

ESP Toolbar Component Tutorial

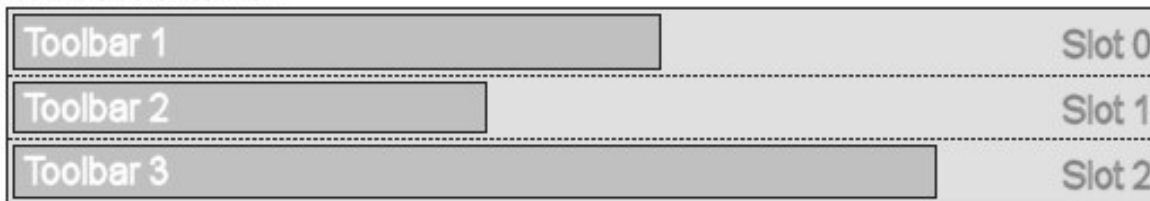
Welcome to the tutorial for the *Exponential Software Developers' Toolbar (ESP Toolbar Component)*. This covers a complete walk through of the functionality and use of the current toolbar version, which at the time of writing is v1.5.19 (May 2006).

The ESP Toolbar is an Internet Explorer 6 style toolbar component, written entirely in Java Swing. As a Java Swing (JFC) component, it can be plugged directly into any Java Swing application or applet that is Java 2 compliant (or better).

Overview

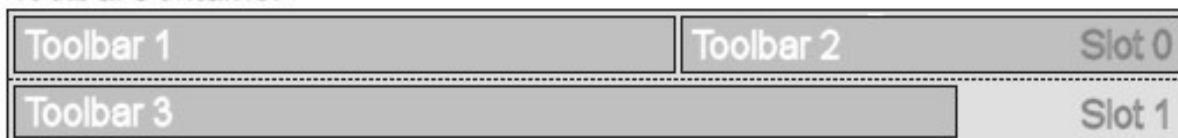
The ESP Toolbar has two major components; the `ToolbarContainer` and the `Toolbar`. The `Toolbar Container` is the main top level component, which can contain one or more `Toolbar` components, placed in horizontal regions known as slots.

ToolbarContainer



Each toolbar component can be created and added to the container, and a new slot will be created for each new toolbar, unless it is added with a specific slot number. It is possible for more than one toolbar to exist in one slot. When the screen containing the ESP Toolbar is displayed at runtime, the user may also grab toolbars with the mouse and shift them around, dragging and dropping them between horizontal slots, or dragging them left and right to reorder toolbars within one slot. When a toolbar is dragged over the top of another horizontally, the toolbars will swap position. Also, if a toolbar is dragged out of a horizontal slot and that slot is left empty, the slot is automatically removed shrinking the toolbar container. This allows users to customize the toolbar configuration to suit their own needs.

ToolbarContainer



If the default width of any individual toolbar exceeds the available display space the toolbar is shrunk and a *drop-down more button* appears on the right-hand side of the toolbar. When the more button is clicked the remaining non-visible toolbar buttons are displayed in the drop-down list for the user to select.

Now lets take a look at a basic ESP Toolbar implementation.

Basic Toolbar Creation

The first step is to create the ESP Toolbar Container and attach it to your Frame or Container. This is done simply as follows:

```
JFrame myAppWindow;  
  
:  
  
ToolbarContainer tbarContainer = new ToolbarContainer();  
tbarContainer.addToFrame(myAppWindow);
```

This is the preferred method, but of course the toolbar container can be added directly to any other container as follows:

```
Container myContainer; // BorderLayout  
  
:  
  
ToolbarContainer tbarContainer = new ToolbarContainer();  
myContainer.add(tbarContainer, BorderLayout.NORTH);
```

If you run your application now, you will see an empty toolbar appear in your frame or panel. The next step is to add a Toolbar component to this container. All you need to do is create a Toolbar instance, and then add buttons to it. Note that the ESP Toolbar supports any Swing JComponent, not just JButtons and other JButton derivatives. Adding a button to a toolbar is done quite easily:

```
Toolbar tbar = new Toolbar();  
tbar.add(new JButton("Button text"));
```

