

XSB Java User Manual

1 March, 2006

XSB, Inc.
25 East Loop Road
Stony Brook, New York 11790
Phone Number: (631) 444 – 6800

Table of Contents

XSBJava User Manual.....	1
Learning XJ by Example.....	2
Concept Definitions.....	2
Gui Term overview.....	2
1 Common Properties.....	3
1.1 XJBorders.....	4
1.2 Colors.....	5
1.3 Fonts.....	6
1.4 XJHelp.....	7
Help file creation.....	7
1.5 XJTool Tips.....	9
2 GTOverview.....	10
3 XJComponents.....	11
3.1 XJContainers.....	12
3.1.1 Label Value Column.....	13
3.1.2 XJBorderLayout.....	15
3.1.3 XJBoxLayout.....	16
3.1.4 XJDesktop.....	17
3.1.5 XJFlowLayout.....	18
3.1.6 Label Value Row.....	19
3.1.7 XJSplitPane.....	20
3.1.8 XJTabbedPane.....	21
3.1.9 XJToolBar.....	22
3.2 XJForm Components.....	23
3.2.1 XJComboBox.....	24
3.2.2 XJButton.....	26
3.3 XJForm Dialogs.....	27
3.4 XJLists.....	28
3.4.1 XJEager Lists.....	29
3.4.2 Lazy Lists.....	30
3.5 XJTrees.....	31
3.5.1 XJEager Trees.....	32
3.5.2 Lazy Tree.....	33
3.6 XJBrowser.....	34
3.7 XJDesktop.....	35
3.8 Xj Label.....	36
4 XJDefinitions.....	37
5 XJFunctions.....	38
6 XJOperations.....	39
6.1 Popup Menu Operations.....	40
7 XJWidgets.....	41

1.1 XJBorders

Borders are specified with *border* property. Values of the property could be

- line
- etched
- raised
- lowered
- empty
- titled

The *empty* value creates an empty border with default margins (12, 11, 12, 11). To create a border with different margins specify border(Top, Left, Bottom, Right) value. If *titled* value is specified for a border then the title should be supplied with *caption* property. Examples:

```
gt(empty_border_pane,  
  [class='com.xsb.xj.containers.XJBorderLayout', border=empty],  
  [gt(titled_border_pane,  
    [class='com.xsb.xj.containers.LabelValueRow', nolabel, border=titled, caption='Titled Border'],  
    [gt(lined_border_pane,  
      [class='com.xsb.xj.containers.LabelValueColumn', nolabel, border=line],  
      [gt(empty_border_pane,  
        [class='com.xsb.xj.containers.LabelValueRow', nolabel, border=empty(100,100,100,100)]  
      )  
    ]  
  )  
])  
])
```

For components like buttons, that have borders on their own, using empty borders might create undesired visual effect. To manage spacing between them use *insets* property. Insets property is specified in the following form: insets(Top, Left, Bottom, Right).

For example,

```
gt(inset_example,  
  [class='com.xsb.xj.containers.LabelValueRow', nolabel],  
  [gt('Insets', [atom, insets(5,6,5,6)],[]),  
    gt('Insets', [class='com.xsb.xj.XJButton', insets(5,20,5,10)],[]),  
  % vs (not recommended)  
  [gt('Border', [atom, border=empty(5,6,5,6)],[]),  
    gt('Border', [class='com.xsb.xj.XJButton', border=empty(5,20,5,3)],[])  
])
```

For Java Look and Feel Guidelines on border and spacing see
<http://java.sun.com/products/jlf/ed2/book/HIG.Visual2.html>

1.2 Colors

Color values can be specified using reserved words for common colors or by using `rgb(Int, Int, Int)` term. The reserved color words are [green, red, black, blue, cyan, darkGray, gray, lightGray, magenta, orange, pink, white, yellow].

Foreground

Foreground color is specified using the color attribute in gui terms. To specify a red XJLabel write the following

```
gt('Red Label', [class='com.xsb.xj.XJLabel', color= red], [])
```

Background

Background color is specified using the background attribute in gui terms. To specify a red background for an atom field the user should write:

```
gt('Text', [atom, background=rgb(255,0,0), caption='Red Field'], [])
```

— HarpreetSingh – 22 Jul 2004

1.3 Fonts

Fonts for components are specified by using the `font(Name, Style, Size)` property in the [GuiTerm?](#). Name is a string, and can refer to any font available on the system (e.g. Times New Roman). Be aware font availability changes from system to system. Style is a string with values: p for plain, b for **bold**, i for *italic* and bi for **bold and italic**. If no value is specified plain is assumed. Size has to be an integer. If the name of the font is not specified the default font for the component is kept and specified values are changed. If the font name is specified, a new font is created for the component.

