Interface Modeling Language (IML)

User's Guide

Product Version 4.1.11

Document Version : 1.1
Date : 2006 January 20
Author : Virtual Weaver Interactive

IML (Interface Modeling Language) is an XML dialect used to build Java Swing user interfaces and beyond, any Java application based on a graphical interface.
IML is composed with two parts, respectively named Control and Display :

- Display part contains all graphical structures of the user interface ;
- Control section contains interface control directives ; under this element are defined bindings between event sources and targets, java classes used to receive and deal events generated by interface elements, button groups, etc.

IML is implemented with Xotics API so, any IML document, when loaded, becomes a real java program with a graphical interface.
IML can accept elements from any other dialect, provided some implementation rules are respected.

Starting

Here is an example showing how is structured an IML document :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<iml xmlns="http://www.virtualweaver.com/IML/iml400.dtd" >
  <resource url="http://www.xotics.org/iml/iml_ex1.jar" />
  <control >
    <controller className="com.xotics.Control1" name="control1" />
    <event-binding eventClass="java.awt.event.ActionEvent"
                   listenerRef="control1"
                   sourceRef="button1" />
  </control>
  <display >
    <frame bounds="0,0,300,200" name="frame1" title="" >
      <box axis="VERTICAL" >
        <cell flexible="true" >
          <flowbox >
            <cell >
              <label horizontalAlignment="CENTER"
                     horizontalTextPosition="CENTER"
                     icon="file:/d:/tmp/cover1.jpg"
                     text="cover 1"
                     verticalAlignment="BOTTOM"
                     verticalTextPosition="BOTTOM" />
            </cell>
            <cell >
              <label horizontalAlignment="CENTER"
                     horizontalTextPosition="CENTER"
                     icon="file:/d:/tmp/cover2.jpg"
                     text="cover 2"
                     verticalAlignment="BOTTOM"
                     verticalTextPosition="BOTTOM" />
            </cell>
          </flowbox>
        </cell>
        <cell >
          <flowbox >
            <cell >
              <button actionCommand="Button1" name="button1" text="Button1" />
            </cell>
          </flowbox>
        </cell>
      </box>
    </frame>
  </display>
</iml>
```

Let's consider the direct children of <iml> root element.
<resource> is used to add resources, as JAR files or directories, to this document, in order to load, for instance, some specific class or icon. An IML document can contain any count of this element.

The Display part, under <display> element, defines a frame looking like this :

Content of this frame is described in the Display chapter.

In the Control part of this document, under <control> element, it is specified to send every Action event from the button to a Java class named control1, which is able to perform some operation in response to the button click.

Root elements

<iml>

It is the root element.

Attributes :
style : URL of a style document ; when this value is not null, style document is played just after IML document loading
Content Model :
A sequence of : any count of <resource>, then an optional <control> followed by a mandatory <display> element.

<resource>

Adds a resource URL to the classpaths specific to the document. Can be useful, for example, to specify the path of specific icons used in the document.

Attributes :
url : the URL to a directory or a JAR file.

Control section

<control>

Control part of an IML document is the subtree rooted by <control> element. It can accept any count of <controller>, <event-binding> and <button-group> elements as children.
More generally, <control> can accept any child element whose Java class implements the following interface :

```
package com.virtualweaver.xotics.dialect.iml.datatype;

public interface ImlControlable {
  public void initControl();
}
```

The method initControl() is called just after the document is loaded, to perform some initialization.

Content Model :
Any count of elements implementing ImlControlable. In IML dialect there is : <controller>, <event-binding> and <button-group>

<controller>

<controller> is the element which defines a controller instance.
A controller is a class used to perform some operation, at its initialization and/or in response to some event. It is generally used to handle an event sent by a graphical element. A controller class must implement the interface :

```
package com.virtualweaver.xotics.dialect.iml.datatype;

public interface ImlEventController {
  public void setDMInstance(XoDMInstance dmi);
  public void setParam(String name, String value);
  public void init();
}
```

First method gives access to the whole IML document, and the second provides an init param, described with child element <param>. Any count of <param> elements can be specified. Init() method is the mast method called.

Attributes :
name : the unique ID of the controller, permits to identify the source or target of an event sending className : the full name of the class to instanciate, which must implement ImlEventController
Content Model :
Any list of <param> elements.

<param>

Each controller's param is provided by a <param> element, associating a name (@name) with a value (as PCDATA).

Attributes :
name : the name of the param PCDATA : value of the param

Below is an example using <controller> and <param> :

```
  <control >
    <controller className="com.xotics.Control1" name="control1" >
      <param name="directory">d:/tmp/</param>
      <param name="kind">B</param>
    </controller>
  </control>
```

In this example, the class `com.xotics.Control1` must implement `ImlEventController`. This class is instanciated when <controller> method `initControl()` is called, and initialized with params (directory=d:/tmp/) and (kind=B). Control1 instance is named in this document "control1".

<event-binding>

Event binding is an operation consisting in linking an event source to an event listener. The element <event-binding> defines such association. First, @eventClass specifies the class name of the event transmitted. @sourceRef and @listenerRef reference the ID in the document of the source of event and corresponding listener.

Attributes :
eventClass : full class name of the event transmitted listenerRef : name of a valid event listener, declared somewhere in the document sourceRef : name of a valid event source, declared somewhere in the document