There are many button types that can be added to a toolbar, but to begin with we will simply add JButtons to the toolbar. The next code snippet creates a Toolbar and adds four buttons to it. Then the toolbar is added to the main ToolbarContainer for display.

```
Toolbar basicToolbar = new Toolbar();  
  
JButton btnFirst = new JButton("First Button");  
JButton btnSecond = new JButton("Second Button");  
JButton btnThird = new JButton("Third Button");  
JButton btnFourth = new JButton("Fourth Button");  
  
basicToolbar.add(btnFirst);  
basicToolbar.add(btnSecond);  
basicToolbar.add(btnThird);  
basicToolbar.add(btnFourth);  
  
toolbarContainer.add(basicToolbar);
```

When you compile and run this code, your toolbar should appear with four text buttons as shown below.

<< insert screenshot>>

<<insert link to example code>>

Multiple Toolbars

You can easily create and add multiple Toolbars to a ToolbarContainer that is attached to your frame. Simply create each toolbar, populate it with buttons and then add it to the container. The code snippet below creates three toolbars, and adds text buttons to them. Each toolbar will appear in it's own slot, one beneath the other. When you run this code, you can drag and drop each toolbar around inside the toolbar container. Stretching and shrinking the JFrame will also affect the appearance of your toolbars, and the *more button* will appear if horizontal space runs out on a toolbar.

```
//  
// Create the "browse" toolbar.  
Toolbar browseToolbar = new Toolbar("Browse");  
  
browseToolbar.add( new JButton("Back") );  
browseToolbar.add( new JButton("Forward") );  
browseToolbar.add( new JButton("Stop") );  
browseToolbar.add( new JButton("Refresh") );  
browseToolbar.add( new JButton("Home") );  
  
//  
// Create the "document" toolbar.  
Toolbar docToolbar = new Toolbar("Document");  
  
docToolbar.add( new JButton("New") );  
docToolbar.add( new JButton("Open") );  
docToolbar.add( new JButton("Save") );  
  
//  
// Create the "email" toolbar.  
Toolbar emailToolbar = new Toolbar("Email");  
  
emailToolbar.add( new JButton("New") );  
emailToolbar.add( new JButton("Reply") );  
emailToolbar.add( new JButton("ReplyAll") );  
emailToolbar.add( new JButton("Forward") );  
  
//  
// Add to the container.  
toolbarContainer.add(browseToolbar);  
toolbarContainer.add(docToolbar);  
toolbarContainer.add(emailToolbar);
```

<<insert screenshot>>

<<insert link to example code>>

You can right-click on the ToolbarContainer to pop up a context menu for the toolbars. This context menu displays the name of each toolbar strip inside this container, and allows the user

to show and hide any of the toolbars inside the container. The context menu also has an option to lock down the toolbars, so that they cannot be dragged around. You will notice that the drag bar disappears when this is done. All these options can also be programmatically controlled as you will discover later in this tutorial.

Toolbar Icons

The majority of toolbars in applications use icons (or small images) rather than plain text buttons as we have seen so far. Other popular toolbar configurations (Internet Explorer 6 included) have icons and text on each button.

Lets start with simple icon buttons, with small images each with a dimension of 16x16 pixels. A lot of applications use this approach for their toolbars. Consider you are building a music application. You may need a toolbar that contains Play, Pause, Stop, Fast Forward, Fast Back, Skip Forward and Skip Back. These icons, being 16x16 pixels in size would look like this:

