

# An Eclipse Nebula widgets primer

A quick guide to Eclipse Nebula's Grid, CDateTime, CompositeTable, PGroup and PShelf widgets

Skill Level: Intermediate

[Scott Delap \(scott@clientjava.com\)](mailto:scott@clientjava.com)

President

Rich Client Solutions Inc.

[Barry Livingston \(iblivin@gmail.com\)](mailto:iblivin@gmail.com)

Senior Software Engineer

MichaelDKelly.com

17 Apr 2007

The SWT toolkit offers a robust interface to the native widgets of the operating system it's running on. However, native widgets often aren't enough. The Eclipse Nebula project is working to bridge this gap with custom widgets for functionality, including calendaring and advanced tables. This tutorial demonstrates five Nebula widgets, including Grid, CDateTime, CompositeTable, PGroup, and PShelf.

## Section 1. Before you start

### About this tutorial

The Standard Widget Toolkit (SWT) provides access to the native widgets of an operating system via Java™ technology. However, the widgets available don't solve every need. The Eclipse Nebula project provides nine widgets to help answer common user interface (UI) programming needs. This tutorial demonstrates five of Nebula's nine widgets, including Grid, CDateTime, CompositeTable, PGroup, and PShelf.

SWT has long suffered from a lack of custom widgets that go beyond the standard ones provided. The Eclipse Nebula project was created as a gathering place for widget authors who wish to release their widgets under the Eclipse Public License and have them incubated in an official Eclipse project. It features some widgets, such as the CDateTime widget, that draw their own interfaces. It also includes widgets, such as CompositeTable, that let you use existing SWT widgets in new ways. In all, the Nebula widgets address areas that in the past have been major holes in the feature set of widgets available in the SWT API.

## Prerequisites

This tutorial was written for developers familiar with SWT, JFace, and the Eclipse Rich Client Platform (RCP).

## System requirements

To run the examples, you need a computer capable of running Eclipse V3.2 adequately and 50 MB of free disk space.

---

## Section 2. Heavyweight and lightweight widgets

Before diving into the widgets provided by Nebula, this tutorial provides a brief discussion of widgets in general. There are two popular widget toolkits for Java UI development: Swing and SWT. Swing is called a *lightweight toolkit* because it uses Java code not only to construct the functionality of a widget but also to paint its appearance using Java 2D. SWT at its core is what is commonly called a *heavyweight widget toolkit*. By using the Java Native Interface (JNI), SWT provides a Java interface to native widgets, such as GTK+ or Win32 objects. In cases where native platform widgets don't provide support for the UI component required, SWT implements its own GUI code in the Java programming language. As a result, SWT includes the performance of native widgets, and the look and feel of a toolkit, such as the Abstract Windows Toolkit (AWT), the precursor to Swing), with custom Java-driven widgets similar to those of Swing.

If you need a widget that isn't available in SWT, you have a few options. You can combine the existing widgets to create a compound widget; or you can create your own functionality from the ground up, similar to CTab. The Nebula widgets show examples of both approaches.

This tutorial covers the widgets in the Eclipse Nebula project that provide functionality not included standard in SWT, such as a date/time picker and advanced composite tables. If you are unfamiliar with the Swing and SWT, see [Resources](#).

---

## Section 3. Date and time widgets for SWT

As stated, SWT is primarily a native widget toolkit. Unlike buttons, tables, and labels, there is no common paradigm for selecting date and time information across operating systems. As a result, SWT has historically been lacking a default date/time selection component. A number of solutions have been created to address this. Below are several. See [Resources](#) for more information.

### **SWTCalendar**

An open source Massachusetts Institute of Technology (MIT) license-based widget that allows date selections

### **JPopupCalendar**

An open source Eclipse Public Licensed widget that allows the selection of dates

### **jaret datechooser**

An open source common public-licensed date-selection widget

None of these widgets supports the selection of time in addition to date. Fortunately, the Nebula project contains a new `CDateTime` widget that supports both date and time selection in a number of styles.