Change style to bold but keep the default font and size

```
gt('Bold Button', [class='com.xsb.xj.XJButton', font(_, b, _)],[])
```

Change style to italic and size to 14 but keep default font

```
gt('Italic: Size 14', [class='com.xsb.xj.XJLabel',font(_, i, 14)],[])
```

Change font to 'Times New Roman', size to 15, and keep plain style

```
gt('Times New Roman', [class='com.xsb.xj.XJLabel',font('Times New Roman', p, 15)], [])
```

-- HarpreetSingh -- 22 Jul 2004

1.4 XJHelp

XJ uses [JavaHelp?](http://java.sun.com/products/javahelp/) (http://java.sun.com/products/javahelp/) system to display online help. jhall.jar file that comes with the system provides that functionality. In order to use that system help files have to be created and XJComponents using these files should provide helpids that associate those component with corresponding help topics.

Help file creation.

Help contents can either be specified in MS Word document or using XSB Prolog declarations. If help topics are specified in MS Word document XSB converter is used to create prolog declarations.

MS Word document has to be formatted so that a header of each subtopic is smaller than the header of the supertopic. Topic headers are used to create a table of contents and their size implies the place in the table of contents hierarchy. Topic headers are also used to separate topics and to serve as topic ids in XJComponents specification. After the document is created it has to be saved as HTML using MS Word facilities. After the HTML file is created readhtml module (from com/xsb/xj/help package) is used to produce prolog guiHelp/3 declarations. Start XSB and execute the following:

```
assert(library_directory('C:\XSBSYS\XSBCVS\lib\utils')).%% for stdutils
import read_html/1 from readhtml.
read_html(HTMLDocFile). %% from HTMLFile.html will produce HTMLFile_gen.P
```

guiHelp/3 declarations could also be created directly by a prolog programmer. That is convenient if it is desirable to keep the help description besides guiTerm/4 declarations. guiHelp/3 facts have the following form:

```
guiHelp(someuniqueid,TOCList,DescriptionList).
```

For example,

```
guiHelp(newClassView,
        ['CDF Editor', 'Classes', 'New Class Window'],
        ['This window allows you to create a new class. ']).
```

- [DescriptionList?](#) could either be text (where HTML tags might be used) or URL or a list of text/HTML descriptions. The description may also contain `internal_ref(IdOrUrl?,Ref)` term, where `IdOrUrl?` is id of another guiHelp. For example, a description for id0 might have a subelement "[a link to something]" and if `guiHelp(id1,[node1,node2,node3],[some text. more text'])` then when html file for id1 is produced it is put in docs/node1/node2/node3/id1.html file, and in id0 file `internal_ref(id1,#subpart)` is substituted by "docs/node1/node2/node3/id1.html#subpart".

If the description is url(URL) then the user will see URL for the node documentation.

- TOCList is path to the topic in table of contents. XJ collects all the TOCLists from all guiHelp/3 facts to create a table of contents. For minor help topics that appear in a popup for gui components like atomic field it might be desirable sometimes not to make help description available from browsing help tree. In such cases TOCList should be [].
- someuniqueid is a unique topic id that will be specified in guiTerm/3 to connect a GUI term with the help topic. Help id help_top is reserved.

2 GTOverview

What gives any XJ application its visual appearance and function is the Gui Term (GT). Below is a graphic reviewing its essential components

The diagram illustrates the structure of a Gui Term (GT) and provides a code example. It is divided into two main sections.

Top Section: GT Structure and Examples

gt: Label [Properties] [Children]

Complex GTs can have one or more children. For example, XJContainers have a child for each XJComponent inside them.

Each GT can be described and/or specialized by setting various properties in the properties list. Some are GT specific, others are category specific, and several apply to all GTs.

Examples

```
atomic
class = 'class_path'
border = line(10,10,10,10)
caption = 'text'
```

For an atom-like GT, Label is its value.

For a more complex, list related GT, this is a constant that indicates its type.

For other more general complex GTs, Label is arbitrary.

Three examples of GTs are shown below, each with arrows pointing to its Label, Properties, and Children components:

```
gt: Label [Properties] [Children]
gt: Label [Properties] [Children]
gt: Label [Properties] [Children]
```

Bottom Section: Code Example

```
GT= gt(btns, ←Label
[ class='com.xsb.xj.containers.XJFlowLayout', ←Properties
  root, align=center, hgap=4,
  width=300, height=50 ],
[ gt('Sample button in an XJFlowLayout. Click to press.') ←Label
  (class='com.xsb.xj.XJButton'), []] ←Properties & Children
]). ←Children
```

— CharlesRojo – 01 Jul 2005

3.1.1 Label Value Column

Description

The LabelValueColumn component works similar to XJLabelValueRow, only this layout displays terms and corresponding labels as horizontally paired items in a column, with the label appearing to the left of the term.