<<show icons required>>

Now right-click on these icons above and save them on your local disk. We can now build a toolbar that contains these icon buttons. To do this we can use the ImageButton class provided with the ESP Toolbar. The code snippet below demonstrates this.

```
// Load the images
Icon iconPlay = new ImageIcon("play.jpg");
Icon iconPause = new ImageIcon("pause.jpg");
Icon iconStop = new ImageIcon("stop.jpg");
Icon iconFFwd = new ImageIcon("ffwd.jpg");
Icon iconFBwd = new ImageIcon("fbwd.jpg");
Icon iconSFwd = new ImageIcon("sfwd.jpg");
Icon iconSBwd = new ImageIcon("sbwd.jpg");

Toolbar musicToolbar = new Toolbar("Music");
musicToolbar.add(new ImageButton(iconPlay), "Play");
musicToolbar.add(new ImageButton(iconPause), "Pause");
musicToolbar.add(new ImageButton(iconStop), "Stop");
musicToolbar.add(new ImageButton(iconFFwd), "Fast Forward");
musicToolbar.add(new ImageButton(iconFBwd), "Fast Backward");
musicToolbar.add(new ImageButton(iconSFwd), "Skip Forward");
musicToolbar.add(new ImageButton(iconSBwd), "Skip Backward");

tbarContainer.add(musicToolbar);
```

When you run this program you will see the toolbar strip appear with the seven icon buttons inside of it. If you hover your mouse over each button the tooltip text will appear for each button describing it's purpose.

<<insert screenshot>>

<<insert link to example code>>

If you are deploying your application from a Java Archive (JAR) file or through Java Web Start etc, you may need to load your images slightly differently. Another way to load images from Jar resources is like this:

```
Icon icon = new ImageIcon(ClassLoader.getResource("my.jpg"));
```

Toolbar Icons and Text

If your toolbar requires both image and text you can use the `ImageTextButton` class provided with the ESP Toolbar. You can specify the position of the text label relative to the icon with constant values of `LEFT`, `BOTTOM`, and `RIGHT`. For example:

```
Toolbar tbar = new Toolbar("Music");
int bottom = ImageTextButton.BOTTOM;

tbar.add(new ImageTextButton(iconPlay, "Play", bottom));
tbar.add(new ImageTextButton(iconPause, "Pause", bottom));
tbar.add(new ImageTextButton(iconStop, "Stop", bottom));
tbar.add(new ImageTextButton(iconFFwd, "Fast Fwd", bottom));
tbar.add(new ImageTextButton(iconFBwd, "Fast Bwd", bottom));
tbar.add(new ImageTextButton(iconSFwd, "Skip Fwd", bottom));
tbar.add(new ImageTextButton(iconSBwd, "Skip Bwd", bottom));

tbarContainer.add(tbar);
```

<<insert screenshot of each text orientation>>

<<insert link to example code>>

Toolbar Rollover Icons

If you want your image buttons to respond to mouse events, and display different icons depending on the button state, you can use the `ImageRolloverButton`. This can be added as above using:

```
tbar.add(new ImageRolloverButton(icon, rolloverIcon, pressedIcon));
```

All of the above buttons rely on the basic Swing `JButton` for the underlying functionality. If you require some specific button style not supplied with this library, use your own customizations of `JButton` instead.

<<insert link to example code>>

Embedded Components

The ESP Toolbar can also handle other embedded components, other than just buttons. In fact any `JComponent` may be added to the toolbar, providing great flexibility. `JLabels`, `JTextFields`, `JCheckBoxes`, `JComboBoxes`, `JSeparators`, `JPanels` and more. As an example, lets create a simple toolbar that mimicks the Internet Explorer address bar. This can be done by creating a `Toolbar`, and adding `JLabel`, `JTextField` and `ImageButton`.

This is also a great point to introduce the concept of stretchiness. Some components added to the toolbar are considered to be stretchy by default (like a JTextField, JComboBox and JMenuBar) and others are not stretchable (like JButtons).

