

GWT - CELLBROWSER WIDGET

http://www.tutorialspoint.com/gwt/gwt_cellbrowser_widget.htm

Copyright © tutorialspoint.com

Introduction

The **CellBrowser** widget represents a **browsable** view of a tree in which only a single node per level may be open at one time.

Class declaration

Following is the declaration for **com.google.gwt.user.cellview.client.CellBrowser<T>** class:

```
public class CellBrowser
    extends AbstractCellTree
        implements ProvidesResize, RequiresResize, HasAnimation
```

Class constructors

S.N. Constructor & Description

- CellBrowser***TreeViewModelviewModel, TrootValue*
Construct a new CellBrowser.
- CellBrowser***TreeViewModelviewModel, TrootValue, CellBrowser. Resourcesresources*
Construct a new CellBrowser with the specified CellBrowser.Resources.

Class methods

S.N. Function name & Description

- protected Widget createPager(HasData display)**
Create a pager to control the list view.
- int getDefaultColumnWidth**
Get the default width of new columns.
- int getMinimumColumnWidth**
Get the minimum width of columns.
- TreeNode getRootTreeNode**
Get the root TreeNode.

boolean isAnimationEnabled

Returns true if animations are enabled, false if not.

6

void onBrowserEvent*Eventevent*

Fired whenever a browser event is received.

7

void onResize

This method must be called whenever the implementor's size has been modified.

8

void setAnimationEnabled*booleanenable*

Enable or disable animations.

9

void setDefaultColumnWidth*intwidth*

Set the default width of new columns.

10

void set Keyboard Selection Policy *HasKeyboardSelectionPolicy. KeyboardSelectionPolicypolicy*

Set the HasKeyboardSelectionPolicy.KeyboardSelectionPolicy.

11

void setMinimumColumnWidth*intminWidth*

Set the minimum width of columns.

Methods inherited

This class inherits methods from the following classes:

- com.google.gwt.user.client.ui.UIObject
- com.google.gwt.user.client.ui.Widget
- com.google.gwt.user.client.ui.Composite
- com.google.gwt.user.cellview.client.AbstractCellTree
- java.lang.Object

CellBrowser Widget Example

This example will take you through simple steps to show usage of a CellBrowser Widget in GWT. Follow the following steps to update the GWT application we created in *GWT - Create Application* chapter:

Step	Description
------	-------------

- | | |
|---|---|
| 1 | Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint</i> as explained in the <i>GWT - Create Application</i> chapter. |
| 2 | Modify <i>HelloWorld.gwt.xml</i> , <i>HelloWorld.css</i> , <i>HelloWorld.html</i> and <i>HelloWorld.java</i> as |

explained below. Keep rest of the files unchanged.

- 3 Compile and run the application to verify the result of the implemented logic.

Following is the content of the modified module descriptor **src/com.tutorialspoint/HelloWorld.gwt.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='helloworld'>
  <!-- Inherit the core Web Toolkit stuff. -->
  <inherits name='com.google.gwt.user.User' />

  <!-- Inherit the default GWT style sheet. -->
  <inherits name='com.google.gwt.user.theme.clean.Clean' />

  <!-- Specify the app entry point class. -->
  <entry-point class='com.tutorialspoint.client.HelloWorld' />

  <!-- Specify the paths for translatable code -->
  <source path='client' />
  <source path='shared' />
</module>
```

Following is the content of the modified Style Sheet file **war/HelloWorld.css**.

```
body{
    text-align: center;
    font-family: verdana, sans-serif;
}
h1{
    font-size: 2em;
    font-weight: bold;
    color: #777777;
    margin: 40px 0px 70px;
    text-align: center;
}
```

Following is the content of the modified HTML host file **war/HelloWorld.html**.

