JDirectoryChooser Documentation

How to Use JDirectoryChooser

JDirectoryChooser provides a GUI for navigating the file system, and then either choosing an existing directory from a tree or managing file system directories' tree structure by creating new directories, renaming, copying, moving or deleting existing directories.

To display a directory chooser, you can either add an instance of JDirectoryChooser to a container, or use the JDirectoryChooser API to show a modal dialog that contains a directory chooser.

A JDirectoryChooser object only presents the GUI for choosing and managing directories. Your program is responsible for doing something with the chosen directory, such as using it as default location for output or doing something with the content of the chosen directory... etc.

Bringing up a standard directory chooser dialog requires only one line of code:

```
// The following code pops up a directory chooser dialog
// pointing to the user's home directory.

File dir = JDirectoryChooser.showDialog(aParent));
```

The argument to the `showDialog` method specifies the parent component for the dialog. The parent component affects the position of the dialog and the frame that the dialog depends on. For example, the Java Look & Feel places the dialog directly over the parent component. If the parent component is in a frame, then the dialog is dependent on that frame, disappearing when the frame is iconified and reappearing when the frame is deiconified.

By default, a directory chooser dialog points to the user's home directory. You can specify the initial directory either by passing such parameter to the **showDialog** method:

```
// The following code pops up a directory chooser dialog
// pointing to the Java home directory.

File initialDir = new File(System.getProperty("java.home"));
File dir = JDirectoryChooser.showDialog(aParent,initialDir));
```

or by setting the **PROP_INITIAL_DIRECTORY** option:

```
// The following code pops up a directory chooser dialog
// pointing to the Java home directory.

DirectoryChooserDefaults.putOption(
        DirectoryChooserDefaults.PROP_INITIAL_DIRECTORY,
        new File(System.getProperty("java.home"))
);
File dir = JDirectoryChooser.showDialog(aParent));
```

The call to showDialog usually appears in the actionPerformed method of a button's or menu item's action listener. The typical example is shown here:

```
/**
 * Event handler.
 *
 * @param e     the action event.
 */
public void actionPerformed(ActionEvent e) {
    File dir = JDirectoryChooser.showDialog(aParent));
    if (dir != null) {
        // This is where a real application would handle user's selection.
        System.out.println("You selected: " + dir.getName());
    } else {
        System.out.println("Operation canceled by user");
    }
}
```

If you want to customize the directory chooser dialog, or develop your own custom dialogs using JDirectoryChooser component keep reading. We'll discuss the following topics:

Customizing the Directory Chooser Dialog

In order to assist you in development of swing applications with custom graphics and multi-language support we designed directory chooser fully customizable. You can change all text resources of the directory chooser dialog either by passing necessary parameters to the showDialog method:

```
// The following code pops up a directory chooser dialog
// with custom title and information text.

File dir = JDirectoryChooser.showDialog(
    aParent,
    initialDir,
    "Select folder",                                    // Custom title
    "Please choose the folder for TMP files."           // Custom text
);
```

or by setting the default options:

```
 // The following code pops up a directory chooser dialog
// with custom title, information, OK and Cancel buttons' text.

// Set custom dialog title.
DirectoryChooserDefaults.putOption(
        DirectoryChooserDefaults.PROP_DIALOG_CAPTION_TEXT,
        "Select folder"
);

// Set custom information text.
DirectoryChooserDefaults.putOption(
        DirectoryChooserDefaults.PROP_DIALOG_TEXT,
        "Please choose the folder for TMP files."
);

// Set custom OK button's text.
DirectoryChooserDefaults.putOption(
        DirectoryChooserDefaults.PROP_OK_TEXT,
        "Yes"
);

// Set custom Cancel button's text.
DirectoryChooserDefaults.putOption(
        DirectoryChooserDefaults.PROP_CANCEL_TEXT,
        "No"
);

// Pop up a dialog.
File dir = JDirectoryChooser.showDialog(
    aParent,
    initialDir
);
```

We recommend to set custom dialog title and information text by passing these parameters to the **showDialog** method. Text of the OK and Cancel buttons should be managed using **DirectoryChooserDefaults** functionality.

In some situations you may want to make particular functions (such as deleting or moving of existing directories) disabled in the directory chooser dialog. This can be done either by passing the necessary access rights as a parameter to the **showDialog** method:

```
// The following code pops up a directory chooser dialog
// with only 'New Directory' and 'Copy' functions enabled.

File dir = JDirectoryChooser.showDialog(
    aParent,
    initialDir,
    "Select folder",
    "Please choose the directory:",
    JDirectoryChooser.ACCESS_NEW | JDirectoryChooser.ACCESS_COPY
);
```

or by setting the default **PROP_ACCESS** option:

```
// The following code pops up a directory chooser dialog
// with only 'New Directory' and 'Copy' functions enabled.

// Set default access rights.
int access = JDirectoryChooser.ACCESS_NEW | JDirectoryChooser.ACCESS_COPY;
DirectoryChooserDefaults.putOption(
        DirectoryChooserDefaults.PROP_ACCESS,
        new Integer(access)
);

// Pop up a dialog.
File dir = JDirectoryChooser.showDialog(
    aParent,
    initialDir
);
```

If you want to use custom icons in the directory chooser dialog you should set the following options:

```
   // Change default icons' theme to custom.
DirectoryChooserDefaults.putOption(
        DirectoryChooserDefaults.PROP_ICONS_THEME,
        new Integer(JDirectoryChooser.ICONS_CUSTOM)
);

// Set custom icon for drives/roots.
DirectoryChooserDefaults.putOption(
        DirectoryChooserDefaults.PROP_ROOT_ICON,
        customRootIcon
);

// Set custom icons for directories.
DirectoryChooserDefaults.putOption(
        DirectoryChooserDefaults.PROP_DIRECTORY_ICON,
        customDirIcon
);
DirectoryChooserDefaults.putOption(
        DirectoryChooserDefaults.PROP_SELECTED_DIRECTORY_ICON,
        customSelectedDirIcon
);
```

Using JDirectoryChooser Component in the Dialogs and Frames

If standard directory chooser dialog does not suite your needs or you want to and directory browsing functionality to your application's frames and dialogs you should add an instance of **JDirectoryChooser** to your custom container. The typical usage of **JDirectoryChooser** component is shown below:

```
 // Create a directory chooser
final JDirectoryChooser dc = new JDirectoryChooser();

// Set custom access rights
dc.setAccess(JDirectoryChooser.ACCESS_NEW);

// Set custom icons theme
dc.setIconsTheme(JDirectoryChooser.ICONS_LOOK_AND_FEEL);

// Add a custom selection listener
dc.addDirectoryChooserSelectionListener(new CustomSelectionListener());

// Expand the first root node
dc.expandFirstRoot();

// Add JDirectoryChooser to the container
add(dc,BorderLayout.CENTER);

...

/**
 * Custom selection listener.
 */
final class CustomSelectionListener implements
    DirectoryChooserSelectionListener {

  /**
   * Called whenever the selection in the attached JDirectoryChooser
   * component changes.
   *
   * @param event   the event object that contains information about
   *                new selection.
   */
  public void valueChanged(DirectoryChooserSelectionEvent event) {
    System.out.println("You selected: " + event.getPath().toString());
  }
}
```