FREE eBook

LEARNING

swt

Free unaffiliated eBook created from
Stack Overflow contributors.

#swt

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: swt

It is an unofficial and free swt ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official swt.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with swt

Remarks

This section provides an overview of what swt is, and why a developer might want to use it.

It should also mention any large subjects within swt, and link out to the related topics. Since the Documentation for swt is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
4.7	2017-06-12
4.6.3	2017-03-01
4.6.2	2016-11-24
4.6.1	2016-09-07
4.6	2016-06-06
4.5.2	2016-02-12
4.5.1	2015-09-04
4.5	2015-06-03
4.4.2	2015-02-04
4.4.1	2014-09-25
4.4	2014-06-06
4.3.2	2014-02-21
4.3.1	2013-09-11
4.3	2013-06-05
4.2.2	2013-02-04
4.2.1	2012-09-14
4.2	2012-06-08
4.1.2	2012-02-23

Version	Release Date
4.1.1	2011-09-12
4.1	2011-06-20
4	2010-07-27
3.8.2	2013-01-31
3.8.1	2012-09-14
3.8	2012-06-08
3.7.2	2012-02-08
3.7.1	2011-09-09
3.7	2011-06-13
3.6.2	2011-02-10
3.6.1	2010-09-09
3.6	2010-06-08
3.5.2	2010-02-11
3.5.1	2009-09-17
3.5	2009-06-11
3.4.2	2009-02-11
3.4.1	2008-09-11
3.4	2008-06-17
3.3.2	2008-02-21
3.3.1.1	2007-10-23
3.3.1	2007-09-21
3.3	2007-06-25
3.2.2	2007-02-12
3.2.1	2006-09-21
3.2	2006-06-29

Version	Release Date
3.1.2	2006-01-18
3.1.1	2005-09-29
3.1	2005-06-27
3.0.2	2005-03-11
3.0.1	2004-09-16
3	2004-06-25
2.1.3	2004-03-10
2.1.2	2003-11-03
2.1.1	2003-06-27
2.1	2003-03-27
2.0.2	2002-11-07
2.0.1	2002-08-29
2	2002-06-27
1	2001-11-07

Examples

Installation or Setup

Detailed instructions on getting swt set up or installed.

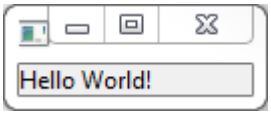
Creating a new SWT progam

Create a new text file named `HelloWorld.java` and paste this code in it:

```
import org.eclipse.swt.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

public class HelloWorld
{
    public static void main(String[] args)
    {
        final Display display = new Display();
        final Shell shell = new Shell(display);
        shell.setLayout(new FillLayout());
```

```
        Label label = new Label(shell, SWT.NONE);
        label.setText("Hello World!");

        shell.pack();
        shell.open();

        while (!shell.isDisposed())
        {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

When you start the program it will look something like this:



A closer look at the Hello World application

The Hello World application consists of a `HelloWorld` class definition and a `main` method.

The main method defines a `Display` and a `Shell`. The display acts as the interface between SWT and the underlying operating system. It handles the platform event model in the form of the SWT event loop. The shell represents a single window of the desktop or window manager.

Widgets are added to the shell by specifying the shell in the constructor of the widget. In this example we create a `Label`. A label is a widget that can display text or an image. In this case we set the text "Hello World!" to it. The widget is added to the shell by specifying our shell as the first argument in the constructor.

To make the label visible in the shell we either have to set a fixed size to it or we need to tell its parent (the shell) how to layout its children.

The `FillLayout` is the simplest SWT `Layout`. It organizes all its children in a single row or column and forces them to have the same size.

The following lines tell the shell to apply its layout and become visible:

```
shell.pack();
shell.open();
```

Last but most importantly, we need to define the event loop of the SWT program. The event loop is needed to transfer the user input events from the underlying operating system widgets to the SWT event system.