---

## Section 4. CDateTime setup

Follow these steps to get started with the `CDateTime` widget:

1. Download [CDateTime](https://eclipse.org/nebula/) from [Eclipse.org/nebula/](https://eclipse.org/nebula/).
2. The content is packaged as a JAR, so use the `jar` command to expand it to a directory.
3. Select **File > Import**, and in the resulting window, under the **General**

category, select **Existing Projects into Workspace**.

4. Browse and select the directory into which you just expanded the CDateTime distribution.

The project you just imported has an incorrectly specified src directory. To correct this, right-click the org.eclipse.swt.nebula.widgets.cdatettime project and select the **Java Build Path** item on the left. Remove the /src folder and add the org.eclipse.swt.nebula.widgets.cdatettime root folder. With the src folder issue corrected, import the needed SWT dependencies:

1. Select **File > Import**.
2. Choose the **Plug-in Development** category and **Plug-ins and Fragments**.
3. Click **Next** and then **Next** again in the subsequent window.
4. Select your platform SWT plug-ins, and add them to the list on the right.
5. Click **Finish** to import the plug-ins.

Now you need to add the plug-ins to the org.eclipse.swt.nebula.widgets.cdatettime project for it to compile:

1. Right-click the project and again select **Java Build Path**.
2. Select the **Project** tab and add the SWT projects.

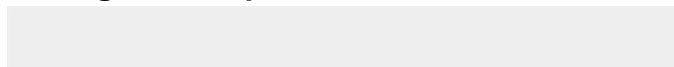
At this point, your cdatettime project should compile and be ready for use.

---

## Section 5. Use CDateTime

Create a new Java project using the File menu. Add the SWT and CDateTime projects as dependencies, similar to the way you modified CDateTime in the previous section. Next, create a package and a class named Example1. Paste in the implementation shown below.

### Listing 1. Example1



```

public class Example1 {
    /**
     * @param args
     */
    public static void main(String[]
args) {
        final Display display = new
Display();
        final Shell shell = new
Shell(display);
        shell.setText("Basic
CDateTime");
        shell.setLayout(new
GridLayout());

        GridLayout layout = new
GridLayout();
        shell.setLayout(layout);

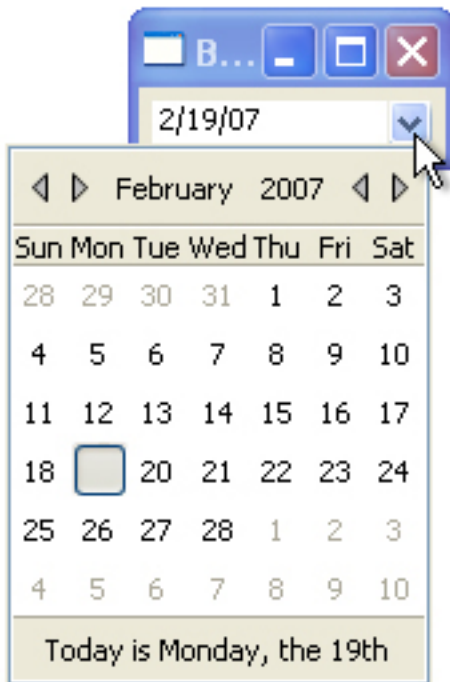
        final CDateTime cdt = new \
CDateTime(shell, CDT.BORDER
| CDT.DROP_DOWN);
        cdt.setLayoutData(new
GridData(SWT.FILL, \
SWT.FILL, true, true));

        shell.pack();
        Point size =
shell.getSize();
        Rectangle screen = \
display.getMonitors()[0].getBounds();
        shell.setBounds(
(screen.width-size.x)/2,
(screen.height-size.y)/2,
                                size.x,
                                size.y
        );
        shell.open();
        while (!shell.isDisposed())
        {
            if
(!display.readAndDispatch())
display.sleep();
        }
        display.dispose();
    }
}