```
<html>
<head>
<title>Hello World</title>
  <link rel="stylesheet" href="HelloWorld.css"/>
  <script language="javascript" src="helloworld/helloworld.nocache.js">
  </script>
</head>
<body>

<h1>CellBrowser Widget Demonstration</h1>
<div ></div>

</body>
</html>
```

Let us have following content of Java file **src/com.tutorialspoint/HelloWorld.java** which will demonstrate use of CellBrowser widget.

```
package com.tutorialspoint.client;

import java.util.ArrayList;
import java.util.List;

import com.google.gwt.cell.client.AbstractCell;
import com.google.gwt.cell.client.Cell;
```

```

import com.google.gwt.cell.client.TextCell;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.safehtml.shared.SafeHtmlBuilder;
import com.google.gwt.user.cellview.client.CellBrowser;
import com.google.gwt.user.cellview.client.
HasKeyBoardSelectionPolicy.KeyboardSelectionPolicy;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.view.client.ListDataProvider;
import com.google.gwt.view.client.SingleSelectionModel;
import com.google.gwt.view.client.TreeViewModel;

public class HelloWorld implements EntryPoint {

    /**
     * A list of songs.
     */
    private static class Playlist {
        private final String name;
        private final List<String> songs = new ArrayList<String>();

        public Playlist(String name) {
            this.name = name;
        }

        /**
         * Add a song to the playlist.
         *
         * @param name the name of the song
         */
        public void addSong(String name) {
            songs.add(name);
        }

        public String getName() {
            return name;
        }

        /**
         * Return the list of songs in the playlist.
         */
        public List<String> getSongs() {
            return songs;
        }
    }

    /**
     * A composer of classical music.
     */
    private static class Composer {
        private final String name;
        private final List<Playlist> playlists = new ArrayList<Playlist>();

        public Composer(String name) {
            this.name = name;
        }

        /**
         * Add a playlist to the composer.
         *
         * @param playlist the playlist to add
         */
        public Playlist addPlaylist(Playlist playlist) {
            playlists.add(playlist);
            return playlist;
        }

        public String getName() {
            return name;
        }
    }

```

```

    }

    /**
     * Return the rockin' playlist for this composer.
     */
    public List<Playlist> getPlaylists() {
        return playlists;
    }
}

/**
 * The model that defines the nodes in the tree.
 */
private static class CustomTreeModel implements TreeViewModel {

    private final List<Composer> composers;

    /**
     * This selection model is shared across all leaf nodes.
     * A selection model can also be shared across all nodes
     * in the tree, or each set of child nodes can have its
     * own instance. This gives you flexibility to determine
     * how nodes are selected.
     */
    private final SingleSelectionModel<String> selectionModel
    = new SingleSelectionModel<String>();

    public CustomTreeModel() {
        // Create a database of information.
        composers = new ArrayList<Composer>();

        // Add compositions by Beethoven.
        {
            Composer beethoven = new Composer("Beethoven");
            composers.add(beethoven);

            Playlist concertos = beethoven.addPlaylist(
                new Playlist("Concertos"));
            concertos.addSong("No. 1 - C");
            concertos.addSong("No. 2 - B-Flat Major");
            concertos.addSong("No. 3 - C Minor");
            concertos.addSong("No. 4 - G Major");
            concertos.addSong("No. 5 - E-Flat Major");

            Playlist quartets = beethoven.addPlaylist(
                new Playlist("Quartets"));
            quartets.addSong("Six String Quartets");
            quartets.addSong("Three String Quartets");
            quartets.addSong("Grosse Fugue for String Quartets");

            Playlist sonatas = beethoven.addPlaylist(
                new Playlist("Sonatas"));
            sonatas.addSong("Sonata in A Minor");
            sonatas.addSong("Sonata in F Major");