Source: /XSBCVS/lib/xj2/com/xsb/xj/containers/LabelValueColumn.java

Javadoc: com.xsb.xj.containers.LabelValueColumn

Example

```
gt('',[class='com.xsb.xj.containers.LabelValueColumn',
fillarea],
[  gt('Button 1', [class='com.xsb.xj.XJButton', caption='Row 1',
  insets(1,3,1,3)],[]),
  gt('Button 2', [class='com.xsb.xj.XJButton', caption='Row 2',
  insets(1,3,1,3)],[]),
  gt('Button 3', [class='com.xsb.xj.XJButton', caption='Row 3',
  insets(1,3,1,3)],[])
]).
```

Properties

• *Parent*

- ◆ *fillarea*: – If the components inside the LabelValueColumn are smaller than the display area then they are resized both horizontally and vertically.
- ◆ *background*: – Replaces the default background of the panel with the specified image.
- ◆ *nolabel*: – If not specified LabelValueColumn adds a label for each component. The text of the label is taken from the caption property of the component. The font for the label is taken from the font property for the container.
- ◆ *anchor=(direction)*:– This anchors the term column to the specified direction. The directions to choose from are center, north, northeast, east, southeast, south, southwest, west, or northwest.
- ◆ *label_anchor=(direction)*:– This is similar to the anchor property, except that it applies to the label column instead of the term column. Note that each LabelValueColumn consists of two columns: one for labels, and another for the associated terms.

• *Children*

- ◆ *insets*: – Child terms of LabelValueColumn can use the insets property to reserve space around themselves. insets(Top,Left,Bottom,Right), values must be integers.
- ◆ *caption=(text)*: – The text that will appear in the label above this child term. For example, caption = 'Row 1' in the above example puts a "Row 1" label above Button 1 (the first child in the list).

3 XJComponents

3.1.2 XJBorderLayout

Description

The XJBorderLayout component is a panel(`javax.swing.JPanel`) using [BorderLayout](#)?(`java.awt.BorderLayout`) to display its children. The panel is split into 5 parts [north, south, east, west and center(default)].

Source: `/XSBCVS/lib/xj2/com/xsb/xj/containers/XJBorderLayout.java`

Javadoc: `com.xsb.xj.XJBorderLayout`

```
gt(' ', [class='com.xsb.xj.containers.XJBorderLayout',
  hgap=10,vgap=10,border(5,5,5,5)],
[  gt('Center',[class='com.xsb.xj.XJLabel'],[]),
  gt('North',[class='com.xsb.xj.XJLabel',layout=north,
    bounds(100,100,100,100)],[]),
  gt('East',[atom],[])
])
```

- Options

- ◆ Parent

hgap

The horizontal gap between components specified in pixels.

vgap

The vertical gap between components specified in pixels.

- ◆ Children

bounds/4

Preferred size of the child component, specified as `bounds(X,Y,Width,Height)`

layout

Specifies the part of the panel the child component should be displayed. If this property is not specified for the child component it is placed in the center position.

- Tips

- ◆ Setting the center component

```
set_center_component(Border_gui, New_gt):-
  buildGUITerm(New_gt, GTM),
  javaMessage(Border_gui, setCenterComponent(GTM)).
```

- ◆ Setting the western component

```
set_west_component(Border_gui, New_gt):-
  buildGUITerm(New_gt, GTM),
  javaMessage(Border_gui, setWestComponent(GTM)).
```

3.1.3 XJBoxLayout

Description

A layout manager that allows multiple components to be laid out either vertically or horizontally. The components will not wrap so, for example, a vertical arrangement of components will stay vertically arranged when the frame is resized. Nesting multiple panels with different combinations of horizontal and vertical gives an effect similar to XJLabelValueRow/XJLabelValueColumn, without the complexity. This component is based on javax.swing.BoxLayout.

Source: /XSBCVS/lib/xj2/com/xsb/xj/containers/XJBoxLayout.java

Javadoc: com.xsb.xj.XJBoxLayout

```
gt(' ', [class='com.xsb.xj.containers.XJBoxLayout', align=y_axis],
[   gt('Top', [class='com.xsb.xj.XJLabel'], []),
    gt(middle, [class='com.xsb.xj.containers.LabelValueRow'],
      [   gt(' ', [atom, caption='Middle'], [])
        ]),
    gt('Bottom', [class='com.xsb.xj.XJLabel'], [])
])
```

- Options

- ◆ Parent

- align*

- Specifies which way child components are laid out. Currently the layout allows for the children to be either laid out vertically(*y_axis*) or horizontally(*x_axis*), this is the default.

- ◆ Children

- ◇ No options available.

3.1.4 XJDesktop

Description

XJDesktop is the GUI window for any XJ Application. It provides the graphical context in which all GUI terms reside. It also provides a default menu at the top, the functionality of which can be altered or added to. Since all GUI terms reside inside XJDesktop, it itself isn't one. It must be created by evoking the XJDesktop constructor through a javaMessage. Any GUI term can be placed inside it with the predicate "showInternalFrame(_SomeGT)" as demonstrated in the example below. See the example for a method of invoking the desktop and adding a basic GUI Term into it.