```
  <control >
    <controller className="com.xotics.Control1" name="control1" />
    <event-binding eventClass="java.awt.event.ActionEvent"
                   listenerRef="control1"
                   sourceRef="button1" />
  </control>
```

In this example, <event-binding> specifies that control1 must listen to Action events originated by button1. To be valid, Control1 must also implement ActionListener, and button1 must have these methods :

```
public void addActionListener(ActionListener l);
public void removeActionListener(ActionListener l);
```


More generally, with @eventClass equals to :

```
<package>.<event name>Event
```

where <package> is the package of the event class, and <event name> is the name of an event, the rules below must be satisfied :

- a class whose instance ID is specified in @listenerRef must implement `<event name>Listener` interface ;
- a class whose instance ID is specified in @sourceRef must have both methods `add<event name>Listener()` and `remove<event name>Listener()`.

Attributes :
eventClass : full class name of the event transmitted listenerRef : ID of the listener instance sourceRef : ID of the event source instance name : ID of this <event-binding> element

<button-group>

The element <button-group> in control section defines a group of buttons as with javax.swing.ButtonGroup. This element's main value is its @name which identifies a specific group.
When initControl() method of <button-group> implementation is invoked, each graphical element in the document :

- whose implementation is derived from javax.swing.AbstractButton ;
- having an @group ;
- @group set to the same value than @name of <button-group> ;

is added to the group.

Attributes :
name : ID of this element

Display section

The element <display> defines the display section by holding every graphical structure described in XML.

Note about graphical attributes

Graphical elements described in this chapter share some attributes. So, for readability purpose, these attributes are grouped as follows :

Swing Common Attributes are :

@alignmentX, @alignmentY, @background, @border, @font, @foreground, @maximumSize, @minimumSize, @name, @opaque, @preferredSize, @styleId, @toolTipText.

Notice that @name is the common attribute to define an ID. Thus, as for any ID, each value must be unique in the document.

Button Base Attributes are :

@actionCommand, @disabledIcon, @enabled, @horizontalAlignment, @horizontalTextPosition, @icon, @iconTextGap, @margin, @pressedIcon, @rolloverIcon, @text, @verticalAlignment, @verticalTextPosition, @visible.

Most of these attributes are direct wrapping of Swing component properties and thus, do not need special information (please refer to Java Swing documentation), excepted for all icon attributes and @styleId.

@styleId

references a Style, a set of attribute values which can be applied at any moment. A Style is identified by an ID, so any graphical element having @styleId set to this ID belongs to this Style and thus will have its attributes set accordingly.

Iconic attributes

(@icon, @rolloverIcon, @pressedIcon, etc.) are set with an URL locating an image file, whose protocol can be any valid Java protocol (file:, http:, ftp:, ..) or "cp:". Protocol cp means Class Path, and thus defines a classpath to an image file. Here is an example :

```
<label icon="http://www.xotics.org/myimage.jpg" />
<label icon="cp:com/virtualweaver/xotics/myimage.jpg" />
```

Attribute saving

These attributes are never written as XML text, mostly to not interfere with any Look&Feel :

Look&Feel interference	@background, @foreground, @font, @opaque, @border, @margin, @pressedIcon, @rolloverIcon, @disabledIcon
Technical reason	@bounds, @visible

If you need to set these attributes permanently, you should use an IML Style document to store these settings, as explained in next section.

Styling

For graphical elements, setting values for some attributes can interfere with current Look&Feel :

@background, @foreground, @font, @opaque, @border, @margin, @pressedIcon, @rolloverIcon, @disabledIcon.

Thus, these attributes can never be saved in XML documents. However, you may want to store some attribute settings involving these particular attributes, such as background color, special border, etc.
You can do it using a separate document called IML Style document, which contains attribute settings to set when an IML document is loaded. Here is an example of style document :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<style xmlns="http://www.virtualweaver.com/IML/iml400.dtd" >
  <context-element select="fn:id('help.box.top')" >
    <attribute name="background" >0,0,0</attribute>
  </context-element>
  <context-element select="fn:id('help.title.label')" >
    <attribute name="foreground" >255,255,255</attribute>
    <attribute name="font" >Arial,Bold,14</attribute>
  </context-element>
</style>
```

This document selects particular elements from a target document, to set particular attributes with specific values.
To perform operations described in such style document, you can set style document URL in @style of <iml> root's element of the target document. Style document is then executed just after target document loading.

<context-element>

Selects elements from a target document, by executing an XPath request. Each element from the result has its attributes set according to <context-element> children.

Attributes :
select : an XPath request to select elements
Content Model :
A list of <attribute> children

<attribute>

Sets a text value to an attribute.

Attributes :
name : XML attribute name PCDATA : text value of this attribute

Data management

Some IML elements are designed to handle data, like <textfield>, <list>, <table>. This data is displayed and sometimes set by the user interface, can be copied to the Clipboard and moved by Drag & Drop.
IML provides a common way to get and set data for such components. To take benfit of it, these components must implement this interface :

```
package com.virtualweaver.xotics.dialect.iml.datatype;

import java.awt.datatransfer.Transferable;

public interface ImlDataContainer {
  public Transferable getTransferableData();
  public void setTransferableData(Transferable t);
}
```

`ImlDataContainer` uses a `Transferable` object to extract and set data of a component. For copy to clipboard and Drag & Drop transfer, the Flavor `DataFlavor.stringFlavor` is used, but you can provide any other Flavors.
List, Table and Tree rendering can also use the `stringFlavor` to retreive a string representation of the data (displayable in a cell), when the data container is not a graphical object.