            Playlist symphonies = beethoven.addPlaylist(
                new Playlist("Symphonies"));
            symphonies.addSong("No. 2 - D Major");
            symphonies.addSong("No. 2 - D Major");
            symphonies.addSong("No. 3 - E-Flat Major");
            symphonies.addSong("No. 4 - B-Flat Major");
            symphonies.addSong("No. 5 - C Minor");
            symphonies.addSong("No. 6 - F Major");
            symphonies.addSong("No. 7 - A Major");
            symphonies.addSong("No. 8 - F Major");
            symphonies.addSong("No. 9 - D Minor");
        }

        // Add compositions by Brahms.

```

```

{
    Composer brahms = new Composer("Brahms");
    composers.add(brahms);
    Playlist concertos = brahms.addPlaylist(
        new Playlist("Concertos"));
    concertos.addSong("Violin Concerto");
    concertos.addSong("Double Concerto - A Minor");
    concertos.addSong("Piano Concerto No. 1 - D Minor");
    concertos.addSong("Piano Concerto No. 2 - B-Flat Major");

    Playlist quartets = brahms.addPlaylist(
        new Playlist("Quartets"));
    quartets.addSong("Piano Quartet No. 1 - G Minor");
    quartets.addSong("Piano Quartet No. 2 - A Major");
    quartets.addSong("Piano Quartet No. 3 - C Minor");
    quartets.addSong("String Quartet No. 3 - B-Flat Minor");

    Playlist sonatas = brahms.addPlaylist(
        new Playlist("Sonatas"));
    sonatas.addSong("Two Sonatas for Clarinet - F Minor");
    sonatas.addSong("Two Sonatas for Clarinet - E-Flat Major");

    Playlist symphonies = brahms.addPlaylist(
        new Playlist("Symphonies"));
    symphonies.addSong("No. 1 - C Minor");
    symphonies.addSong("No. 2 - D Minor");
    symphonies.addSong("No. 3 - F Major");
    symphonies.addSong("No. 4 - E Minor");
}

// Add compositions by Mozart.
{
    Composer mozart = new Composer("Mozart");
    composers.add(mozart);
    Playlist concertos = mozart.addPlaylist(
        new Playlist("Concertos"));
    concertos.addSong("Piano Concerto No. 12");
    concertos.addSong("Piano Concerto No. 17");
    concertos.addSong("Clarinet Concerto");
    concertos.addSong("Violin Concerto No. 5");
    concertos.addSong("Violin Concerto No. 4");
}
}

/**
 * Get the {@link NodeInfo} that provides the children of the specified
 * value.
 */
public <T> NodeInfo<?> getNodeInfo(T value) {
    if (value == null) {
        // LEVEL 0.
        // We passed null as the root value. Return the composers.

        // Create a data provider that contains the list of composers.
        ListDataProvider<Composer> dataProvider
            = new ListDataProvider<Composer>(composers);

        // Create a cell to display a composer.
        Cell<Composer> cell = new AbstractCell<Composer>(){
            @Override
            public void render(Composer value, Object key,
                SafeHtmlBuilder sb) {
                sb.appendEscaped(value.getName());
            }
        };

        // Return a node info that pairs the data provider and the cell.
        return new DefaultNodeInfo<Composer>(dataProvider, cell);
    } else if (value instanceof Composer) {

```

```

        // LEVEL 1.
        // We want the children of the composer. Return the playlists.
        ListDataProvider<Playlist> dataProvider
        = new ListDataProvider<Playlist>(
        ((Composer) value).getPlaylists());
        Cell<Playlist> cell = new AbstractCell<Playlist>() {
            @Override
            public void render(Playlist value, Object key,
            SafeHtmlBuilder sb) {
                if (value != null) {
                    sb.appendEscaped(value.getName());
                }
            }
        };
        return new DefaultNodeInfo<Playlist>(dataProvider, cell);
    } else if (value instanceof Playlist) {
        // LEVEL 2 - LEAF.
        // We want the children of the playlist. Return the songs.
        ListDataProvider<String> dataProvider
        = new ListDataProvider<String>(
        ((Playlist) value).getSongs());