Source: /XSBCVS/lib/xj2/com/xsb/xj/containers/XJDesktop.java
Javadoc: com.xsb.xj.XJDesktop

Example

```
%the XJ program enters at xjmain...
xjmain :-
    ipPrologEngine(Engine),    %acquires a handle to the prolog engine, as Engine
    xjConsole(Console),      %acquires a handle to the console, as Console
    ipObjectSpec(int, W, [800], _), %acquires the Object for an integer with a value of 800, as W and
    ipObjectSpec(int, H, [800], _),

    % below, javaMessage is invoked to call the XJDesktop constructor of the XJDesktop class,
    % with 'XJ Wiki's example dekstop' as the first argument, W (800) as the width arg, H (800) as th
    % Engine as the PrologEngine arg, and Console as the XJTopLevel arg. Other constructors are
    % available, and can be viewed in the javadocs.
    javaMessage('com.xsb.xj.XJDesktop', D, 'XJDesktop'(string('XJ Wiki's example dekstop'), W, H, Engi
    asserta(xjDesktop(D)),

    % now the XJDesktop is created, but GUI Terms must be added. This is accomplished by doGoal.
    doGoal.

doGoal :- GT = (make some GT here), showInternalFrame(GT).
```

Properties

- *Not Applicable*

-- CharlesRojo -- 01 Jul 2005

3.1.5 XJFlowLayout

Description

A flow layout arranges components in a left-to-right flow, much like lines of text in a paragraph. Flow layouts are typically used to arrange buttons in a panel. It will arrange buttons left to right until no more buttons fit on the same line. Each line is centered.

Source: /XSBCVS/lib/xj2/com/xsb/xj/containers/XJFlowLayout.java

Javadoc: com.xsb.xj.XJFlowLayout

```
gt(btns,[class='com.xsb.xj.containers.XJFlowLayout', root,
  align=left, hgap=4, width=300,height=50],
[
  gt('< Back',[class='com.xsb.xj.XJButton',root, myGUI(Back),
    disabled],[ ]),
  gt('Next >',[class='com.xsb.xj.XJButton',root,
    myGUI(Next)],[ ]),
  gt('Cancel',[class='com.xsb.xj.XJButton',root,
    myGUI(Cancel), CancelOP],[ ])
])
```

- Options

- ◆ Parent

- align*

- Alignment of the layout, possible values are:

- left*

- components are left-justified

- right*

- components are right-justified

- center*

- components are centered (default)

- leading*

- each row of components should be justified to the leading edge of the container's orientation, for example, to the left in left-to-right orientations.

- trailing*

- each row of components should be justified to the trailing edge of the container's orientation, for example, to the right in left-to-right orientations.

- hgap*

- Specify the horizontal gap between components. Default value is 5.

- vgap*

- Specify the vertical gap between components. Default value is 5.

- ◆ Children

- ◇ No available options.

3.1.6 Label Value Row

Description

The LabelValueRow component is a panel(javax.swing.JPanel) using a grid layout(java.awt.GridBagLayout) to display terms and corresponding labels in a horizontal row. Alternatives for this component are XJFlowLayout and XJBoxLayout.

Source: /XSBCVS/lib/xj2/com/xsb/xj/containers/LabelValueRow.java

Javadoc: com.xsb.xj.containers.XJLabelValueRow

```
gt('', [class='com.xsb.xj.containers.LabelValueRow',
      fillarea, background='images/back.jpg'],
[   gt('', [class='com.xsb.xj.XJButton', caption='Test1',
          insets(5,3,5,3)],[]),
    gt('', [class='com.xsb.xj.XJButton', caption='Test2',
          insets(3,5,3,5)],[])
])

gt('', [class='com.xsb.xj.containers.LabelValueRow',
      fillarea, nolabel],
[   gt('', [atom, large, caption='text area'],[])
])
```

- Options:

- ◆ Parent:

- fillarea*

- If the components inside the LabelValueRow are smaller than the display area then they are resized both horizontally and vertically.

- background*

- Replaces the default background of the panel with the specified image.

- nolabel*

- If not specified LabelValueRow adds a label for each component. The text of the label is taken from the caption property of the component. The font for the label is taken from the font property for the container.

- ◆ Children:

- insets*

- Child terms of LabelValueRow can use the insets property to reserve space around themselves. insets(Top,Left,Bottom,Right), values must be integers.

3.1.7 XJSplitPane

Description

A type of panel, derived from JSplitPane, designed to contain two XJComponents (which can also be of XJSplitPane type). The amount of area allocated to each Component depends on what is set for the "divider" property. The Pane can also be split either horizontal or vertical depending on whether or not the "vertical" property is set.

When constructing this type of GT, the Label field is arbitrary, the Properties list can contain those listed below plus the universal ones, and there can be only two GTs inside the Children list.