<display>

Part of the document containing all graphical structures. To be compatible with ImlPlayer, one of the direct children of <display> should have @name set to "`MainImlComponent`" value. This is the way to identify the component which will be displayed at startup. If the component is not a frame, it is decorated automatically by a frame component.

Attributes :
laf : the class path to the Swing Look and Feel to use for decorating graphical objects, such as : `javax.swing.plaf.metal.MetalLookAndFeel`
Content Model :
Any list of JComponent elements. If a child is a <frame> element, frame.frame implementation is automatically choosen.

Containers

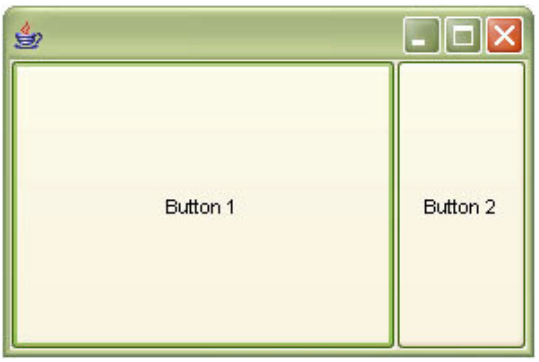
<box>

The box is a container which lays out its children sequentially, along a vertical or horizontal axis, depending upon @axis.
Each content item of the box is wrapped by an element <cell>, which contains a layout attribute, @flexible, specifying whether the component can be resized as needed or at its preferred size.

This document part :

```
<box>
  <cell flexible="true" >
    <button text="Button 1" />
  </cell>
  <cell >
    <button text="Button 2" />
  </cell>
</box>
```

is rendered by this :



By default, a box @axis is HORIZONTAL. First cell, containing "Button 1" is flexible, which means that the button's width is set to fill the maximum available space.
"Button 2" is not flexible, so its width is its preferred width.
Since axis is horizontal, heights are set to the boxes height.

Attributes :
Swing common attributes axis : can take VERTICAL or HORIZONTAL value,
Content Model :
List of <cell> of type cell.boxCell (implemented by ImlBoxCell).

<cell> (type : cell.boxCell)

container for any box content.

Attributes :
Swing common attributes flexible : indicate whether content can take available space or must have its preferred size.
Content Model :
zero or one graphical element (extending JComponent)

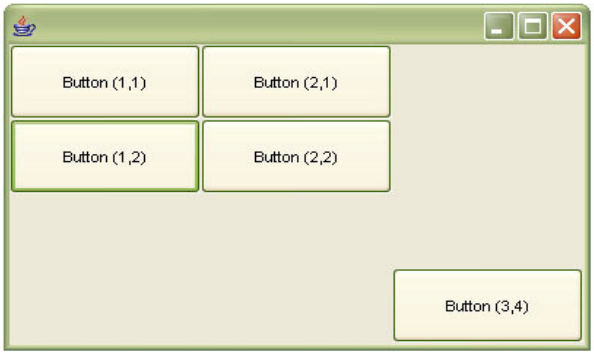
<gridbox>

This container displays its content in a grid. Each grid cell is defined by x,y coordinates starting at (1, 1). Consider this example :

This document part :

```
<gridbox>
  <cell >
    <button text="Button (1,1)" />
  </cell>
  <cell x="3" y="4" >
    <button text="Button (3,4)" />
  </cell>
  <cell y="2" >
    <button text="Button (1,2)" />
  </cell>
  <cell x="2" y="2" >
    <button text="Button (2,2)" />
  </cell>
  <cell x="2" >
    <button text="Button (2,1)" />
  </cell>
</gridbox>
```

corresponds to :



A gridbox can be configured with a particular row and col count, and specific row and col sizes. By default, a gridbox has no fixed col/row count. Such a count is computed to display every grid cell, at each cell add and remove operation.
By default, columns and rows are sized to fit the gridbox size. To set a specific col/row size, use @rowSize and @colSize. Value 0 disable the use of these attributes. In a same way, col and row count can be fixed by using @cols and @rows, which must be set to 0 to indicate an unlimited count.

Attributes :
Swing common attributes cols : number of columns of the grid, 0 means no fixed count (default) rows : number of rows of the grid, 0 means no fixed count (default) hgap : horizontal gap between two cells (0 by default) vgap : vertical gap between two cells (0 by default) colSize : size of a column in pixel, 0 means no fixed col size (default) rowSize : size of a row in pixel, 0 means no fixed row size (default)
Content Model :
Any count of <cell> of type cell.gridCell

<cell> (type : cell.gridCell)

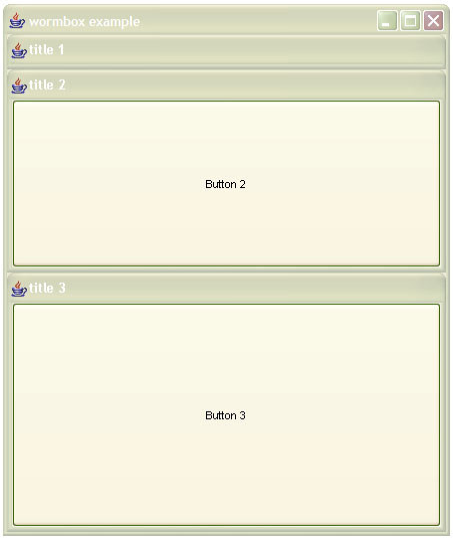
container for any gridbox content.