```
while (!shell.isDisposed())
{
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
```

This loop will run until the shell is disposed. Once this happens, the display is disposed as well and the program will terminate. While the program is looping, it will read the next operating system event and transfer it to SWT. If there is no event, the thread will sleep until the next event arrives.

Read Getting started with swt online: https://riptutorial.com/swt/topic/6753/getting-started-with-swt

Chapter 2: ExpandBar

Introduction

The `ExpandBar` is a widget which displays a series of horizontal expand bar items. All child controls should be instances of `ExpandItem`.

Syntax

- ExpandBar(Composite parent, int style)

Parameters

Parameter	Details
parent	The parent `Composite` on which the `ExpandBar` will be created
style	The style bits

Remarks

Per the Eclipse documentation, the `ExpandBar` class is not intended to be subclassed.

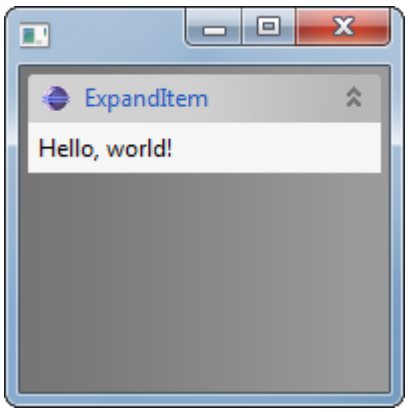
Examples

Simple Example

```
public class ExpandBarExample {

    private final Display display;
    private final Shell shell;

    public ExpandBarExample() {
        display = new Display();
        shell = new Shell(display);
        shell.setLayout(new FillLayout());

        // Create the ExpandBar on the Shell
        final ExpandBar expandBar = new ExpandBar(shell, SWT.NONE);
        expandBar.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));

        // Add a single ExpandItem to the ExpandBar
        final ExpandItem expandItem = new ExpandItem(expandBar, SWT.NONE);
        expandItem.setText("ExpandItem");
        expandItem.setImage(new Image(display, "src/main/resources/sample.gif"));

        // Create a Composite which will hold all of the content in the ExpandItem
```

```
        final Composite expandContent = new Composite(expandBar, SWT.NONE);
        expandContent.setLayout(new GridLayout());

        final Label label = new Label(expandContent, SWT.NONE);
        label.setText("Hello, world!");

        // Set the Composite as the control for the ExpandItem
        expandItem.setControl(expandContent);
        // Set the height of the ExpandItem to be the computed size of its content
        expandItem.setHeight(expandContent.computeSize(SWT.DEFAULT, SWT.DEFAULT).y);
        expandItem.setExpanded(true);
    }

    public void run() {
        shell.setSize(200, 200);
        shell.open();

        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
        display.dispose();
    }

    public static void main(String... args) {
        new ExpandBarExample().run();
    }

}
```

Results in:

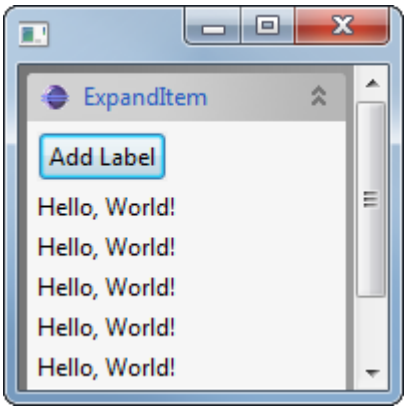


Dynamic ExpandItem Content

```
public class ExpandBarDynamicContentExample {

    private final Display display;
    private final Shell shell;

    public ExpandBarDynamicContentExample() {
        display = new Display();
        shell = new Shell(display);
        shell.setLayout(new FillLayout());

        // Create the ExpandBar on the Shell
```

```
        final ExpandBar expandBar = new ExpandBar(shell, SWT.V_SCROLL);
        expandBar.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));

        // Add a single ExpandItem to the ExpandBar
        final ExpandItem expandItem = new ExpandItem(expandBar, SWT.NONE);
        expandItem.setText("ExpandItem");
        expandItem.setImage(new Image(display, "src/main/resources/sample.gif"));

        // Create a Composite which will hold all of the content in the ExpandItem
        final Composite expandContent = new Composite(expandBar, SWT.NONE);
        expandContent.setLayout(new GridLayout());