Source: /XSBCVS/lib/xj2/com/xsb/xj/containers/XJSplitPane.java

Javadoc: com.xsb.xj.XJSplitPane

Example

```
gt('Split Pane',
[
  class='com.xsb.xj.containers.XJSplitPane',
  root,
  divider=50
],
[
  gt('',[class='com.xsb.xj.containers.XJBorderLayout',
  hgap=10,vgap=10,border(5,5,5,5)],
  [
    gt('Left Panel',[class='com.xsb.xj.XJLabel',
    border(5,5,5,5)], [])
  ]),
  gt('',[class='com.xsb.xj.containers.XJBorderLayout',
  hgap=10,vgap=10,border(5,5,5,5)],
  [
    gt('Right Panel',[class='com.xsb.xj.XJLabel',
    border(5,5,5,5)], [])
  ])
]).
```

Properties

• *Parent*

- ◆ *vertical*: – Indicates that the two inner components should be positioned such that component one lies above component two, with a horizontal division between them
- ◆ *divider=(number from 0–100)*: – Indicates the percentage in the SplitPane that the first component should occupy. For example, divider = 50 would allocate 50% of the SplitPane's space to component one. Likewise, divider = 40 would allocate 40% to component one, and so on.
- ◆ *expandable*: – Setting this property adds widgets to the divider that allow for one of the two components to be easily collapsed/expanded.
- ◆ *dividerSize=(positive number)*: – Specifies the size, in pixels, of the separating divider.

• *Children*

- ◆ No options available.

3.1.8 XJTabbedPane

Description

The XJTabbedPane component emulates the [JTabbedPane](#)? (javax.swing.JTabbedPane) component from java swing library. The component lets the user switch between a group of components by clicking on a tab with a given title and/or icon.

Source: /XSBCVS/lib/xj2/com/xsb/xj/XJTabbedPane.java

Javadoc: com.xsb.xj.XJTabbedPane

This `gt` creates a tabbed pane with three tabs:

```
gt('',[class='com.xsb.xj.XJTabbedPane',placement=top],
[ gt('',[class='com.xsb.xj.containers.XJBorderLayout',
  caption='First Tab', icon='images\one.gif'], [...]),
  gt('',[class='com.xsb.xj.containers.LabelValueRow',
    tip='this is third tab', caption='Second Tab'], [...]),
  gt('',[class='com.xsb.xj.containers.LabelValueColumn',
    disabled, caption='Third Tab'], [...])
])
```

- Options

- ◆ Parent

- placement*

- Specifies where the tabs for the child components should be placed. Possible values are top, bottom(default), left and right.

- selected*

- By default the first tab is selected, but the user can specify any tab as selected by default by using `selected(Tab Number)`.

- ◆ Children

- caption*

- Used as the title of the tab.

- tip*

- Used as the tool tip for the tab.

- icon*

- Icon for the tab.

- disabled*

- Sets the tab to be disabled

- Tips

- ◆ Listen to selection changes

3.1.9 XJToolBar

Description

The XJToolBar component emulates JToolBar from the java swing library. The component is ideal to hold buttons for commonly used commands in a window. The following [GuiTerm?](#) creates a horizontal toolbar with four buttons (Load, Merge, Save and Unload) and separators on either side of the Save button.

Source: /XSBCVS/lib/xj2/com/xsb/xj/containers/XJToolBar.java

Javadoc: com.xsb.xj.XJToolBar

```
gt(' ', [class='com.xsb.xj.containers.XJToolBar', notfloatable,
  rollover],
[  gt('Load', [class='com.xsb.xj.XJButton', caption='', LoadOP
    myGUI(LoadBtn), tip='Load OMS',
    icon='../images/Open16.gif'], []),
  gt('Merge', [class='com.xsb.xj.XJButton', caption='',
    myGUI(MergeBtn), tip='Merge OMS',
    icon='../images/Import16.gif', separator, MergeOP], []),
  gt('Save', [class='com.xsb.xj.XJButton', caption='',
    myGUI(SaveBtn), tip='Save OMS', icon='../images/Save16.gif',
    separator, SaveOP], []),
  gt('Unload', [class='com.xsb.xj.XJButton', root, caption='',
    myGUI(UnloadBtn), tip='Unload current OMS', UnloadOP], [])
])
```

- Options

- ◆ Parent

- notfloatable*

- By default tool bars are floatable that is they can be dragged into a different position within the same container or out into its own window. Some look and feels might not implement floatable tool bars; they will ignore this property. This options disables the feature.

- rollover*

- Hides borders for buttons on the toolbar and displays them only when the mouse is over the button.

- vertical*

- The default orientation for the tool bar is horizontal, this property changes it to vertical.

- ◆ Children

- separator*

- Adds a separator to the toolbar after the child.

3.2.1 XJComboBox

Description

XJComboBox component(`javax.swing.JComboBox`) allows the user to select a value from a given list or add an element to the list. The model defining the data in the list can be either eager or lazy.

Source: `/XSBCVS/lib/xj2/com/xsb/xj/XJComboBox.java`

Javadoc: `com.xsb.xj.XJComboBox`

The following is an example of an eager combo box:

```
gt('.', [class = 'com.xsb.xj.XJComboBox', editable, width=150, list(gt(_, [atom], []))], [a, [a,b,c]]).
```

The list term, `list(gt(_, [atom], []))` inside the component definition defines the template used to display items in the list. The third argument of the gui term contains the data for the list, the first element in the list is the selected item (a), and the second element of the list is the list of possible values (a,b,c).

