color
JCombo	id
JDesktopPane	N/A
JEditorPane	text
JFormattedTextField	text
JInternalFrame	N/A
JLabel	text
JLayeredPane	N/A
JList	id
JOptionPane	N/A
JPanel	N/A
JPasswordField	text
JProgressBar	value
JRadioButton	selected
JRootPane	N/A
JScrollBar	value
JScrollPane	N/A
JSlider	value
JSpinner	value
JSplitPane	N/A
JTabbedPane	N/A
JTable	N/A
JTextArea	text
JTextComponent	text
JTextField	text

JTextPane	text
JToggleButton	selected
JToolBar	N/A
JToolTip	tipText
JTree	N/A
JViewport	N/A
FileButton	files
FontChooser	font
RadioGroup	id
ColorButton	color

N/A: it means the component have no primary attribute.

Furthermore, You can also change the above default primary attribute setting for each component by put the '@' before the attribute name.

For example, we want to set "visible" as the JButton's primary attribute, the definition can write as following.

```
<JButton id="ok_btn" text="ok" pressedIcon="iamges/press.gif"
rolloverIcon="images/rollover.gif" toolTipText="I am a OK button"
@visible="true"/>
```

2.3.3 Additional attributes provided by DUIL

Besides the component's bean attribute, DUIL provide the following additional attributes that can be used in resource file.

ATTRIBUTE	TYPE	VALUE
id	String	The component's id
language	String	The dialog's language
layoutContrains	Object	The attribute will be used when add component to parent.
action	String	It can be the combination of following value (separate with ;): ➢ APPLY ➢ HIDEN_WINDOW ➢ CLOSE_WINDOW

2.3.4 I18n attribute

You can use the DUIL's i18n property for your component's attribute by put '$' before it.
For example:

```
<JButton id="ok_btn" $text="ok" pressedIcon="iamges/press.gif"
rolloverIcon="images/rollover.gif" toolTipText="I am a OK button"
@visible="true"/>
```

Then the ok_btn's **"text"** will support Internationalization.
DUIL will load the "ok_btn.text" from specific language resource as ok_btn's text value.
For example, If you specific the dialog's language=" fr_CA", then DUIL will load the string value of "ok_btn.text" from XXX_fr_CA.properties.

2.4 DUIL Script

If the attribute's value is start with '=', DUIL will regard it as script.
For example:

```
<Dialog name="testDlg" layout="=new BorderLayout()">
    <JPanel layoutContrains="CENTER"/>
    ...
   </JPanel>
</Dialog>
```

When DUIL process the testDlg, a BorderLayout instance will be created and assign to testDlg's layout attribute

2.5 DUIL variable

You can use the DUIL variables in the script, it will enable you can build a relationship between UI component.

DUIL variable format:

```
$(component_express[$prop_express])
```

the **component_express** compose of the following reserved keyword (separate with dot ('.')):

- this
- parent

- root
- user
- bridgeRoot
- env

this

"this" represents the current component, such as

```
<JPanel layout="=new BoxLayout($(this),BoxLayout.Y_AXIS )">
  ...
</JPanel>
```

The layout attribute's value is a DUIL script that contain the DUIL variable "$(this)" in the above definition, $(this) is represent the JPanel instance.

parent

The means of **"parent"** and **"root"** may be deduced by the same analogy as **"this"**, **"parent"** represent the current component's parent, the **"root"** represent the top component of current dialog.

Please refer to the demoCommon.

bridgeRoot

It is represent the bridge of current dialog, if the current dialog has not add to a bridge then the "bridgeRoot" equals to the "root"

Please refer to the demoBridge.

user

It is user-defined extendable interface (ValueRender) for user., such as

```
$(parent.user)
```

This variable represent the parent component's ValueRender, furthermore you can pass parameter to ValueRender in variable expression.

```
$(parent.user.param1)
```

DUIL will return parent component's ValueRender.getValue(param1) as the variable's value.

Please refer to the demoBrowser

env

It means the external environment, it enable user can use the environment in DUIL script.

Please refer to the I18n

the **prop_express** is seperated by '.', it is used to define the bean attribute of component.

For example:

```
<JPanel>
    <JTextField name="editor"/>
    <JLabel text="=$(parent.editor)">
</JPanel>
```

According the above UI definition:

$(root.editor) means the JTextField component.
$(root.editor$text) means the JTextField's text attribute, it is same with the $(root.editor).getText()
$(root.editor$document.length) means the JTextField document's length attribute, it is same with the $(root.editor).getDocument().getLength();

2.6 UI initialized

2.6.1 Initialize attribute

The initial value will be stored in IORender, you can provide your value to specific component's specific attribute.

Give a value for specific component.

```
DefaultIORender ioRender = new DefaultIORender();
ioRender.save("welcome", "hello, welcome you select DUIL solution:)");
```

Give a value for specific component's attribute
Code

```
ioRender.save("welcome.enabled",Boolean.FALSE);
ioRender.save("welcome.backgroud",Color.orange);
```

DUIL definition

```
<JTextField id="welcome" text="" enabled="" background="#000000"/>
```

After running, the welcome's background will be replace with Color.orange

Note: you must define the attributes in DUIL definition file if you want provide value for it through IORender, in this sample, the "enable" and "background" must be defined.

Please refer to demoCommon for detail.

2.6.2 Install listener

You can install a listener to your component through DefaultListenerRender, this example will show how to provide an ActionListener for your button.

Code:

```
DefaultListenerRender listenerRender = new DefaultListenerRender();
listenerRender.addListener("closeAction", new yourAction ());
```

DUIL definition

```
<JButton text="close" actionListener="closeAction"/>
```

Please refer to demoBridge for detail.

Furthermore, if you want install a listener for the UI component's internal component, such as provide the DocumentListner for JTextField, you can use.
Code:

```
DefaultListenerRender listenerRender = new DefaultListenerRender();
listenerRender.addListener("docListener ", new yourDocumentListener ());
```

The corresponding DUIL definition:

```
<JTextField text="helo" document.documentListener ="docListener"/>
```

2.7 I/O processing

Each components that have "id" attribute can exchange value with external through IORender.developer can use the deferent value in deferent case for a component, DUIL use ResultRender to meet this case.

The DefaultResultRender will use the component's primary attribute's value as the component's result, if the component have no primary attribute or "id" is the primary attribute, then the result for this component is null.

For example, The JComboBox's primary attribute is "id", so if you use the DefaultResultRender for the JComboBox, the result for it will be null, but in some case we want use the selectedItem as the JComboBox's result, so you specify the "selectedItem" as the primary attribute by place a '@' before the attribute name in DUIL UI definition.

DUIL definition

```
<JComboBox id="my_value" @selectedItem=""/>
```

2.8 Renders

DUIL will use the deferent Renders to provide deferent information during create the JAVA GUI, all of the Renders are put in the CompositeRenders.