```

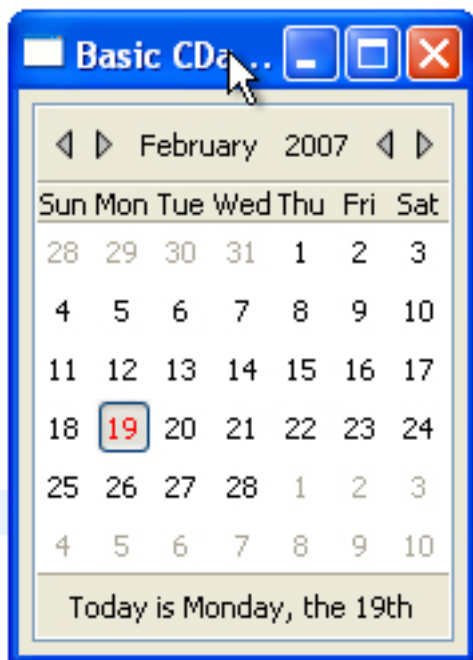
The example first creates a new SWT shell and initializes its layout. Next, a new CDateTime widget is added with the style CDT.DROP\_DOWN. Finally, the shell is centered on the screen and opened. To see this code in action, right-click the class and select **Run As > SWT Application**. You should see a window similar to Figure 1 when the combo box drops down.

### Figure 1. The example date drop-down



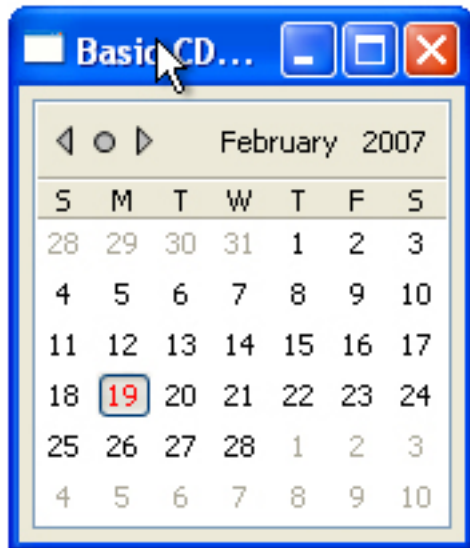
Change the CDT.DROP\_DOWN style to CDT.SIMPLE and rerun the example. You now see the calendar widget that was previously in the pop-up embedded in the window.

**Figure 2. A graphical calendar**



Next, OR in the CDT.COMPACT style to the existing styles. Doing so creates a slightly more compact version of the calendar, as shown below.

**Figure 3. The calendar in compact mode**

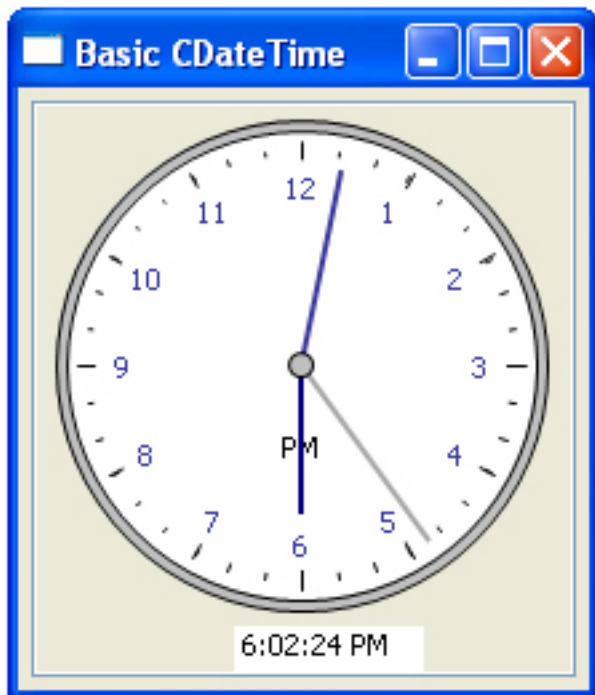


---

## Section 6. Using CDateTime for time selection

In addition to letting you graphically pick a date, the CDateTime widget also supports a number of visual ways to select a time. The first is an analog clock representation. You can display it by changing the CDT.COMPACT style constant to be CDT.TIME\_MEDIUM. Running the example now shows a window similar to Figure 4.