Attributes :
Swing common attributes x : horizontal position index in the grid, starts at 1 (default) y : vertical position index int the grid, starts at 1 (default) vflex : boolean indicating if cell content height can be resized to the cell height (true, the default) or must by set to its preferred height rows : number of vertical cells the content must occupy cols : number of horizontal cells the content must occupy hflex : boolean indicating if cell content width can be resized to the cell width (true, the default) or must be set to preferred width
Content Model :
zero or one graphical element (extending JComponent)

<wormbox>

This container displays its children in internal frames laid out vertically. Here is an example :

```
<wormbox >
  <cell title="title 1" >
    <button text="Button 1" />
  </cell>
  <cell title="title 2" >
    <button text="Button 2" />
  </cell>
  <cell title="title 3" >
    <button text="Button 3" />
  </cell>
</wormbox>
```



Each item of a wormbox is wrapped by a cell represented by an internal frame. Each item can be expanded or closed by clicking on the title bar, or by setting @expanded in <cell> element. The same title bar can be used to resize an item, by DnD operation.

Attributes :
Swing common attributes
Content Model :
Any count of <cell> of type cell.wormCell

<cell> (type : cell.wormCell)

container for any wormbox content.

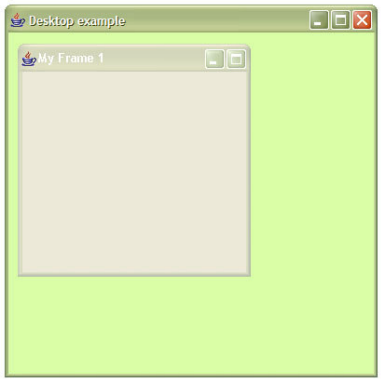
Attributes :
name : ID of this element icon : icon to display in an item title bar title : title of an item expanded : true if item is expanded, false otherwise
Content Model :
zero or one graphical element (extending JComponent)

<desktop>

The container of internal frames. Such document part :

```
<desktop>
  <frame visible="true" bounds="10,12,250,250" name="frame1" title="My Frame 1" />
</desktop>
```

renders as :



Attributes :
Swing common attributes selected : name of currently selected frame, if any
Content Model :
Any count of <frame> (of type frame.internalFrame)

<ebox>

An <ebox> is a box able to display an external IML document identified by an URL.
When setting @dmURL to a valid IML document URL, <ebox> loads it immediately, initializes its control section, then try to display the component whose @name is set to "MainImlComponent" value.

Attributes :
Swing common attributes dmUrl : URL of the document to load

<flowbox>

A box organizing its children in horizontal sequence, as shown below :

```
<flowbox >
 <cell >
  <label text="cover 1" icon="file:/d:/tmp/cover1.jpg" horizontalAlignment="CENTER"
verticalAlignment="BOTTOM" horizontalTextPosition="CENTER"
verticalTextPosition="BOTTOM" />
 </cell>
 <cell >
  <label text="cover 2" icon="file:/d:/tmp/cover2.jpg" horizontalAlignment="CENTER"
verticalAlignment="BOTTOM" horizontalTextPosition="CENTER"
verticalTextPosition="BOTTOM" />
 </cell>
 <cell >
  <label text="cover 3" icon="file:/d:/tmp/cover3.jpg" horizontalAlignment="CENTER"
verticalAlignment="BOTTOM" horizontalTextPosition="CENTER"
verticalTextPosition="BOTTOM" />
 </cell>
 <cell >
  <label text="cover 4" icon="file:/d:/tmp/cover4.jpg" horizontalAlignment="CENTER"
verticalAlignment="BOTTOM" horizontalTextPosition="CENTER"
verticalTextPosition="BOTTOM" />
 </cell>
</flowbox>
```

Attributes :
Swing common attributes align : one of this constants : LEFT, LEADING, CENTER, RIGHT, TRAILING hgap : horizontal distance between two components vgap : vertical distance between two components
Content Model :
Any count of <cell> (of type cell.flowCell)

<cell> (type : cell.flowCell)

container for any flowbox content.

Attributes :
Swing common attributes
Content Model :
zero or one graphical element (extending JComponent)

<scrollbox>

A box able to scroll its content.
Here are two examples, one using <scrollbox>, not the other :

```
<frame>
 <flowbox >
  <cell >
   <label icon="file:/tx/bc2.jpg"/>
  </cell>
 </flowbox>
</frame>
```

```
<frame>
 <scrollbox >
  <flowbox >
   <cell >
    <label icon="file:/tx/bc2.jpg"/>
   </cell>
  </flowbox>
 </scrollbox>
</frame>
```

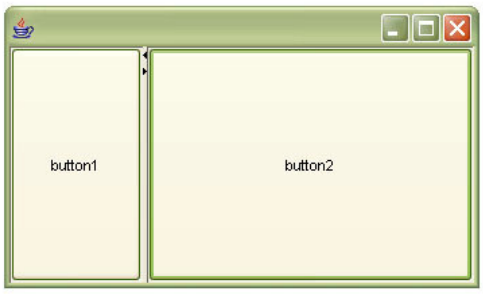
Attributes :
Swing common attributes
Content Model :
zero or one graphical element (extending JComponent)

<splitter>

A box separating its content in two resizable parts :

```
<splitter oneTouchExpandable="true" >
  <button text="button1" />
  <button text="button2" />
</splitter>
```

This document renders as :



The split can be done horizontally or vertically. The separation bar is called divider. The divider can be moved, but can be limited by minimum and maximum sizes of content. Several options can be applied on the divider.