You must specify the component's id if you want to use the renders for the specific component.

The following code will initialize the ok_btn's text to "apply" in the above case.

```
CompositeRenders renders = new CompositeRenders();
DefaultIORender ioRender = new DefaultIORender();
renders.setIORender(ioRender);
render.save("ok_btn.text","apply");
```

All of the Renders are put in the CompositeRenders and pass to DUIL, You can use them to specific the Renders for specific component with the following method to addXXXRenders provided by CompositeRenders:

- Prototype1:

  ```
  public void setXXXRender (String dlgId, String comId, AbstractCom.XXXRender render)
  ```

  Set the specific dialog's specific component.

- Prototype2:

  ```
  public void setXXXRender (String dlgId, AbstractCom.XXXRender render)
  ```

  Set the specific dialog's renders.

- Prototype3:

  ```
  public void setXXXRender (AbstractCom.XXXRender render)
  ```

  Set the global renders.

DUIL will use the most accurate render for component, for example.
A dialog name "dialog1" contain two button, one is named with "button1", another is "button2", and the following code be used:

```
setXXXRender("dialog1","button1", XXXRender1).
setXXXRender("dialog1", XXXRender2).
setXXXRender(XXXRender2).
```

DUIL will use the XXXRender1 for dialog1.button1, and use XXXRender2 for button2.

```
public void setIORender (String dlgId, String comId, AbstractCom.IORender render)
public void setIORender (String dlgId, AbstractCom.IORender render)
public void setIORender (AbstractCom.IORender render)
```

```
public void setListenerRender(String dlgId, String comId, AbstractCom.ListenerRender render)
public void setListenerRender(String dlgId, AbstractCom.ListenerRender render)
public void setListenerRender(AbstractCom.ListenerRender render)
```

```
public void setResultRender (String dlgId, String comId, AbstractCom.ResultRender render)
public void setResultRender (String dlgId, AbstractCom.ResultRender render)
public void setResultRender (AbstractCom.ResultRender render)
```

```
public void setValueRender (String dlgId, String comId, AbstractCom.ValueRender render)
public void setValueRender (String dlgId, AbstractCom.ValueRender render)
public void setValueRender (AbstractCom.ValueRender render)
```

2.8.1 IORender

DUIL will use this Render to load/save dialog value from/to external; it will be used in the following case:

- Initialize the component.
- Update the component's value during dialog running.
- Save the dialog's result.

```
public static interface IORender extends Render
{
  /**
   * get the value from render for specific key..
   * @param key the IORender's key
   * @return
   * The value in this IORender with the specified key
   */
  public Object load(String key);

  /**
   * Get the default value render for specific key.
   * @param key the IORender's key
   * @return
   * The default value in this IORender with the specified key
   */
  public Object loadDefault(String key);
```

```
    /**
     * To determine the init mode of the value from load/loadModel.
     * @param key the IORender's key
     * @return
     * true if the specific key will append to the existed value, otherwise, false will
return
     */
    public boolean isAppend(String key);

    /**
     * Determine whether the specific attr need to be saved.
     * @param key the IORender's key
     * @param attr the attr.
     * @return
     * true means the specific attr need to be saved.
     */
    public boolean isSavedAttr(String key, String attr);

    /**
     * save the value to render.
     * @param key the IORender's key
     * @param value the value of key.
     */
    public void save(String key, Object value);

  }
```

Set the IORender in CompositeRender

```
CompositeRenders renders = new CompositeRenders();
DefaultIORender ioRender = new DefaultIORender();
ioRender.save("welcome", "hello, welcome you select DUIL solution:)");
renders.setIORender(ioRender);
```

2.8.2 ListenerRender

DUIL will use this Render to install the Listener for components.

```
  public static interface ListenerRender extends Render
  {
    /**
     *Get the listener for specific component.
     * @param key the ListenerRender's key
```

```
     * @param com the component
     * @return
     * The listener for specific listener
     */
    public EventListener getListener(String key, AbstractCom com);

    /**
     * This method is provided for developer to add listener for specific key.
     * @param key the ListenerRender's key
     * @param listener
     */
    public void addListener(String key, EventListener listener);   }
```

Set the ListenerRender in CompositeRender

```
CompositeRenders renders = new CompositeRenders();
DefaultListenerRender listenerRender = new DefaultListenerRender ();
listenerRender.save(new MyActionListener());
renders.setListenerRender(listenerRender);
```

2.8.3 Resource

You can specify the Resource for a UI component for load international string.

Set the Resource in CompositeRender

```
CompositeRenders renders = new CompositeRenders();
renders.setListenerRender("resource_base");
```

Then all of values of the I18n attribute will be load from the resource_baseXXX.properties file (XXX represent the Locale ID, such as en_US, the zh_CN).

2.8.4 ValueRender

It is used to provide the user-interface for specific component.

```
public static interface ValueRender extends Render
{
  /**
   *Get value from the specific component's user-defined interface
   * @param com DUI component
   * @param name the value of the name you want to fetch.
```

```
     * @return
     *   The value for specific name.
     */
    Object getValue(AbstractCom com, Object name);
    }
```

For example:
The DUIL definition:

```
<JPanel layoutContrains="NORTH">
    <FileButton name="openFile" open="true" text="open DUIL XML..."/>
        <JcomboBox id="=$(parent.openFile.user.opend_file)" name="dialogs"
toolTipText="dialogs list">
        <option value="-1">------N/A-----</option>
        </JComboBox>
        <JButton name="view" text="view" toolTipText="browse the select dialog"
actionListener="viewAction"/>
</JPanel>
```

The JAVA code:

```
public static class FileRender implements AbstractCom.ValueRender
 {
    public Object getValue(AbstractCom com, Object key)
    { //return the String[] value that will be used to set to the JComboBox
    }
 }
```

```
 CompositeRenders renders = new CompositeRenders();
 renders.setValueRender(new FileRender());
```

The JcomboBox's sub item will be set to the openFile (a fileButton)'s user interface's value.