        // Add a button to add some dynamic content
        final Button button = new Button(expandContent, SWT.PUSH);
        button.setText("Add Label");
        button.addSelectionListener(new SelectionAdapter() {
            @Override
            public void widgetSelected(SelectionEvent e) {
                // Add another Label to the ExpandItem content
                new Label(expandContent, SWT.NONE).setText("Hello, World!");
                // Re-compute the size of the content
                expandItem.setHeight(expandContent.computeSize(SWT.DEFAULT, SWT.DEFAULT).y);
            }
        });

        // Set the Composite as the control for the ExpandItem
        expandItem.setControl(expandContent);
        // Set the height of the ExpandItem to be the computed size of its content
        expandItem.setHeight(expandContent.computeSize(SWT.DEFAULT, SWT.DEFAULT).y);
        expandItem.setExpanded(true);
    }

    public void run() {
        shell.setSize(200, 200);
        shell.open();

        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
        display.dispose();
    }

    public static void main(String... args) {
        new ExpandBarDynamicContentExample().run();
    }

}
```

Results in:



The key difference here is that we need to re-compute the size of the `ExpandItem` after the content changes. After adding a new `Label` within the `SelectionAdapter`:

```
expandItem.setHeight(expandContent.computeSize(SWT.DEFAULT, SWT.DEFAULT).y);
```

Also notice the use of the `SWT.V_SCROLL` style bit in the `ExpandBar` constructor. This is certainly optional, however it is added to ensure that all content is accessible as the number of `Label` objects grows.

Read ExpandBar online: https://riptutorial.com/swt/topic/9609/expandbar

Chapter 3: GridLayout

Introduction

When an instance of `GridLayout` is set on a `Composite` (or a subclass of `Composite`), all child controls will be arranged in a grid pattern. When there are multiple columns, the grid is populated from left to right, and from top to bottom.

In addition to specifying the number of columns, you can optionally adjust the margins around the grid, as well as the spacing between cells in the grid via various member variables.

Syntax

- GridLayout()
- GridLayout(int numColumns, boolean makeColumnsEqualWidth)

Parameters

Parameter	Details
numColumns	The number of columns in the grid
makeColumnsEqualWidth	Whether or not the columns in the layout should be the same width

Remarks

When using a `GridLayout` on a `Composite` (or a subclass of `Composite`), child controls should set their layout data to a **unique** instance of `GridData` to dictate how the child should be displayed within the parent:

```
// ...
final Shell shell = new Shell(display);
shell.setLayout(new GridLayout());

final Composite child = new Composite(shell, SWT.NONE);
child.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
// ...
```

Failure to set a `GridData` object on child controls may result in the child not being positioned as desired. Furthermore, if the child is a `Composite` (or a subclass of `Composite`), the child and its children may not be visible at all.

If the wrong layout data is used (eg. `FormData` instead of `GridData`), the result will be a `ClassCastException` at runtime:

Exception in thread "main" java.lang.ClassCastException:
org.eclipse.swt.layout.FormData cannot be cast to org.eclipse.swt.layout.GridData

Examples

Single-Column

In this example, we use the default, no-args `GridLayout()` constructor to create a layout with a single column.

```
public class SingleColumnGridLayoutExample {

    private final Display display;
    private final Shell shell;

    public SingleColumnGridLayoutExample() {
        display = new Display();
        shell = new Shell(display);

        // Create the layout and apply to the Shell
        shell.setLayout(new GridLayout());

        // Add the child controls to the Shell - in this case, in a single column
        final Button buttonA = new Button(shell, SWT.PUSH);
        buttonA.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
        buttonA.setText("Button A");

        final Button buttonB = new Button(shell, SWT.PUSH);
        buttonB.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
        buttonB.setText("Button B");

        final Button buttonC = new Button(shell, SWT.PUSH);
        buttonC.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
        buttonC.setText("Button C");

        final Button buttonD = new Button(shell, SWT.PUSH);
        buttonD.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
        buttonD.setText("Button D");
    }

    public void run() {
        shell.pack();
        shell.open();

        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
        display.dispose();
    }

    public static void main(final String... args) {
        new SingleColumnGridLayoutExample().run();
    }

}
```

Results in:

Multi-Column

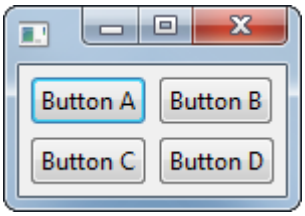
Similar to the Single-Column Example above, if we instead use the `GridLayout(int, boolean)` constructor, we can create a layout with multiple columns.

In this case we create two columns, each of which are the same width.