You can control stretchiness by pre-wrapping your component in a ToolbarItem first, before adding it to a Toolbar. The ToolbarItem allows you to specify a value for boolean stretchable. See below.

```
Toolbar musicToolbar = new Toolbar("Music");
int columns = 10;
:
// Make non-stretchable text field
JTextField searchText = new JTextField(columns);
musicToolbar.add(new ToolbarItem(searchText, false));
```

<<insert link to example code>>

Menus and Menu Bars

Java Swing provides JMenuBar and JMenu items to create application menus, directly underneath the window title bar. The ESP Toolbar allows you to pull application menu bars into the ToolbarContainer, and turn them into a Toolbar strip that can be shifted around like any other toolbar strip.

This can be achieved in two ways;

- Add an entire JMenuBar to a Toolbar object.
- Add individual JMenu items to a Toolbar object.

<<insert screenshot>>

<<insert link to example code>>

Locking Toolbars

The ESP Toolbar as a whole can be locked or unlocked. When unlocked (which is the default) users are able to grab toolbar strips from their left edge and drag them around to reposition them inside the toolbar container. If the container is locked, then the left edge does not display the visual drag bar, and the user cannot drag and drop toolbars and reposition them.

In addition, the user can pop up the right-click context menu for the toolbar container and change the state of toolbar locking by selecting the 'Lock Toolbars' option. When this occurs, the visual drag bars will appear and disappear accordingly.

Programmatically, you can;

- Lock or unlock toolbars at the container level: `ToolbarContainer.setLocked()`.
- Lock or unlock individual toolbars: `Toolbar.setLocked()`.
- Allow or disallow users from locking the toolbars (in the right-click context menu): `ToolbarContainer.setAllowUserLock()`.

<<insert screenshot>>
<<insert link to example code>>

Hiding Toolbars

Users can open the right-click context menu to see a list of all toolbars in the container. From here they can show or hide any toolbar by selecting the 'toolbar name' in the popup menu. A tick next to each toolbar name indicates which toolbars are currently visible.

Programatically you can prevent users from hiding toolbars by calling the method `Toolbar.setHideable()` and passing in `false`.

Toolbar showing and hiding can also be programmatically controlled using the methods `ToolbarContainer.showToolbar()` and `toolbarContainer.hideToolbar()`.

Auto Restore Mode

The ESP Toolbar has an 'Auto Restore' feature. This allows the toolbar to save it's state to the user's home directory in properties called a `.toolbar` file. The toolbar container will store the horizontal slot position and location of each toolbar, as well as it's locked state into the properties file.

When the application is loaded again, the toolbar positions are restored from this properties file automatically.

To turn on the Auto Restore mode, you can create the `ToolbarContainer` with the Auto Restore flag like this:

```
boolean autoRestore = true;  
ToolbarContainer tb = new ToolbarContainer(autoRestore);
```

XML Configuration

Developers can write Java Swing code to create a `ToolbarContainer` and individual `Toolbar` components, and then add `JComponents` to them as needed.

A simpler way to create a toolbar configuration is to use XML. The ESP Toolbar supports XML tags that supply toolbar component information to the `ToolbarContainer` during construction. An example `Toolbar` XML definition is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>  
<esp-toolbar>  
  <toolbar name="Demo" id="ID_DEMO" slot="0">  
    <label value="Address:"/>  
    <text id="address" columns="30"/>  
    <button id="navigate" value="Navigate" image="box16.gif" label-position="right"  
      tooltip="press me now..." disabled="false"/>  
    <separator/>  
    <choice id="site">  
      <choice-item>Google</choice-item>  
      <choice-item>Yahoo!</choice-item>  
      <choice-item>Microsoft</choice-item>  
      <choice-item>IBM</choice-item>  
    </choice>  
  </toolbar>  
</esp-toolbar>
```