2.9 Components provided by DUIL

2.9.1 RadioGroup

RadioGroup is a JPanel Container that supports ButtonGroup; it is used to contain the JRadioButton.
Such as

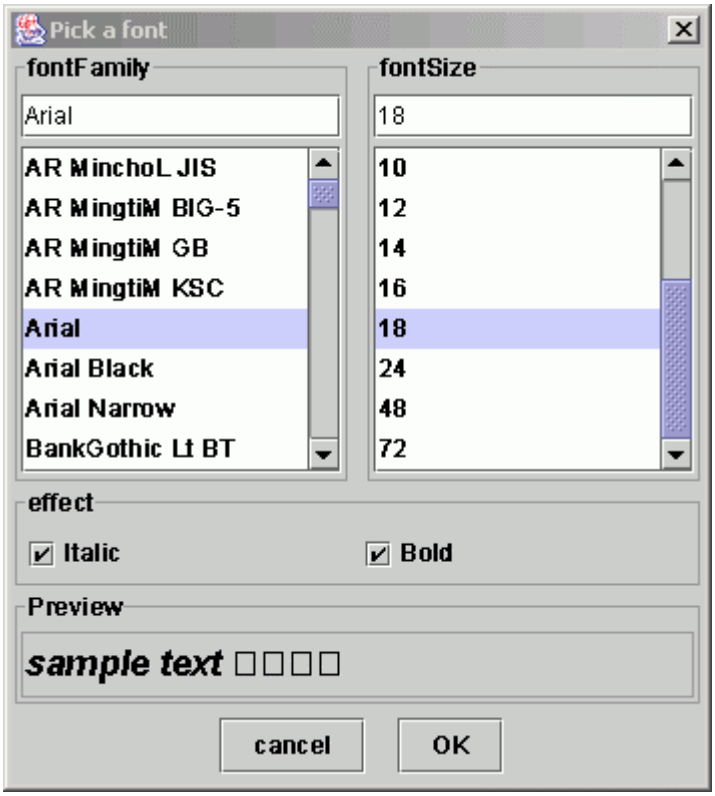
```
    <RadioGroup id="choice" name="choice" border="true" text="would you have a
```

```
coffee">
        <JRadioButton name="yes" text="Yes" toolTipText="yes, I like coffee"
selected="true" />
        <JRadioButton name="no" text="No" toolTipText="no, I like tea" />
    </RadioGroup>
```

Primary attribute
Please refer to the demoDUIVar_common

2.9.2 FontChooser

This component is used to enable user can select a font.



2.9.3 FileButton

This is a component that used to choose Files, it will popup a FileChooserDialog when click it.

2.9.4 ColorButton

This component is used to enable user can select a font, a ColorChooserDialog will popup when click it.

2.10 Dialog's behavior

ComFactory provide a method ShowDialog to enable you show your UI unit that defined in the XML definition in a dialog very easy.

Generally, a Dialog is used to show some thing to user and fetch the result that user provide. DUIL provide an easy solution to meet this case.

You can use the **action** attribute for the button components that contained in your UI unit.
The action can be the following value, each value can be combine with ';'

- APPLY
  This value is used to save the dialog's value to IORender.
- HIDE_WINDOW
  Hide the dialog
- CLOSE_WINDOW
  Dispose the dialog.

"APPLY;CLOSE_WINDOW" means that you want to close the dialog with saving the value to IORender.
"CLOSE" means that you want to close the dialog without saving the value to IORender.

2.11 Use DUIL in your program

You can use DUIL in the following step:

1. Create the ComFactory
2. Provide your Renders to ComFactory
3. Get the Render of your UI XML definition
4. Fetch the result of your UI if you need.

The UI unit definition:

```
<DIALOG id="myUI" layout="=new GridLayout(4,2)" text="demoUI: pease input your information"  preferredSize="270,140">
    <JLabel text="userName:"/>
    <JTextField name="username" text="put your name here"/>
    <JLabel text="password:"/>
    <JTextField name="password" text="put your password here"/>
    <JButton name="cancel" text="cancel" action="CLOSE_WINDOW"/>
    <JButton name="ok" text="ok" action="APPLY;CLOSE_WINDOW"/>
```

```
</DIALOG>
```

```
//1.create the Comfactory
    ComFactory factroy = ComFactory.createInstance();
//2. Provide the Renders.
    CompositeRenders renders = new CompositeRenders();
    AbstractCom.IORender ioRender = new DefaultIORender();
    DefaultListenerRender listenerRender = new DefaultListenerRender();
    listenerRender.addListener("viewAction", new ViewActionHandler());
    renders.setValueRender(new AbstractCom.DefaultValueRender());
    ioRender.save("username","initial text");
    renders.setIORender(ioRender);
    renders.setListenerRender(listenerRender);
    renders.setValueRender(new FileRender());
//3. Get the render for your UI XML definition.
    Reader reader =new InputStreamReader(new FileReader("myUI.xml"));
//4. Show it
//you can use the showDialog to show your UI definition in a dialog directly.
    Object obj=factroy.showDialog(reader, "id", "myUI", renders, null, true);
//you can use the createComponent to generate the GUI for your UI XML definition.
    AbstractCom ac = factory.createComponent(file, "id", " myUI", renders);
    Component myCom=ac.getComponent();
//you can use the myCom to your program.
//5. get the dialog result after user selection.
    if (obj != null)
    {
        String retval = ( (AbstractCom) obj).getComponent().getName();
        if ("ok".equals(retval))
        {
            //fetch the result
            String result_user=ioRender. load("username");
            String result_pass=ioRender. load("password");
        }
    }
```

2.12 Integrated user-defined component

DUIL provide an easy mechanism to enable you use the user-defined component in DUIL definition XML.

There are two methods to integrate your component.

1. Use the any wrapper

Any is a wrapper that can contain all of the component, but in this way, you component will have not the default primary attribute (but you can specify the primary attribute by use '@').

```
<any object="new myComponent()"/>
```

But you cannot specific DUIL tag for DUIL component.
2. Write a wrapper for your component.
You can specify the tag for your component if you write a wrapper for it; the wrapper will extends from org.lx.gui.dui.AbstractCom.
You overwrite the following method to provide some.

Please refer to the API for detail.

2.13 Customer tag

You can customize the component tag as you like in your UI definition file.
Such as you don't want to use the JTextField as the tag, you can add one line in $DUIL_HOME/duil_cfg.xml

```
<element name="MYEDITOR" class="org.lx.gui.dui.com.ComJTextField"/>
```

Then in your UI definition file, you can define your UI with MYEDITOR as following.

```
<MYEDITOR text="hello, world"/>
```

3 Config DUIL

DUIL provide a XML file to init the work parameters, you can configure the following options:

1. Add user-defined component's wrapper
2. Specify the log output level.
3. Porive the codec for DUIL, the codec (coder and decoder) is used to convert the value between java data type and string type, such as string-color coder can convert #ff0000 to the java.awt.Color type.

The log's level attribute is used to control the log information; you can set it to 1(DEBUG) when you are designing your UI.