**Figure 4. The analog clock**



Click an hour, a minute, or the second hand on the clock, and drag it. Using this technique, you can adjust the time selected. The CDateTime can also be changed to allow time selection with a spinner. Modify the constructor style to include the OR'd style of CDT.SPINNER. You should see a window similar to the one shown below.

**Figure 5. The clock with a spinner**



A more discrete time selection appearance is also available. Remove the CDT.SPINNER style and, instead, OR in the styles of CDT.CLOCK\_DISCRETE and CDT.HORIZONTAL. Doing so creates a button-based time selector that supports a.m./p.m., hours and minutes in 5-minute increments, as shown below.

**Figure 6. Discrete time selection**



The time-based selectors can be used in drop-down mode just like the date-based selectors. You can see this in action by changing the CDT.SIMPLE style back to CDT.DROP\_DOWN.

## Section 7. Manipulate values and formats

The CDateTime widget supports a number of methods to listen to events and manipulate its value. Include the code in Listing 2 after the construction of the `cdt` variable.

**Listing 2. Selection and modification events**

```

        cdt.addSelectionListener(new
SelectionListener() {
            public void \
widgetDefaultSelected(SelectionEvent e) {
                // TODO
                Auto-generated method stub
            }
            public void \
widgetSelected(SelectionEvent e) {
                System.out.println\
("WidgetSelected " \
                +
                cdt.getSelection());
            }
        });
        cdt.addModifyListener\
(new ModifyListener() {

```

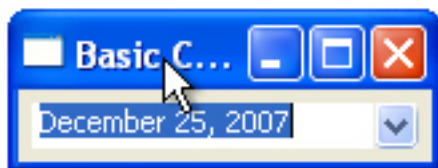
```
public void \
modifyText(ModifyEvent e) {
System.out.println\
("ModifyEvent " + cdt.getSelection());
}}
```

This demonstrates CDateTime's support for selection and modification events. It also shows the `getSelection()` method, which you can use to get the currently selected date/time. Correspondingly, a `setSelection(Date date)` method is available to change the date.

The CDateTime widget also lets you change the textual format of the current date. This can be done in two ways. First, a number of format constants are available, such as `CDT.DATE_LONG` and `CDT.DATE_SHORT`. You can order these together with time constants such as `CDT.TIME_SHORT`. Include the line of code below, and you should see a window similar to Figure 7 when it runs.

```
final CDateTime cdt = new CDateTime(shell, CDT.BORDER | CDT.DROP_DOWN
| CDT.DATE_MEDIUM | CDT.TIME_MEDIUM);
```

**Figure 7. Text formatting**



Alternatively, you can set the textual display using a specific time pattern. The pattern is defined by the `SimpleDateFormat` operators. Modify the example to set the pattern, as shown below.

```
cdt.setPattern("'Meeting on' EEEE, MMMM d '@' h:mm 'in the'a");
```

Run the example to see the effects of this change.