Attributes :
Swing common attributes continuousLayout : boolean, if set to true, the layout of each part is done continuously when the separation bar is moved dividerLocation : location in pixels of the divider dividerSize : width in pixel of the divider

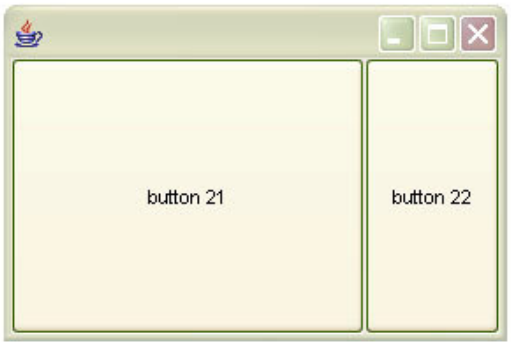
oneTouchExpandable : if true, the divider can be moved in one click to its extreme positions. orientation : VERTICAL or HORIZONTAL (the default) resizeWeight : percentage (float between 0 and 1) to place the divider, 0 is completely left/up, 1 is completely right/down
Content Model :
Zero, one or two graphical elements (extending JComponent)

<switchbox>

This box stacks its content to show only one child at a time. Considering this document below, two boxes enclosed by <cell> named box1 and box2, are children of the <switchbox>.

```
<switchbox>
  <cell name="box1" >
    <box axis="VERTICAL" >
      <cell flexible="true" >
        <button text="button 11" />
      </cell>
      <cell flexible="true" >
        <button text="button 12" />
      </cell>
    </box>
  </cell>
  <cell name="box2" >
    <box >
      <cell flexible="true" >
        <button text="button 21" />
      </cell>
      <cell >
        <button text="button 22" />
      </cell>
    </box>
  </cell>
</switchbox>
```

Left picture shows <switchbox> with @showing set to "box1", and right picture has @showing set to "box2" :



Attributes :
Swing common attributes showing : the name of the child <cell> to display.
Content Model :
Any count of <cell> of type cell.switchCell

<cell> (type : cell.switchCell)

container for any switchbox content.

Attributes :
Swing common attributes name : in this context, this common attribute has a particular importance : it permits to choose which content to display in a <switchbox>
Content Model :
zero or one graphical element (extending JComponent)

<tabbox>

A box displaying its children with tabs.

```
<tabbox >
  <cell title="Big cover 1" >
    <label icon="file:/d:/tmp/bcover1.jpg" />
  </cell>
  <cell title="Big cover 2" >
    <label icon="file:/d:/tmp/bcover2.jpg" />
  </cell>
</tabbox>
```

Attributes :
Swing common attributes selectedIndex : <cell> child index to display
Content Model :
Any count of <cell> of type cell.tabCell

<cell> (type : cell.tabCell)

container for any tabbox content.

Attributes :
Swing common attributes icon : URL of the icon to display in a tab title : title of the tab
Content Model :
zero or one graphical element (extending JComponent)

Frames

<frame> (type : frame.frame)

Element representing a JFrame object. Excepted as child of <desktop>, this is the allways choosen implementation of <frame>.

Attributes :
name : unique ID of this component resizable : boolean indicating whether this is resizable or not styleId : a string used to define a set of attribute values title : title of the window frameIcon : URL of icon to set in title bar visible : boolean to set this frame visible or not bounds : a Rectangle representing size and location of this frame
Content Model :
zero or one graphical element (extending JComponent)

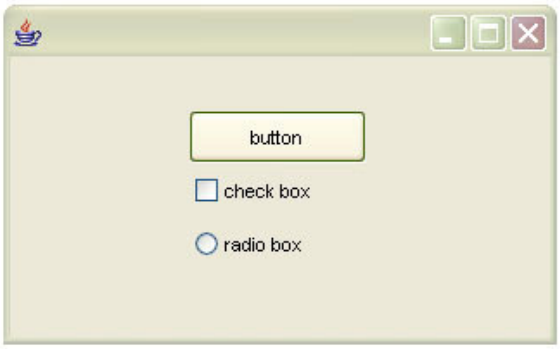
<frame> (type : frame.iframe)

Element representing a JInternalFrame object. This implementation is allways child of <desktop>.

Attributes :
name : unique ID of this component resizable : boolean indicating whether window is resizable or not maximizable : boolean indicating whether window is maximizable or not iconifiable : boolean indicating whether window is iconifiable or not styleId : a string used to define a set of style attribute values, such as background, font, margin, or any other attributes. Any component having a particular styleId value TODO title : title of the internal window framIcon : URL of icon to set in title bar visible : boolean to set this frame visible or not bounds : a Rectangle representing size and location of this frame in its desktop parent
Content Model :
zero or one graphical element (extending JComponent)

Buttons and labels

In this document, the different button elements :




```
<gridbox cols="3" rows="5" >
  <cell x="2" y="2" >
    <button text="button" />
  </cell>
  <cell x="2" y="3" >
    <checkbox text="check box" />
  </cell>
  <cell x="2" y="4" >
    <radio text="radio box" />
  </cell>
</gridbox>
```

<button>

Attributes :
Swing common attributes Button base attributes

<checkbox>

Attributes :
Swing common attributes Button base attributes selected : tells whether this component is selected or not selectedIcon : URL of icon to display when this is selected

<label>

Element representing a non editable text associated with an optional icon. This element can be used to display an image, some text, or both.