The level can be the following value:
1:DEBUG

3:MESSAGE
5:NOTICE
7:WARNING
9:ERROR

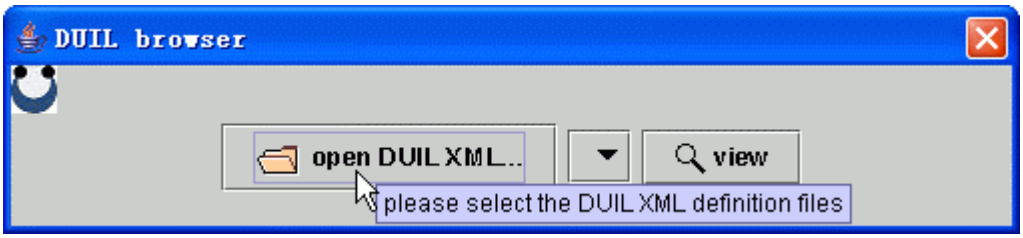
The default level is ERROR, that will improve the performance of DUIL, but we suggest that you set the level to DEBUG when you design your UI, because in DEBUG level, DUIL will output more information about your UI that will to help you resolve many problem.

A simple DUIL config file:

```
<DUIL>
 <elements ver="1.0">
     <element name="YourButtonTag" class="your component wrapper class"/>

 </elements>

 <log level="1"/>
</DUIL>
```

4 duilBrowser

DUIL provide a browser for you to view your UI definition.

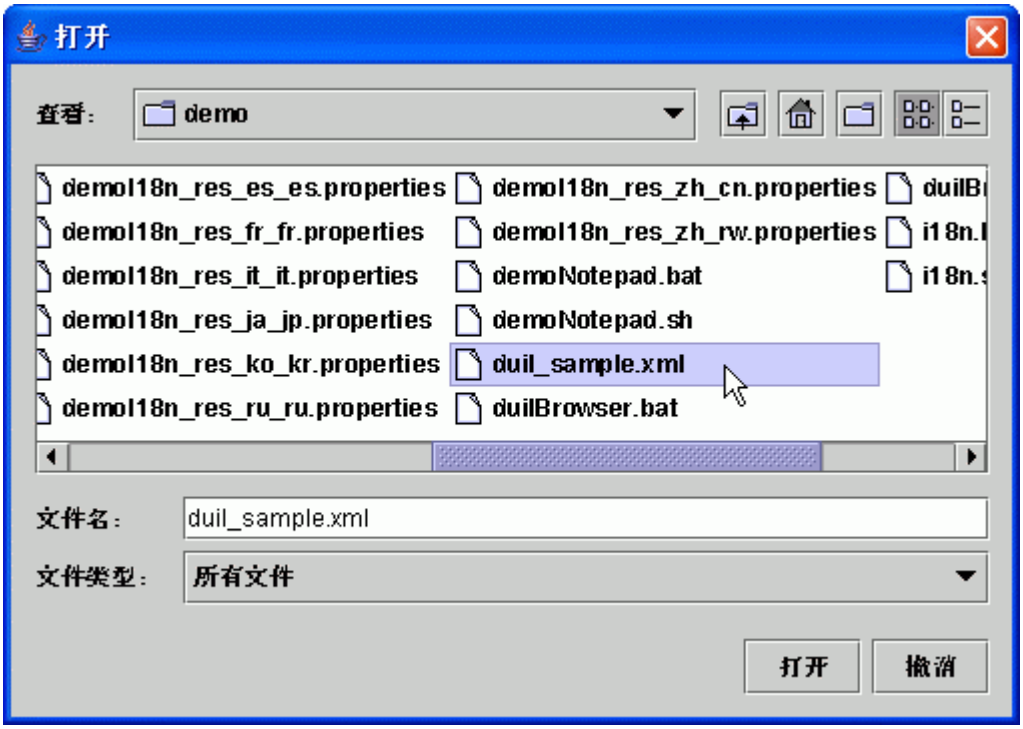


graph 4 Browser UI

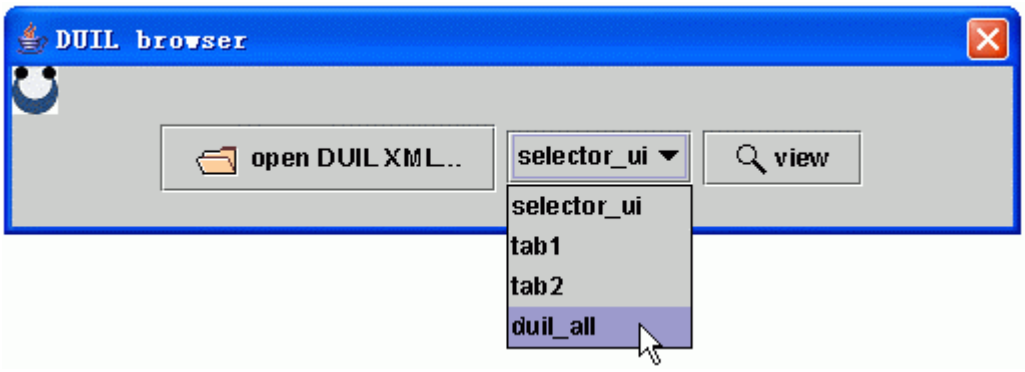
Usage；

1. Click the “open DUIL XML...” button to select your XML UI definition.

There is a duil_sample.xml file in DUIL package’s demo directory, this XML contains most java component that define with DUIL.