```

<toolbar name="Icon Buttons" id="ID_ICONS" slot="1">
  <button image="tb1.gif" id="back" value="Back" label-position="right"
    tooltip="Back to the previous page"/>
  <button image="tb2.gif" id="fwd" value="Forward" label-position="right"
    tooltip="Forward to the next page"/>
  <button image="tb3.gif" id="stop" value="Stop" label-position="right"
    tooltip="Stop downloading the page"/>
  <button image="tb4.gif" id="refresh" value="Refresh" label-position="right"
    tooltip="Refresh the page"/>
  <button image="tb5.gif" id="home" value="Home" label-position="right"
    tooltip="Home page"/>
</toolbar>
<toolbar name="Small Icons" id="ID_SMALL" slot="2">
  <button image="books16.gif" id="books" tooltip="Books are cool"/>
  <button image="dog16.gif" id="dog" tooltip="My pet dog looks like this"/>
  <button image="feedback16.gif" id="feedback" tooltip="Send me some feedback"/>
  <button image="home16.gif" id="home" tooltip="My house"/>
  <button image="lightbulb16.gif" id="bulb" tooltip="Ideas page"/>
</toolbar>
</esp-toolbar>

```

Once you have written your XML definition, you can create a full toolbar setup by calling:

```

File xmlFile = new File(...);
ToolbarContainer tb = new ToolbarContainer(xmlFile);

```

You can also use XML content in String format to configure a toolbar. This is useful if you wish to 'generate' a toolbar setup and pass it in:

```

String xml = ...;
ToolbarContainer tb = new ToolbarContainer(xml);

```

Any images that you may be referencing in your XML definition will need to be present on the Java classpath so that they can be loaded successfully by the ClassLoader. This makes toolbar configuration a lot simpler to implement.

XML Component Lists

Once a ToolbarContainer has been configured via XML, you can gain access to all the Toolbar objects and component objects using the XML Component List.

From the ToolbarContainer, you can access Toolbar objects like this:

```

List xmlList = toolbarContainer.getXmlComponentList();
Toolbar demoToolbar = (Toolbar) xmlList.get(0);
Toolbar iconToolbar = (Toolbar) xmlList.get(1);

```

Each Toolbar created also has an XML Component List of it's own. The Toolbar list contains all component objects created for this toolbar. You can access each component in the Toolbar like this:

```

List tbarXmlList = demoToolbar.getXmlComponentList();
JTextField textAddress = (JTextField) tbarXmlList.get(0);
JButton btnNavigate = (JButton) tbarXmlList.get(1);
JComboBox choiceSite = (JComboBox) tbarXmlList.get(2);
:
// do some stuff with each component ...

```

Once you have each component, you can add Action Listeners, Mouse Listeners, etc to handle appropriate events, and respond to them.

Another way to traverse Toolbars and Toolbar components is to use List Iterators. The following code demonstrates the use of Iterators.

```
Iterator it = tc.getXMLComponentList().iterator();
while (it.hasNext())
{
    Toolbar toolbar = (Toolbar) it.next();

    // Traverse all Toolbar components in order
    Iterator tbit = toolbar.getXmlComponentList().iterator();
    while (tbit.hasNext())
    {
        JComponent component = (JComponent) tbit.next();