Attributes :
Swing common attributes text : content of this label icon : URL of label's icon horizontalAlignment : horizontal alignment of text horizontalTextPosition : horizontal position of the text relative to its icon verticalAlignment : vertical alignment of text verticalTextPosition : vertical position of the text relative to its icon

<radio>

Attributes :
Swing common attributes Button base attributes selected : boolean to set this radio selected or not group : name of a radio group, declared in control section of the IML document

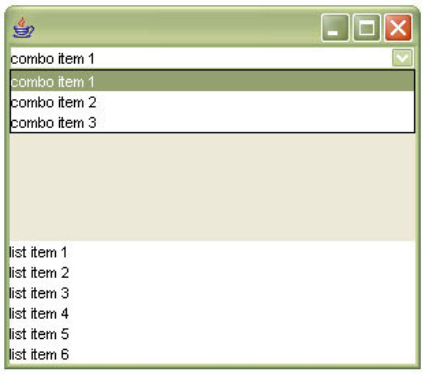
<toggle-button>

A two-state button.

Attributes :
Swing common attributes Button base attributes

Lists

Here is displayed in the same document a <combobox> and a <list> :




```
<box axis="VERTICAL" >
  <cell >
    <combobox >
      <cell selected="true" >
        <label text="combo item 1" />
      </cell>
      <cell >
        <label text="combo item 2" />
      </cell>
      <cell >
        <label text="combo item 3" />
      </cell>
    </combobox>
  </cell>
  <cell flexible="true" />
  <cell >
    <list >
      <cell >
        <label text="list item 1" />
      </cell>
      <cell >
        <label text="list item 2" />
      </cell>
      <cell >
        <label text="list item 3" />
      </cell>
      <cell >
        <label text="list item 4" />
      </cell>
      <cell >
        <label text="list item 5" />
      </cell>
      <cell >
        <label text="list item 6" />
      </cell>
    </list>
  </cell>
</box>
```

<combobox>

Standard combo box, an association of a popup list and a textfield.

Attributes :
Swing common attributes editable : if the textfield part is editable or not enabled : if the component can display its popup
Content Model :
Any count of <cell> of type cell.itemCell (implemented by ImlListItemCell)

<list>

Element representing a standard list, non editable

Attributes :
Swing common attributes
Content Model :
Any count of <cell> of type cell.itemCell (implemented by ImlListItemCell)

<cell> (type : cell.itemCell)

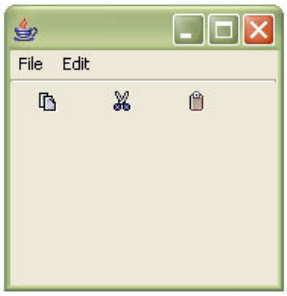
container for any item of a list. It can be used to wrap an item of a <list> or <combobox>. It is also used to keep trace of item selection.

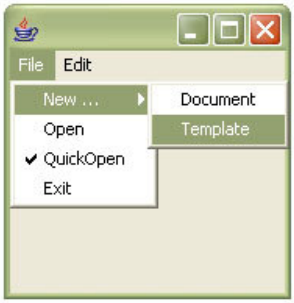
Attributes :
name : to identify every item selected : true to mark this item selected
Content Model :
zero or one element implementing ImlDataContainer interface

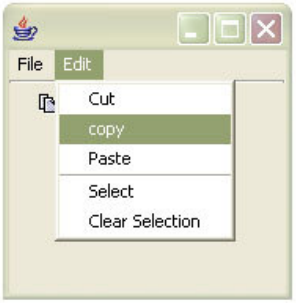
Menus and Toolbars

```
<box axis="VERTICAL" >
  <cell >
    <menubar>
      <menu text="File" >
        <menu text="New ..." >
          <menu-item text="Document" />
          <menu-item text="Template" />
        </menu>
        <menu-item text="Open" />
        <check-menu-item text="QuickOpen" />
        <menu-item text="Exit" />
      </menu>
      <menu text="Edit" >
        <menu-item text="Cut" />
        <menu-item text="copy" />
        <menu-item text="Paste" />
        <separator/>
        <menu-item text="Select" />
        <menu-item text="Clear Selection" />
      </menu>
    </menubar>
  </cell>
  <cell >
    <toolbar>
      <button icon="cp:toolbarButtonGraphics/general/Copy16.gif" />
      <button icon="cp:toolbarButtonGraphics/general/Cut16.gif" />
      <button icon="cp:toolbarButtonGraphics/general/Paste16.gif" />
    </toolbar>
  </cell>
</box>
```

Three different views of this document :







<menubar>

The container for every set of menus.

Attributes :
Swing common attributes
Content Model :
Any count of <menu> elements

<menu>

This element builds a menu or a sub-menu.