        // Use the shared selection model.
        return new DefaultNodeInfo<String>(dataProvider, new TextCell(),
        selectionModel, null);
    }
    return null;
}

/**
 * Check if the specified value represents a leaf node.
 * Leaf nodes cannot be opened.
 */
public boolean isLeaf(Object value) {
    // The leaf nodes are the songs, which are Strings.
    if (value instanceof String) {
        return true;
    }
    return false;
}

public void onModuleLoad() {
    // Create a model for the browser.
    TreeViewModel model = new CustomTreeModel();

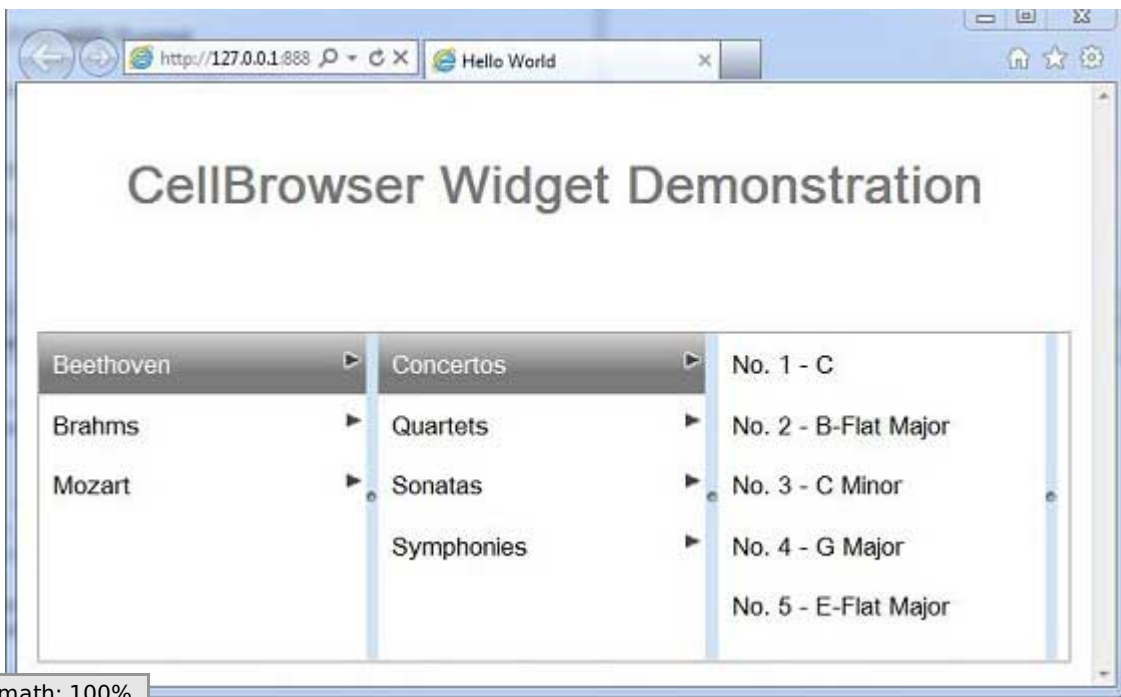
    /**
     * Create the browser using the model. We use <code>null</code> as the
     * default value of the root node. The default value will be passed to
     * CustomTreeModel#getNodeInfo();
     */
    CellBrowser browser = new CellBrowser(model, null);
    browser.setKeyboardSelectionPolicy(KeyboardSelectionPolicy.ENABLED);
    browser.setHeight("200");
    browser.setWidth("630");

    VerticalPanel panel = new VerticalPanel();
    panel.setBorderWidth(1);
    panel.add(browser);

    // Add the widgets to the root panel.
    RootPanel.get().add(panel);
}
}

```

Once you are ready with all the changes done, let us compile and run the application in development mode as we did in [GWT - Create Application](#) chapter. If everything is fine with your application, this will produce following result:



Processing math: 100%