        // Do stuff with each component...
    }
}
```

XML Definition

Any toolbar XML file or text string must be well formed, and use the correct elements and attributes as defined in this section.

ELEMENT: esp-toolbar

COMMENT: This is the top level toolbar container element. The XML file should only include one of these elements per configuration.

ATTRIBUTES:

width	[optional] the preferred width of the Toolbar Container to set (type: int).
height	[optional] the preferred height of the Toolbar Container to set (type: int).
auto-restore	[optional] turns on/off the auto-restore mode (type: boolean).
allow-user-lock	[optional] turns on/off the user's ability to lock the toolbar from the context menu (type: boolean).

CHILDREN: Child nodes of <esp-toolbar> can be any number of <toolbar> sub-elements.

ELEMENT: toolbar

COMMENT: This element defines a single draggable toolbar strip, and the components contained within it are defined in sub-elements. You can provide any number of toolbar elements inside an <esp-toolbar> parent element. If your configuration only needs one Toolbar, you may use a single <toolbar> element at the root level.

ATTRIBUTES:

name	[optional] the name of this Toolbar, which will appear in the right-click context menu (type: String).
------	--

id	[optional] the id of this Toolbar, which is used to identify this Toolbar, and is used during Auto-Restore (type: String).
slot	[optional] the slot number to add this Toolbar to, where the first horizontal slot is zero (type: int).
x	[optional] the preferred X position of this Toolbar within the slot, used for ordering multiple Toolbars added to one slot (type: int).
hideable	[optional] sets whether this Toolbar can be hidden by the user, in the right-click context menu (type: boolean).
locked	[optional] locks this Toolbar down so that it cannot be dragged around by the user (type: boolean).

CHILDREN: Child nodes of <toolbar> can be any of <label>, <choice>, <button>, <separator> and <text> sub-elements.

ELEMENT: button

COMMENT: This element defines a button inside a Toolbar. You can provide any number of button elements inside a <toolbar> parent element.

ATTRIBUTES:

value	[optional] the text value to display in the button, if this button requires text (type: String).
id	[optional] the id of this button, which is used to identify this component (type: String).
tooltip	[optional] the tool tip text to display for this button when the user hovers their mouse over the button (type: String).
disabled	[optional] set to true if this button should be disabled by default (type: boolean).
image	[optional] the image file name (and path if required) for the icon to display in the button (type: String).
rollover-image	[optional] the image file name (and path if required) for the rollover image to display when the mouse is over the button (type: String).
pressed-image	[optional] the image file name (and path if required) for the pressed image to display when the mouse is pressing the button (type: String).
label-position	[optional] the relative position of the label text, from the button image. Valid values are "left", "right" and "bottom" (type: String).

CHILDREN: Child nodes are not required for <button>.

ELEMENT: choice

COMMENT: This element defines a choice (drop down box) inside a Toolbar. You can provide any number of choice elements inside a <toolbar> parent element.

ATTRIBUTES:

id	[optional] the id of this choice component, which is used to identify it (type: String).
disabled	[optional] set to true if this button should be disabled by default (type: boolean).

CHILDREN: Child nodes of <choice> can be any number of <choice-item> sub-elements. Each <choice-item> becomes an entry in the drop down list.

ELEMENT: choice-item

COMMENT: This element defines a choice item inside a choice (drop down box). You can provide any number of choice-item elements inside a <choice> parent element. The <choice-item> tag and end tag </choice-item> encloses text to be added to the choice component. For example:

```
<choice-item>Item One</choice-item>
```

ATTRIBUTES: none.

CHILDREN: Child nodes are not required for <choice-item>.

ELEMENT: label

COMMENT: This element defines a text label inside a Toolbar. You can provide any number of label elements inside a <toolbar> parent element.

ATTRIBUTES:

value	[mandatory] the text to display in the label (type: String).
id	[optional] the id of this label, which is used to identify this component (type: String).
align	[optional] the horizontal text alignment to use inside the label, which is only necessary if you are stretching the label using the width attribute. One of "left", "right" or "center" (type: String).
width	[optional] changes the default width of the label to this value in pixels (type: int).

CHILDREN: Child nodes are not required for <label>.

ELEMENT: separator

COMMENT: This element defines a vertical visual separator inside a Toolbar. You can provide any number of separator elements inside a <toolbar> parent element.

ATTRIBUTES: none.

CHILDREN: Child nodes are not required for <separator>.

ELEMENT: text

COMMENT: This element defines a text edit box inside a Toolbar. You can provide any number of text elements inside a <toolbar> parent element.

ATTRIBUTES:

value	[optional] the text value to display inside the text field (type: String).
id	[optional] the id of this text box, which is used to identify this component (type: String).
columns	[optional] the number of character columns to use when sizing this component. Alternatively you can use the width attribute (type: int).
width	[optional] the width of this component to set in pixels (type: int).
disabled	[optional] true if this component will be disabled by default (type: boolean).

CHILDREN: Child nodes are not required for <text>.