Attributes :
actionCommand : string sent on an Action event, when clicking on the menu background, disabledIcon, font, foreground, icon, name, opaque, pressedIcon, rolloverIcon, selectedIcon, styleId, text
Content Model :
Any count of elements <menu>, <menu-item>, <check-menu-item>, <separator>

<menu-item>

An item of a menu. Generally used to launch an action upon selection.

Attributes :
actionCommand : string sent on an Action event, when clicking on the menu background, disabledIcon, font, foreground, icon, name, opaque, pressedIcon, rolloverIcon, selectedIcon, styleId, text

<check-menu-item>

A checkable item of a menu. Keeps tracking of selection with @selected.

Attributes :
actionCommand : string sent on an Action event, when clicking on the menu selected : true if this item is checked background, disabledIcon, font, foreground, icon, name, opaque, pressedIcon, rolloverIcon, selectedIcon, styleId, text

<separator> (type separator.menuSeparator)

to add a space between two menu items. This element has no attribute.

<toolbar>

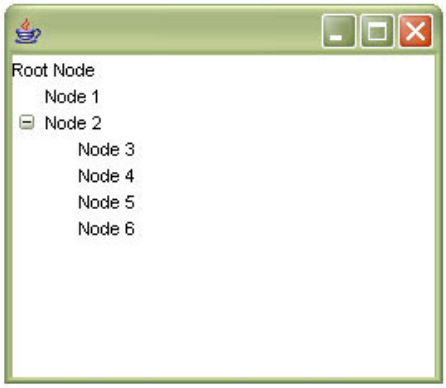
A container which displays its content in sequence. It's generally used to display a list of buttons with icons.

Attributes :
Swing common attributes orientation : VERTICAL or HORIZONTAL
Content Model :
Any graphical component. If a child is <separator>, separator.toolBarSeparator type is selected

<separator> (type separator.toolBarSeparator)

to add a space between two toolbar children. This element has no attribute.

Trees




```
<tree >
  <node >
    <node >
      <cell >
        <label text="Node 1" />
      </cell>
    </node>
    <node >
      <cell >
        <label text="Node 2" />
      </cell>
      <node >
        <cell >
          <label text="Node 3" />
        </cell>
      </node>
      <node >
        <cell >
          <label text="Node 4" />
        </cell>
      </node>
      <node >
        <cell >
          <label text="Node 5" />
        </cell>
      </node>
      <node >
        <cell >
          <label text="Node 6" />
        </cell>
      </node>
    </node>
    <cell >
      <label text="Root Node" />
    </cell>
  </node>
</tree>
```

<tree>

This is the container which displays data as a tree.

Attributes :
Swing common attributes editor : full name of a class implementing javax.swing.tree.TreeCellEditor, used to edit values renderer : full name of a class implementing javax.swing.tree.TreeCellRenderer, used to render values rootVisible : true if is wanted to show the root node
Content Model :
Zero or one <node>, considered as the root node.

<node>

A node of the tree.

Attributes :
name : ID of this element selected : true if this node is selected
Content Model :
Any count of <node>, considered as sub-nodes, and zero or one <cell> of type cell.treeCell.

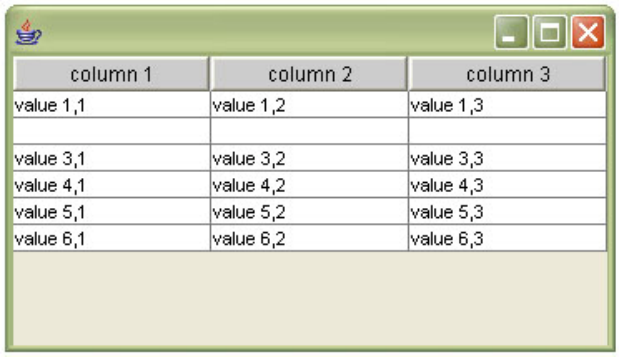
<cell> (type : cell.treeCell)

container for any node value.

Attributes :
name : to identify every item selected : true to mark this item selected
Content Model :
Zero or one element implementing ImlDataContainer interface.

Tables

Here is an example of table document :




```
<box axis="VERTICAL" >
  <cell >
    <scrollbox >
      <table >
        <header >
          <column name="column 1" />
          <column name="column 2" />
          <column name="column 3" />
        </header>
        <rows >
          <row >
            <cell >
              <label text="value 1,1" />
            </cell>
            <cell >
              <label text="value 1,2" />
            </cell>
            <cell >
              <label text="value 1,3" />
            </cell>
          </row>
          <row />
          <row >
            <cell >
              <label text="value 3,1" />
            </cell>
            <cell >
              <label text="value 3,2" />
            </cell>
            <cell >
              <label text="value 3,3" />
            </cell>
          </row>
          <row >
            <cell >
              <label text="value 4,1" />
            </cell>
            <cell >
              <label text="value 4,2" />
            </cell>
            <cell >
              <label text="value 4,3" />
            </cell>
          </row>
          <row >
            <cell >
              <label text="value 5,1" />
            </cell>
            <cell >
              <label text="value 5,2" />
            </cell>
            <cell >
              <label text="value 5,3" />
            </cell>
```

```
          </row>
          <row >
            <cell >
              <label text="value 6,1" />
            </cell>
            <cell >
              <label text="value 6,2" />
            </cell>
            <cell >
              <label text="value 6,3" />
            </cell>
          </row>
        </rows>
      </table>
    </scrollbox>
  </cell>
</box>
```

<table>

This is the container which displays data in a table. Table's defintion is organized in two parts. <header> contains all the columns description, and <rows> contains all the data of this table.