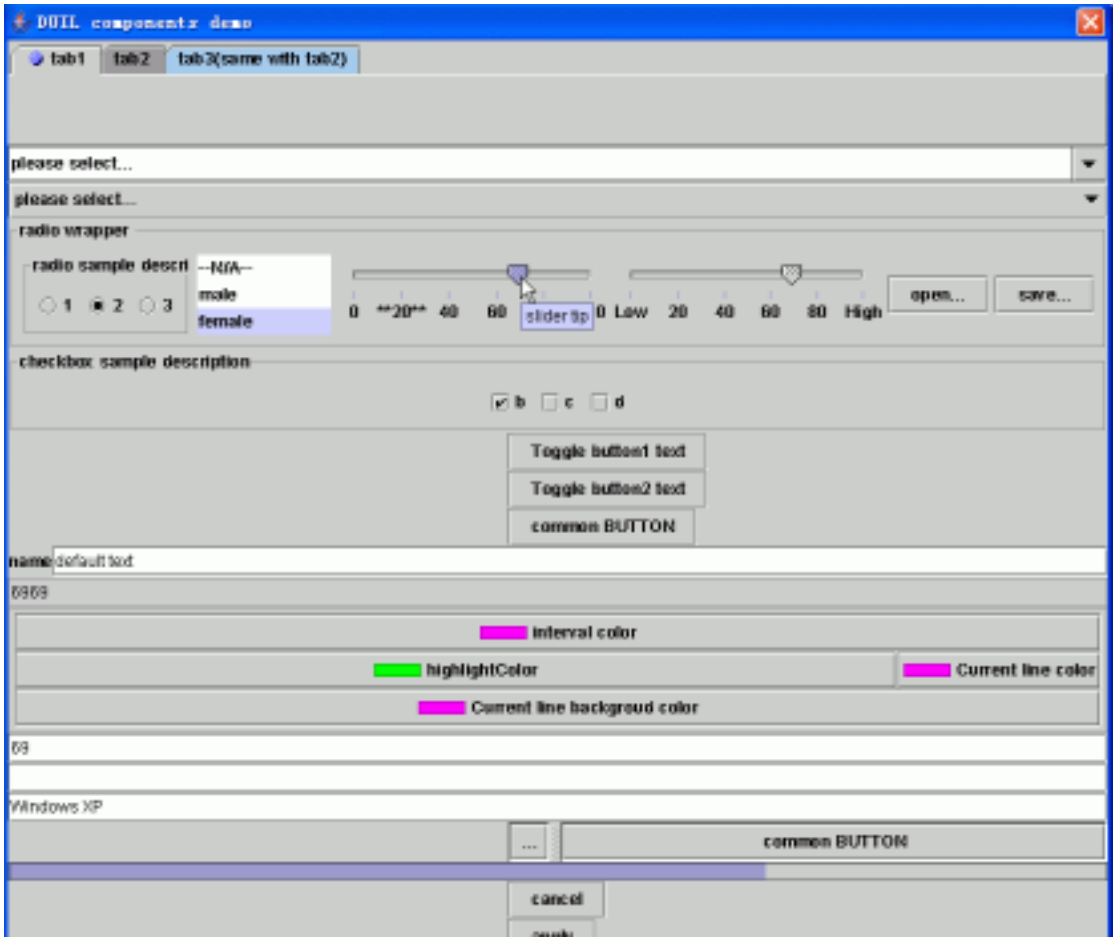


2. After you select the dialog, all of the top level UI unit will be list in the combo list.



You can see four UI units in the XML UI definition file in this case.

3. After select the UI unit, press the “view” button, then the UI will occur, It is so cool, yeah!



Tips: you can drag the slider, and then the content of editor will change automatically.

duilBrowser's source code and UI definition XML are contained in DUIL package, it will show the following feature for you.

- FileButton
- DUIL script
- DUIL user-definition variable

This browser is used to browser the DUIL XML definition.

DUIL XML:

```
    <JPanel id="duilBrowser" layout="=new BorderLayout()" text="DUIL browser"
preferredSize="500,60">
        <JPanel layoutContrains="NORTH">
            <FileButton name="openFile" open="true" text="open DUIL XML..."/>
            <JComboBox                  id="=$(parent.openFile.user.opend_file)"
name="dialogs" toolTipText="dialogs list">
            <option value="-1">------N/A-----</option>
            </JComboBox>
            <JButton name="view" text="view" toolTipText="browse the select
```

```
dialog" actionListener="viewAction"/>
        </JPanel>
    </JPanel>
```

JAVA code:

The following is the code skeleton:

```
public class browser
{
  public static class ViewActionHandler extends AbstractDUIListener implements
      ActionListener
  {
    public void actionPerformed(ActionEvent e)
    {
        Object selected = dlg_chooser.getComAttribute("selectedItem");
        if (selected instanceof ItemWrapper)
        {
         XMLElementModel model = (XMLElementModel) ( (ItemWrapper) selected).
                    getContext();
       //show the specific component
       }
    }
  }

//support the “user” DUIL variable.
  public static class FileRender implements AbstractCom.ValueRender
  {
    public Object getValue(AbstractCom com, Object key)
    {
    //when the key is “opend_file”, return the DUI Model in this file
    }
  }

  public static PipedWriter m_writer;
  public static PipedReader m_reader;

  public static void main(String[] argv)
  {
    Log.init(true, Log.DEBUG);
//…create the ComFactroy
    ComFactroy factroy = ComFactroy.getSingleInstance();
    CompositeRenders renders = new CompositeRenders();
    AbstractCom.IORender ioRender = new DefaultIORender();
    DefaultListenerRender listenerRender = new DefaultListenerRender();
    listenerRender.addListener("viewAction", new ViewActionHandler());
    renders.setValueRender(new AbstractCom.DefaultValueRender());
```

```
    renders.setIORender(ioRender);
    renders.setListenerRender(listenerRender);
    renders.setValueRender(new FileRender());
    in =new InputStreamReader(
        browser.class.getResourceAsStream(
        "/test/DUIL_browser.xml"));
    model = parser.parser(in);
    factroy.showDialog(model.getChild("id", "DUIL_Browser"), renders, null, true);
    System.exit(0);
  }
```