The following are examples of a lazy combo box:

```
gt(BestMatch, [class='com.xsb.xj.XJComboBox', width=165,
  popupWidth=250, lazylist(gt(_, [opaque], [])),
  lazy(Template, Goal, Context)], [])

gt(InitialLabel, [class='com.xsb.xj.XJComboBox', prologcached,
  opaque, root, operation(term(A,B), (writeln(A-B)), selectionChanged),
  lazylist(gt(objecta, [class='com.xsb.xj.XJLabel', nolabel, justify=left],
    [ gt(_, [opaque, invisible], []) %hidden
    ])),
  lazy(X, test_box(X, _), _)], [InitialValueOfHiddenNode])
```

The list term, in the eager combo box is replaced with the lazylist term above. And a lazy goal is added to the [GuiTerm?](#) property list to specify the data model.

The following is an example of an eager combobox with list of terms:

```
gt('.', [caption='Test', class='com.xsb.xj.XJComboBox', root,
  width=150, popupWidth=200,
  list(gt(_, [class='com.xsb.xj.XJLabel', justify=left, caption=''],
    [ gt(_, [atom, invisible, opaque], [])
    ]))
  ]), [SelectedTerm, Term_list]) ].
```

`Term_list` here is a list of terms that are of the form 'Label'(A). As you can see from the template for the list the label will be displayed and the value (A) will not be displayed.

- Options

- ◆ Parent

- editable*

- Allows the user to modify elements in the list.

- popupWidth*

Sets the width of the drop down list. This option changes the ui used for the combo box making appear different from when the option is not specified.

- ◆ Children
 - ◇ No options available.

- Tips

- ◆ Adding an item to the list `add_new_item(Combo_box_gui, New_item):-`
`buildTermModelArray([New_item], TMA), javaMessage(Combo_box_gui, Model, getModel),`
`javaMessage(Model, addTerms(TMA)).`
- ◆ Remove an item from the list:

```
delete_item(Combo_box_gui, Item):-  
    buildTermModelArray([Item], TM),  
    javaMessage(Combo_box_gui, Model, getModel),  
    javaMessage(Model, deleteTerms(TM)).
```

- ◆ Set the selected item in the list:

```
set_selected(Combo_box_gui, Item):-  
    buildTermModel(Item, TM),  
    javaMessage(Combo_box_gui, Model, getModel),  
    javaMessage(Model, setSelectedItem(TM)).
```

- ◆ Get the selected item in the list:

```
get_selected_item(Combo_box_gui, Selected_item):-  
    javaMessage(Combo_box_gui, GT, getGT),  
    javaMessage(GT, TM, getTermModel),  
    recoverTermModel(TM, [Selected_item, Item_list]).
```

— HarpreetSingh – 22 Jul 2004

3.2.2 XJButton

Description

A basic button, derived from the JButton class. It has several properties available that change its appearance, and arbitrary operations may be associated with pressing it, as demonstrated below (TODO: demonstrate how to do it).

Source: /XSBCVS/lib/xj2/com/xsb/xj/XJButton.java

Javadoc: com.xsb.xj.XJButton

Example

```
gt(btns,
  [ class='com.xsb.xj.containers.XJFlowLayout', root,
    align=center, hgap=4, width=300,height=50],
  [ gt('Sample button in an XJFlowLayout. Click to press.',[class='com.xsb.xj.XJButton'],[])
  ]).
```

Properties

• *Parent*

- ◆ *margin(Top, Left, Bottom, Right)*: – Specifies the margins surrounding the button.
- ◆ *zeroInset*: – To be used as an alternative to margin, specifying that the margins are to be 0 on all sides.
- ◆ *hpos={right, left, center, or leading}*: – Setting this property to one of the specified values...
- ◆ *vpos={top, bottom, center}*: – Setting this property to one of the specified values...
- ◆ *width=(positive number)*: – Specifies the button width.
- ◆ *height=(positive number)*: – Specifies the button height.
- ◆ *icon=(file path)*: – Specifies an icon to show inside the button.

• *Children*

- ◆ No options available.

— CharlesRojo – 01 Jul 2005

3.4 XJLists

3.4.1 XJEager Lists

```
gt('.', [root,
  list(
    gt(match, [constant, caption='', class='com.xsb.xj.ValueRow', root, width=500],
      [
        gt('', [opaque, invisible], []),
        gt(_, [class='com.xsb.xj.XJCheckBox', nocaption, myGUI(Check), tip='Triple click to select i
        gt(_, [atom, readonly, width=35, caption='Column 1'], []),
        gt(_, [atom, readonly, width=45, caption='Column 2'], []),
        gt(_, [atom, readonly, width=55, caption='Column 3'], [])
      ]
    )
  ], Datalist)

Datalist = [match(Hidden, CheckBox, Column1, Column2, Column3)]
```

- Notes

- ◆ Eager lists are not sortable.
- ◆ When dealing with lists user has the option to move columns around and view them in any order, by simply dragging the column.
- ◆ Users also have the option to show and hide columns in list. To activate this feature the user right clicks on one of the column headers.