```
public class MultiColumnGridLayoutExample {

    private final Display display;
    private final Shell shell;

    public MultiColumnGridLayoutExample() {
        display = new Display();
        shell = new Shell(display);

        // Create the layout and apply to the Shell
        shell.setLayout(new GridLayout(2, true));

        // Add the child controls to the Shell - in this case, in two columns
        final Button buttonA = new Button(shell, SWT.PUSH);
        buttonA.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
        buttonA.setText("Button A");

        final Button buttonB = new Button(shell, SWT.PUSH);
        buttonB.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
        buttonB.setText("Button B");

        final Button buttonC = new Button(shell, SWT.PUSH);
        buttonC.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
        buttonC.setText("Button C");

        final Button buttonD = new Button(shell, SWT.PUSH);
        buttonD.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
        buttonD.setText("Button D");
    }

    public void run() {
        shell.pack();
        shell.open();

        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
```

```
        display.dispose();
    }

    public static void main(final String... args) {
        new MultiColumnGridLayoutExample().run();
    }

}
```

Results in:



Non-Default Spacing and Margins

By leveraging some of the member variables in the `GridLayout` instance, we can change the margins around the layout, and spacing between cells. In this example we set the following:

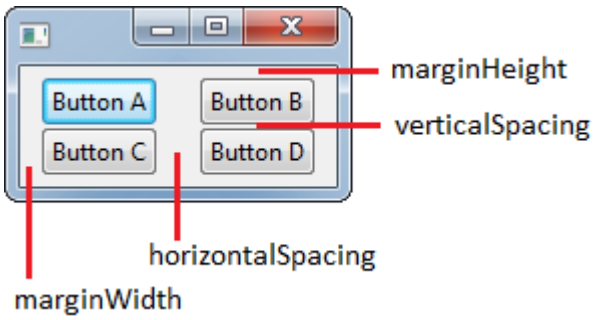
1. `verticalSpacing = 0` - Sets the vertical spacing between cells to `0px`.
2. `horizontalSpacing = 20` - Sets the horizontal spacing between cells to `20px`.
3. `marginWidth = 10` - Sets the left and right margins of the layout to `10px`.

Note: We do not modify the `marginHeight`, so it stays at the default `5px`.

```
public class GridLayoutExample {

    private final Display display;
    private final Shell shell;

    public GridLayoutExample() {
        display = new Display();
        shell = new Shell(display);

        // Create a layout with two columns of equal width
        final GridLayout shellLayout = new GridLayout(2, true);
        shellLayout.verticalSpacing = 0; // Vertical spacing between cells
        shellLayout.horizontalSpacing = 20; // Horizontal spacing between cells
        shellLayout.marginWidth = 10; // Horizontal margin around the layout
        shell.setLayout(shellLayout);

        final Button buttonA = new Button(shell, SWT.PUSH);
        buttonA.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
        buttonA.setText("Button A");

        final Button buttonB = new Button(shell, SWT.PUSH);
        buttonB.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
        buttonB.setText("Button B");

        final Button buttonC = new Button(shell, SWT.PUSH);
        buttonC.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
        buttonC.setText("Button C");
```

```
        final Button buttonD = new Button(shell, SWT.PUSH);
        buttonD.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));
        buttonD.setText("Button D");
    }

    public void run() {
        shell.pack();
        shell.open();

        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
        display.dispose();
    }

    public static void main(final String... args) {
        new GridLayoutExample().run();
    }

}
```

Results in:



Read GridLayout online: https://riptutorial.com/swt/topic/9445/gridlayout

Credits

S. No	Chapters	Contributors
1	Getting started with swt	Baz, Community
2	ExpandBar	avojak
3	GridLayout	avojak