5 Sample

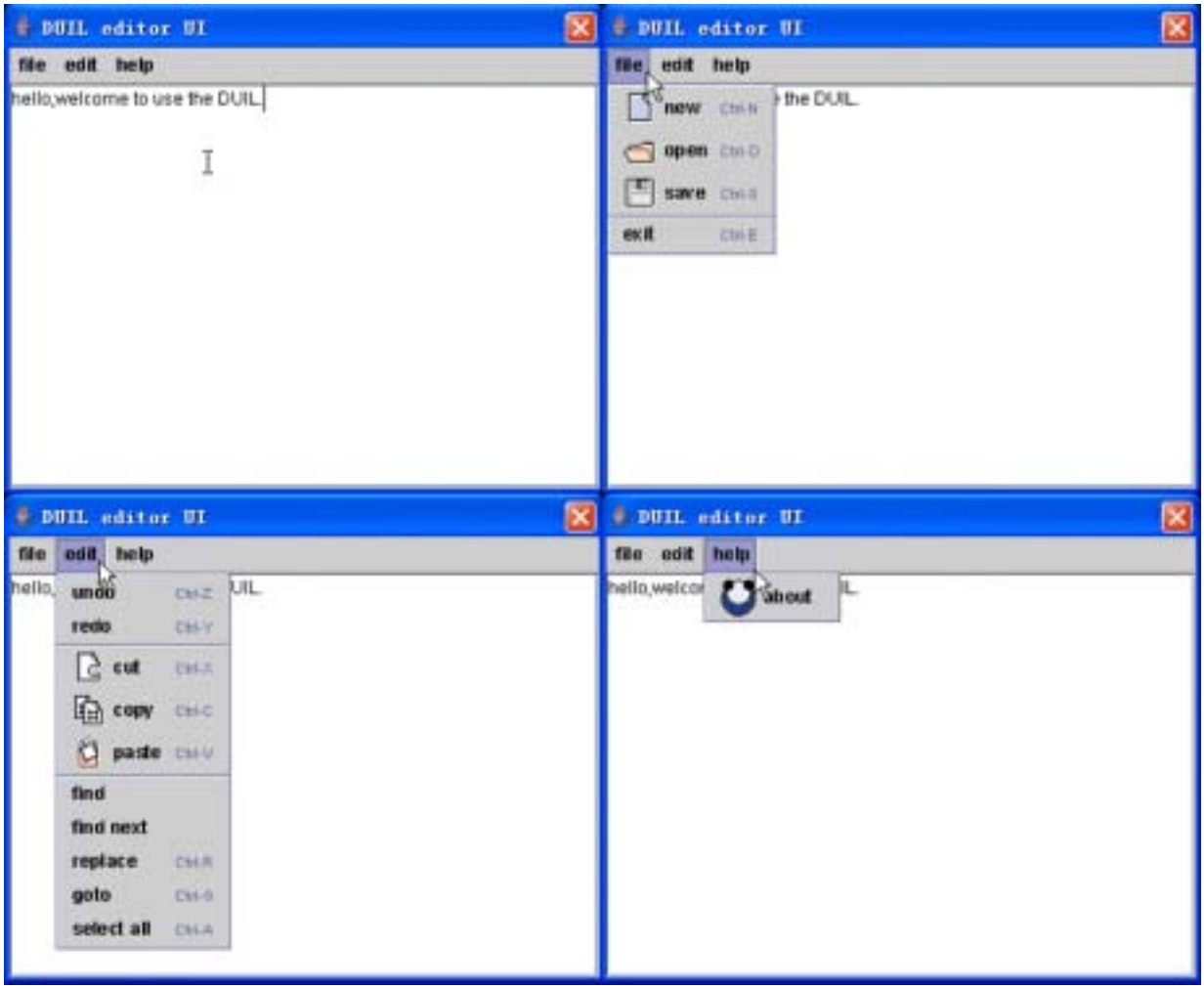
5.1 demoNotePad

- It is a sample that show how to use DUIL to generate a simple Notepad's UI. please refer the demo source for detail.
  UI XML definition:

```
<JPanel text="DUIL editor UI" layout="=new BorderLayout()" id="demoNotepad"
preferredSize="400,300">
   <JMenuBar name="menubar" layoutContrains="NORTH">
      <JMenu text="file" name="file">
          <JMenuItem text="new" name="new" icon="images/new.gif"
accelerator="=KeyStroke.getKeyStroke (\"ctrl N\")"/>
          <JMenuItem text="open" name="open" icon="images/open.gif"
accelerator="=KeyStroke.getKeyStroke(\"ctrl O\")"/>
          <JMenuItem text="save" name="save" icon="images/save.gif"
accelerator="=KeyStroke.getKeyStroke(\"ctrl S\")"/>
        <JSeparator/>
                  <JMenuItem text="exit" name="exit"
accelerator="=KeyStroke.getKeyStroke(\"ctrl E\")"/>

      </JMenu>
      <JMenu text="edit" name="edit">
                <JMenuItem text="undo" name="undo"
accelerator="=KeyStroke.getKeyStroke(\"ctrl Z\")"/>
                  <JMenuItem text="redo" name="redo"
accelerator="=KeyStroke.getKeyStroke(\"ctrl Y\")"/>
        <JSeparator/>
          <JMenuItem text="cut" name="cut" icon="images/cut.gif"
accelerator="=KeyStroke.getKeyStroke(\"ctrl X\")"/>
          <JMenuItem text="copy" name="copy" icon="images/copy.gif"
accelerator="=KeyStroke.getKeyStroke(\"ctrl C\")"/>
```

```
          <JMenuItem text="paste" name="paste" icon="images/paste.gif"
accelerator="=KeyStroke.getKeyStroke(\"ctrl V\")"/>
        <JSeparator/>
        <JMenuItem text="find" name="find" />
        <JMenuItem text="find next" name="find_next" />
                              <JMenuItem                          text="replace"
name="replace"accelerator="=KeyStroke.getKeyStroke(\"ctrl R\")"/>
                    <JMenuItem            text="goto"            name="goto"
accelerator="=KeyStroke.getKeyStroke(\"ctrl G\")"/>
               <JMenuItem       text="select       all"       name="select_all"
accelerator="=KeyStroke.getKeyStroke(\"ctrl A\")"/>
      </JMenu>
      <JMenu text="help" name="help">
         <JMenuItem text="about" name="about" icon="images/duil_small.gif"/>
      </JMenu>

   </JMenuBar>
   <JScrollPane layoutContrains="CENTER">
      <JTextArea text="hello,welcome to use the DUIL."/>
   </JScrollPane>


</JPanel>
```

The following is the result of the above definition.



graph 5 demoNotePad

5.2 demoCommon

This sample will show the usage of:

- RadioGroup
- DUIL script
- DUIL variable (this/parent/root)

```
<JPanel layout="=new BorderLayout()" id="demoCommon">
 <JPanel layout="=new BoxLayout($(this),BoxLayout.Y_AXIS )">
     <JTextField id="welcome" text="" enabled="" background="#000000"/>
     <RadioGroup id="choice" name="choice" border="true" text="would you have a coffee">
         <JRadioButton name="yes" text="Yes" toolTipText="yes, I like coffee" selected="true" />
         <JRadioButton name="no" text="No" toolTipText="no, I like tea" />
```

```
      </RadioGroup>
      <JLabel
text="=$(parent.yes).isSelected()?$(parent.yes).getToolTipText():$(parent.no).getTool
TipText()"/>
      <JCheckBox name="chk" text="enable image button" selected="true"/>
      <JButton text="imageButton"   enabled="=$(parent.chk).isSelected()"
          verticalTextPosition="BOTTOM" horizontalTextPosition="CENTER"
          disabledIcon="images/b1d.gif"                    icon="images/b1.gif"
pressedIcon="images/b1p.gif" rolloverIcon="images/b1r.gif"/>
  </JPanel>
  <JPanel layoutContrains="SOUTH" >
      <JButton text="close" action="CLOSE_WINDOW"/>
  </JPanel>
</JPanel>
```

In this dialog, if you chose the yes, then the label's text attribute will update automatically, yeah, DUIL is so easy, I needn't write any java code to implement this function.