3.4.2 Lazy Lists

Lazy lists differ from eager lists in that only solutions for visible records in the list are being passed to Java. With lazy lists the user does not have to wait for a long time for a large list to be transferred to Java side to be displayed. The other difference between eager and lazy lists is that eager lists persist to prolog side on per-list basis when the list is being closed, whenever updatable lazy lists persist on per-record basis (which is mostly used in non-prolog cached lists).

There are currently two types of lazy lists – those that use prolog cache and those that do not. If a list is prolog cached it should be identified by *prologcached* property of the list. Gui terms for lazy lists are defined as follows:

```
gt('XJ$LAZY',[lazylist(Template(s)),...],lazy(TemplateTerm, Goal, Context))
```

The instantiations of `TemplateTerm` represent data for rows in the table. `TemplateTerm` is being instantiated by means of calling `Goal`. `Template` argument of the `lazylist` term is a gui term that describes how `TemplateTerm` is to be rendered.

For example:

```
gt('XJ$LAZY',[prologcached, myGUI(GUI), sortable, sorted([desc(3)]),
  lazylist(
    gt(lazy_list,[constant,caption='My Lazy list',class='com.xsb.xj.ValueRow',root,updatable],
    [  gt('',[opaque, invisible],[]),
      gt('',[atom, readonly, caption='Column 1', width=4|Ops],[]),
      gt('',[atom, readonly, caption='Column 2', width=50|Ops],[]),
      gt('',[atom, readonly, caption='Column 3', width=220|Ops],[])
    ])
  ],lazy(view(A,B,C,D), equiv_goal(A,B,C,D, Context),Context)),
```

- Options

- ◆ Parent

- sortable*

- When set the user has the option to click on the column headers to sort the table. Sorting specification can be given for each column are used when the user clicks the column header.

- sorted*

- Initial sort specification for a prolog cached list. The sort specifications are the same as those used by machine: `parsort/4`.

- Notes

- ◆ Lazy lists can not contain editable components

- ◆ `XJCheckBoxes`? and `XJComboBoxes` can be added to lists but the user must triple click to activate the component.

- ◆ All lazy lists are sortable.

3.5.1 XJEager Trees

There are currently no eager trees in XJ.

3.5.2 Lazy Tree

Description

A tree that can be populated with various types of GTs (aka, a polymorphic tree), and displays prolog data that is retrieved in a lazy fashion. In other words, the branches of the tree fill themselves only when they need to be displayed.

Source:

Javadoc:

Example

```
gt('XJ$LAZY',
  [caption='Lazy Tree', prologcached, sorted,
  lazytree(
    [ gt(_, [class='com.xsb.xj.ValueRow', typename='any_type',
      root, readonly, typicalSize=1, borderless],
      [ gt(_, [class='com.xsb.xj.XJLabel', borderless, tip='Class'
        ], []),
        gt(_, [ opaque], [])
      ]
    )
  ],
  lazy(lt_subclass_goal(_,_,_), lt_label_goal(_,_,_), null)).
```

```
lt_subclass_goal('Root Node', null, any_type) :-
  !,
  true.
```

```
lt_subclass_goal(Child, Parent, any_type) :-
  childParent(Child, Parent).
```

```
lt_label_goal(Child, any_type, Node) :-
  Node =.. [' ', Child, Child].
```

```
childParent('Leaf Node 1', 'Node b').
childParent('Leaf Node 2', 'Node b').
childParent('Node b', 'Node c').
childParent('Node c', 'Root Node').
```

Properties

- *Parent*

- ◆ lazytree(A_Gui_Term [List_of_Gui_Terms]): –

- *Children*

- ◆ lazy(subclass_goal(,_,), label_goal(,_,), Context): –

— CharlesRojo – 05 Aug 2005

3.6 XJBrowser

Description

A quite powerful component, XJBrowser is a ready-to-go web browser panel that allows for URL entering, link-following and authentication. It has the commonly-found Back, Forward, and Home buttons available in a toolbar at the top, along with an Address bar which may or may not be editable, depending on what the 'web' property is set to.

Source: /XSBCVS/lib/xj2/com/xsb/xj/XJBrowser.java

Javadoc: com.xsb.xj.XJBrowser

Example

```
gt('Browser', [class='com.xsb.xj.XJBrowser', file='http://xce.xsb.com', web=true, root], []).
```

Properties

- **Parent**

- ◆ *web=(true or false)*: – Sets whether or not the user can specify arbitrary web addresses to go to.
- ◆ *file=(web address)*: – Sets up the starting address to initially browse.

- **Children**

- ◆ XJBrowser has no children.

— CharlesRojo – 29 Jun 2005

3.7 XJDesktop

Description

XJDesktop is the GUI window for any XJ Application. It provides the graphical context in which all GUI terms reside. It also provides a default menu at the top, the functionality of which can be altered or added to. Since all GUI terms reside inside XJDesktop, it itself isn't one. It must be created by evoking the XJDesktop constructor through a javaMessage. Any GUI term can be placed inside it with the predicate "showInternalFrame(_SomeGT)" as demonstrated in the example below. See the example for a method of invoking the desktop and adding a basic GUI Term into it.