Attributes :
Swing common attributes
Content Model :
An element <header>, which contains columns description, followed by an element <rows> which contains all the rows.

<header>

The container for columns description.

Content Model :
At least one <column>

<column>

This element describes the definition of a table's column.

Attributes :
editor : full name of a class implementing javax.swing.tree.TreeCellEditor, used to edit values renderer : full name of a class implementing javax.swing.tree.TreeCellRenderer, used to render values maxWidth : max width of the column in pixel minWidth : min width of the column in pixel name : displayed name of the column ; do not confuse with @name in most IMLelements which are ID preferredWidth : preferred width of the column in pixel resizable : true if this column is resizable width : the width of the column in pixel

<rows>

The container for table's data.

Content Model :
Any count of <row>

<row>

In an IML table, data is organized in a list of rows. Each <row> contains cells, and each <cell> contains a value. Normally, a row should contains as many <cell> values as column count, but this is not mandatory. In the example, the second <row> is empty.

Content Model :
Any count of <cell> of type cell.tableCell. Normally, the <cell> children count is the same as the column count.

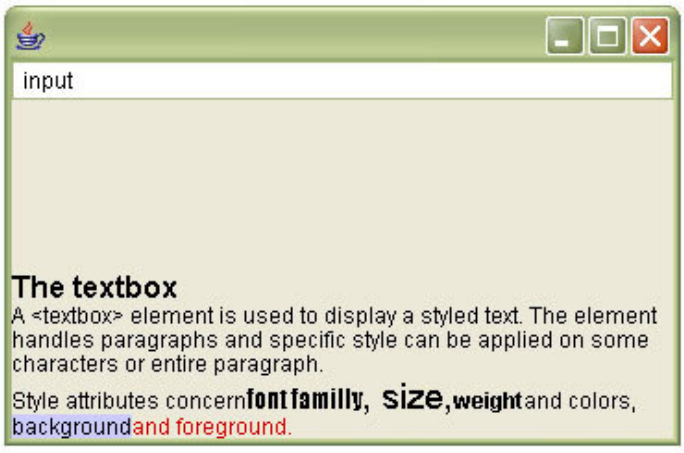
<cell> (type : cell.tableCell)

container for any table value.

Attributes :
editable : true if the cell is editable
Content Model :
Zero or one element implementing ImlDataContainer interface.

Text Inputs

```
      <box axis="VERTICAL" >
        <cell >
          <textfield />
        </cell>
        <cell flexible="true" />
        <cell flexible="true" >
          <textbox >
            <paragraph attributes="size:16.0; family:Arial; weight:bold" >
              <text >The textbox</text>
            </paragraph>
            <paragraph attributes="size:12.0; family:Arial" >
              <text >A &lt;textbox> element is used to display a styled text. The
element handles paragraphs and specific style can be applied on some characters or
entire paragraph.</text>
            </paragraph>
            <paragraph attributes="size:12.0; family:Arial" >
              <text >Style attributes concern</text>
              <text attributes="size:14.0; family:Impact" >font familly</text>
              <text attributes="size:19.0" >, size,</text>
              <text attributes="weight:bold" >weight</text>
              <text >and colors,</text>
              <text attributes="background:204,204,255" >background</text>
              <text attributes="foreground:204,0,0" >and foreground.</text>
            </paragraph>
          </textbox>
        </cell>
      </box>
```



<textfield>

An element to enter some text on a single line.

Attributes :
Swing common attributes text : the entered text

<textbox>

This element is used to display and enter some styled text, structured in paragraphs.

Attributes :
Swing common attributes editable : true if the textbox is editable
Content Model :
Any count of <paragraph> elements.

<paragraph>

Contains a paragraph of text.

Attributes :
attributes : style attributes to applied on entire paragraph
Content Model :
Any count of <text> elements.

<text>

Contains a part of text having specific style attributes.

Attributes :
attributes : style attributes to applied on this text PCDATA : the text content

The IML Player

IML Player is both an application and a set of useful methods, to load and process an IML document.
IML Player is automatically executed when invoking :

```
java –jar iml400.jar
```

Where iml400.jar is the JAR file containing all IML code.

As an application, IML Player can be executed with or without argument :
The unique argument is the URL of the IML document to process. If omitted, a dialog box is displayed to enter a valid URL to an IML document.

As a Java class, IML Player contains utility methods to load and process an IML document :

```
package com.virtualweaver.xotics.dialect.iml.player;

public class ImlPlayer {

  public static XoDMInstance loadAndInit(XoEnvironment environ,
                                         URL url,
                                         HashMap initOptions) throws XoException;
  public static Component selectDisplayableComponent(XoDMInstance imldoc)
  throws XoException;
  public static JFrame display(XoDMInstance imldoc) throws XoException;

}
```

`loadAndInit()` loads an IML document from an URL and initializes it by calling each `Imlcontrolable.initControl()`.
`selectDisplayableComponent()` returns the component which should be displayed, following these rules :

- either returns the direct child of <display> being a `java.awt.Component` whose @name is "`MainImlComponent`",
- or, if not found, returns the first direct child of <display> being a `java.awt.Component`.

`display()` selects the displayable component and returns it if instance of javax.swing.JFrame, or returns a JFrame containing it otherwise.