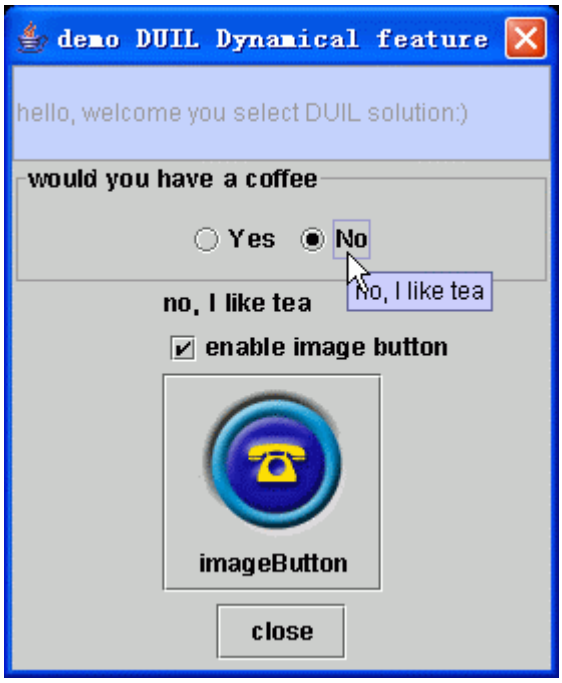
You can use **"root"** here also, but the performance is low than **"parent"**, because the search range of **"range"** is bigger than the range of **"parent"**,

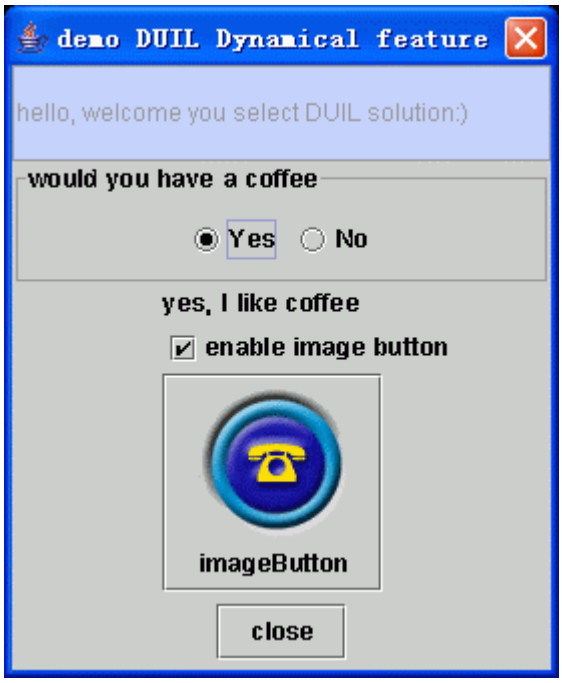
The following is the UI for this sample.

You can also change the XML UI component's attributes with java code.

```
//1. provide the initial value for component
  DefaultIORender ioRender = new DefaultIORender();
  ioRender.save("welcome", "hello, welcome you select DUIL solution:)");
  ioRender.save("welcome.enabled", Boolean.FALSE);
  ioRender.save("welcome.background", "#C2D3FC");
  renders.setIORender(ioRender);
  factory.showDialog(reader, "id", "demoCommon", renders, null, true);
```

After you initialize with the above code, the UI are list as following:





graph 6 demoCommon UI

5.3 demoI18n

This sample will show the usage of:

- DUIL script
- DUIL variable (env)
- I18n

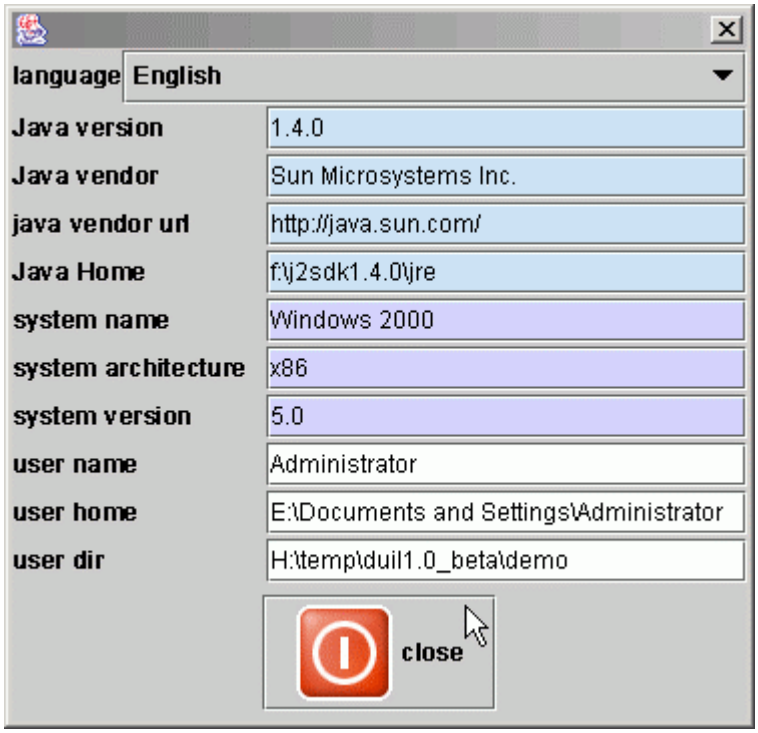
```
    <JPanel layout="=new BorderLayout()" id="demoI18n" size="300,200"
language="=$(lang).getSelectedItem().getContext().getAttribute(\"value\")">
        <JPanellayout="=new BoxLayout($(this),BoxLayout.X_AXIS )"
layoutContrains="NORTH">
            <JLabel id="cur_lang" $text="lang:"/>
            <JComboBox name="lang">
                <option value="en_US">English</option>
                <option value="zh_CN">Chinese (Simplified)</option>
                <option value="zh_TW">Chinese (Traditional)</option>
                <option value="de_DE">German</option>
                <option value="es_ES">Spanish</option>
                <option value="fr_FR">French</option>
                <option value="it_IT">Italian</option>
                <option value="ja_JP">Japanese</option>
                <option value="ko_KR">Korean</option>
                <option value="ru_RU">Russian</option>
            </JComboBox>
        </JPanel>
    <JPanel layout="=new GridBagLayout()" >
            <JLabel id="jv_ver" $text="java version" layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.RELATIVE ,1

    ,0,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL ,new
Insets(1,0,1,0),10,0)"/>
            <JTextField text="=$(env.java.version)" background="#CDE0F5"
toolTipText="Java Runtime Environment version" layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.REMAINDER ,1

    ,1,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL,new
Insets(1,0,1,0),10,0)"/>
            <JLabel id="jv_vendor" $text="java vendor" layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
```

```
                                ,GridBagConstraints.RELATIVE ,1

    ,0,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL          ,new
Insets(1,0,1,0),10,0)"/>
            <JTextField    text="=$(env.java.vendor)"    background="#CDE0F5"
toolTipText="Java  Runtime  Environment  vendor"    layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.REMAINDER ,1

    ,1,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL,new
Insets(1,0,1,0),10,0)"/>
            <JLabel      id="jv_vendor_url"      $text="java      vendor      url"
layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.RELATIVE ,1

    ,0,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL          ,new
Insets(1,0,1,0),10,0)"/>
            <JTextField  text="=$(env.java.vendor.url)"  background="#CDE0F5"
toolTipText="Java        vendor        URL"        layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.REMAINDER ,1

    ,1,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL,new
Insets(1,0,1,0),10,0)"/>
            <JLabel  id="jv_home"  $text="java  home"  layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.RELATIVE ,1

    ,0,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL          ,new
Insets(1,0,1,0),10,0)"/>
            <JTextField    text="=$(env.java.home)"    background="#CDE0F5"
toolTipText="Java      installation      directory"      layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.REMAINDER ,1

    ,1,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL,new
Insets(1,0,1,0),10,0)"/>

            <JLabel id="sys_name" $text="system name" layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.RELATIVE ,1

    ,0,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL          ,new
```

```
Insets(1,0,1,0),10,0)"/>
            <JTextField      text="=$(env.os.name)"      background="#D0D0FF"
toolTipText="Operating      system      name"      layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.REMAINDER ,1

    ,1,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL,new
Insets(1,0,1,0),10,0)"/>
            <JLabel      id="sys_arc"      $text="system      architecture"
layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.RELATIVE ,1

    ,0,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL      ,new
Insets(1,0,1,0),10,0)"/>
            <JTextField      text="=$(env.os.arch)"      background="#D0D0FF"
toolTipText="Operating    system    architecture"      layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.REMAINDER ,1

    ,1,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL,new
Insets(1,0,1,0),10,0)"/>
            <JLabel id="sys_ver" $text="system version" layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.RELATIVE ,1

    ,0,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL      ,new
Insets(1,0,1,0),10,0)"/>
            <JTextField      text="=$(env.os.version)"      background="#D0D0FF"
toolTipText="Operating      system      version"      layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.REMAINDER ,1

    ,1,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL,new
Insets(1,0,1,0),10,0)"/>
            <JLabel id="usr_name" $text="user name" layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.RELATIVE ,1

    ,0,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL      ,new
Insets(1,0,1,0),10,0)"/>
            <JTextField  text="=$(env.user.name)"  toolTipText="User's  account
name"                                                  layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
```

```
                                ,GridBagConstraints.REMAINDER ,1

    ,1,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL,new
Insets(1,0,1,0),10,0)"/>
            <JLabel id="usr_home" $text="user home" layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.RELATIVE ,1

    ,0,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL          ,new
Insets(1,0,1,0),10,0)"/>
            <JTextField text="=$(env.user.home)" toolTipText="User's home
directory"                                                  layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.REMAINDER ,1

    ,1,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL,new
Insets(1,0,1,0),10,0)"/>
            <JLabel id="usr_dir" $text="user dir" layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.RELATIVE ,1

    ,0,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL          ,new
Insets(1,0,1,0),10,0)"/>
            <JTextField text="=$(env.user.dir)" toolTipText="User's current
working              directory"                             layoutContrains="=new
GridBagConstraints(GridBagConstraints.RELATIVE ,GridBagConstraints.RELATIVE
                                ,GridBagConstraints.REMAINDER ,1

    ,1,0,GridBagConstraints.CENTER,GridBagConstraints.HORIZONTAL,new
Insets(1,0,1,0),10,0)"/>
    </JPanel>
    <JPanel layoutContrains="SOUTH" >
        <JButton                text="close"                icon="images/exit.gif"
pressedIcon="images/exitp.gif"                      rolloverIcon="images/exitr.gif"
action="CLOSE_WINDOW"/>
    </JPanel>
    </JPanel>
```