Source: /XSBCVS/lib/xj2/com/xsb/xj/containers/XJDesktop.java

Javadoc: com.xsb.xj.XJDesktop

Example

```
%the XJ program enters at xjmain...
```

```
xjmain :-
```

```
    ipPrologEngine(Engine),    %acquires a handle to the prolog engine, as Engine
    xjConsole(Console),    %acquires a handle to the console, as Console
    ipObjectSpec(int, W, [800], _), %acquires the Object for an integer with a value of 800, as W and
    ipObjectSpec(int, H, [800], _),
```

```
% below, javaMessage is invoked to call the XJDesktop constructor of the XJDesktop class,
% with 'XJ Wiki's example dekstop' as the first argument, W (800) as the width arg, H (800) as th
% Engine as the PrologEngine arg, and Console as the XJTopLevel arg. Other constructors are
% available, and can be viewed in the javadocs.
```

```
javaMessage('com.xsb.xj.XJDesktop', D, 'XJDesktop'(string('XJ Wiki's example dekstop'), W, H, Engi
asserta(xjDesktop(D)),
```

```
% now the XJDesktop is created, but GUI Terms must be added. This is accomplished by doGoal.
doGoal.
```

```
doGoal :- GT = (make some GT here), showInternalFrame(GT).
```

Properties

- *Not Applicable*

— CharlesRojo – 01 Jul 2005

3.8 Xj Label

Source: /XSBCVS/lib/xj2/com/xsb/xj/XJLabel.java

Javadoc: com.xsb.xj.XJLabel

XJLabel is a subclass of JLabel in Java. It is used to represent text or an image or both. By default, the label text is justified right. To change its relative location to center or left set justify property to justify = center or justify = left. For example,

```
gt('Text Label', [class='com.xsb.xj.XJLabel', justify = left], [])
```

JLabel can also be used to display an image (with or without text). Use icon property with the image location as a value to specify the image location.

If a label is used to describe an input field or some other XJComponent it is recommended use labelfor([GuiRef?](#)) property to specify that. That improves accessibility of the GUI. Also it is possible to specify mnemonic property for the label. If labelfor property is specified along with mnemonic property the component will receive focus when the mnemonic is activated.

See also <http://java.sun.com/j2se/1.4.2/docs/api/javaw/swing/JLabel.html> for description of JLabel. —
TatyanaVidrevich – 09 Aug 2004

4 XJDefinitions

Lazy List

Similar to the list concept in Prolog, lazy lists provide the same functionality, but retrieve elements of the list on the basis of a request. Instead of retrieving the full contents of the list and displaying as many elements as possible, this construct will retrieve only the portions of the list that the user needs.

Annotation

For a given term, T, its annotation is another term derived from it, taking the form below,

```
gt(TopNode, AnnotationProperties, [TransformedChild1, ...])
```

such that :

TopNode is either the constant '.' for lists, the constant 'XJ\$LAZY' for lazy lists, or the label for the term. (Ex: For the list [1,2,3], '.' is the TopNode. For a list whose elements are retrieved from a database in a lazy model, 'XJ\$LAZY' is the TopNode. For the three following terms—>random_term, random_term(7), and random_term(child_term(1), child_term(2)), the label for each is 'random_term,' so the TopNode in the annotation for each would be 'random_term').

AnnotationProperties is a list of properties that apply to the term. These properties vary with the type of term.

TransformedChild1...N are annotated terms made from T's original children.

Every term has an annotation, but with respect to XJ, the 'GUI Term' annotation (gt(...)), with its indicative properties & children, is the principle mechanism for specifying the appearance and function of an XJ application.

Template

An annotation intended to "match" several different terms. A template, GT, for a term, T, is an annotation with variables, such that there is some more instantiated variant of GT that is the annotation of T.

XJModel

An auxiliary program which defines a set of templates covering the data tuples for the program's visible predicates.

— CharlesRojo – 05 Jul 2005

5 XJFunctions

tbd

6 XJOperations

6.1 Popup Menu Operations

To add a pop menu operation to a list, tree or any other component set the third argument of the operation to either `menu(Text)` or `menu(Text,[Options])`. The currently available options are `mnemonic(M)` and `image(I)`.

To add sub menus inside a popup menu set the third arg to `menu(Choice,a(b,c,d))`. This creates a sub menu a with menu items b,c and d. The Choice var is unified with the selection in the form of a list representing the path (e.g. [a,b]).

A complete menu operation could look something like this:

```
operation(term(_,_), writeln(abc), menu('Write abc', [image('abc.gif'), mnemonic(w)]))
```

TODO: separators, accelerators, options for sub menus.

7 XJWidgets

- xj2.P
- xjdisplays.P
- xjcdfwidgets
 - ◆ desktop_utils.P
 - ◆ toolbar_widget.P
 - ◆ cdfDisplayUtils.P
 - ◆ abbreviationPanel.P
 - ◆ rel_widget.P
- xjomswidgets
 - ◆ toolbar_widget