In this dialog, you can change the language of the UI interface, the DUIL will the show the dialog with the specifc language's string.



graph 7 DemoI18n UI

5.4 demoBridge

This sample will show the usage of:

- DUIL script
- DUIL variable (bridgeRoot)

This demo shows how to build relationship between two components that locate in deferent dialog.
When you drag the slider in dialog1, the spinner in dialog2's value will change with corresponding value, and also, if you change the spinner's value, the value of slider will change also.

DUIL definition XML:

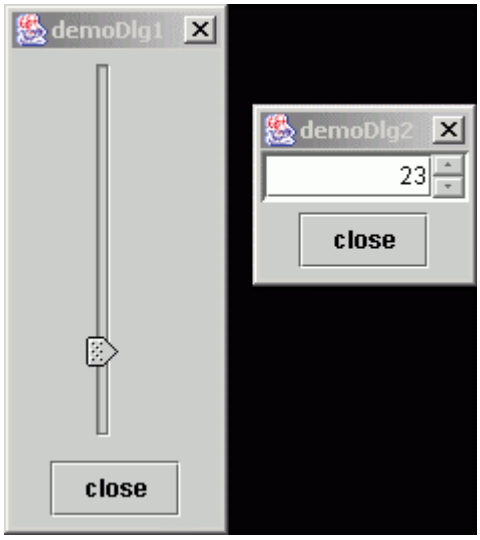
```
<JPanel layout="=new BorderLayout()" id="demoDlg1">
    <JSlider                                                    name="slider"
value="=$(bridgeRoot.spinner)==null?0:$(bridgeRoot.spinner).getValue()"
paintTrack="true" paintTicks="true" extent="0" minimum="0" maximum="100"
paintTrack="true" toolTipText="slider tip" />
    <JPanel layoutContrains="SOUTH" >
        <JButton text="close" actionListener="closeAction"/>
    </JPanel>
</JPanel >
```

```
<JPanel layout="=new BorderLayout()" id="demoDlg2">
    <JSpinner                                                    name="spinner"
value="=$(bridgeRoot.slider)==null?0:$(bridgeRoot.slider).getValue()"/>
    <JPanel layoutContrains="SOUTH" >
        <JButton text="close" actionListener="closeAction"/>
    </JPanel>
</JPanel >
```

JAVA code:

```
    ComFactory factory = ComFactory.createInstance(null);
    String file = "/org/lx/demo/demoBridge.xml";
    CompositeRenders renders = new CompositeRenders();
    JDialog dialog = new JDialog();
    renders.setListenerRender(listenerRender);
    //1. create the dialog1
    AbstractCom ac = factory.createComponent(file, "id",
                                            "demoDlg1", renders);
    ComBridge bridge = new ComBridge();
    bridge.add(ac);
    dialog.setContentPane( (Container) ac.getComponent());
    dialog.setTitle("demoDlg1");
    dialog.pack();
    dialog.setDefaultCloseOperation(dialog.EXIT_ON_CLOSE);
    dialog.show();
    dialog.setLocationRelativeTo(null);

    //2. create the dialog2
    ac = factory.createComponent(file, "id", "demoDlg2", renders);
    bridge.add(ac);
    dialog = new JDialog();
    dialog.setTitle("demoDlg2");
    dialog.setContentPane( (Container) ac.getComponent());
    dialog.pack();
    dialog.setDefaultCloseOperation(dialog.EXIT_ON_CLOSE);

    dialog.show();
    dialog.setLocationRelativeTo(null);
```

The GUI is show as following:



graph 8 demoBridge UI