---

## Section 8. CompositeTable

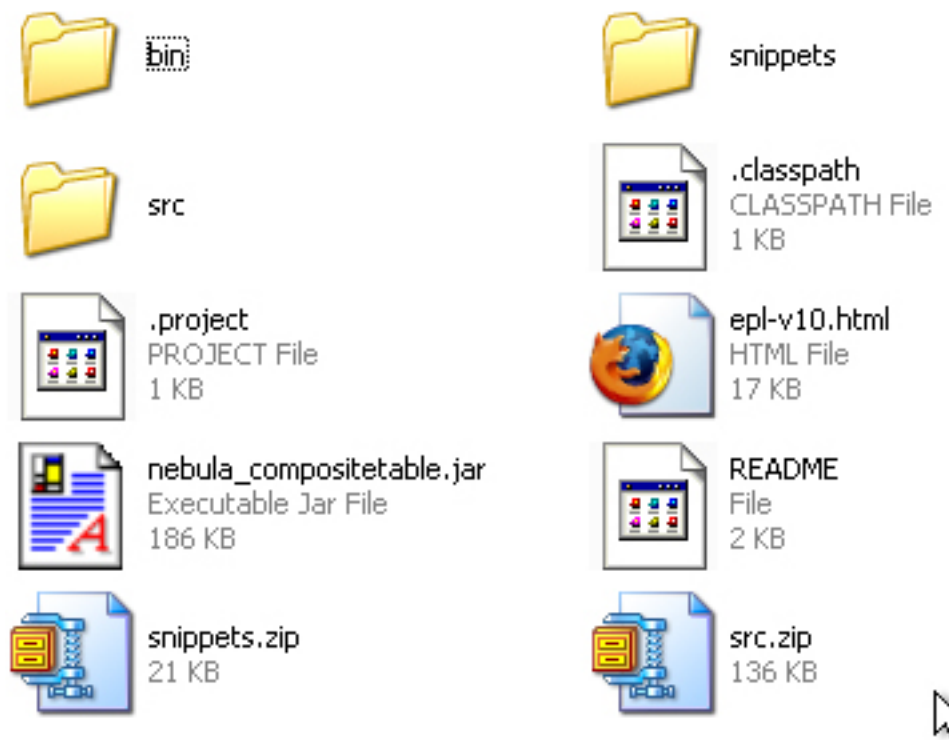
CompositeTable is a flexible widget that provides table-based displays of data while giving you the power to create a custom layout and display for the information in each row. The first step in using this widget is to [download the beta version](#). The compressed file on the Web site isn't packaged as an Eclipse project out of the box,

so you need to perform a few setup steps before using the composite table:

1. Expand the compressed file into a directory. The directory now contains src.zip and snippets.zip among its files.
2. Expand src.zip into the directory; it creates a src folder.
3. Expand snippets.zip into the same directory, but add \snippets to the expansion location. This is necessary because the top level of snippets.zip is the package name.

At this point, your composite table directory should appear similar to Figure 8. You're ready to create an Eclipse project for the CompositeTable source.

**Figure 8. The composite table directory**



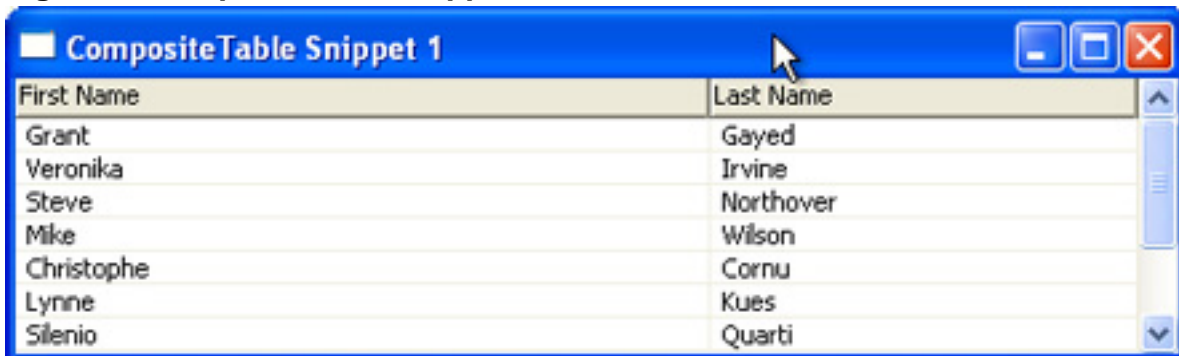
1. Select **File > New > Project**.
2. In the Project window, select **Java Project**.
3. On the next screen, name the project org.eclipse.swt.nebula.widgets.compositetable.
4. Select the **Create project from existing source** option and select the

- directory into which you expanded the composite table. Click **Next**.
5. Right-click the src and snippets folders, and mark them as source folders.
  6. Change to the **Libraries** tab and select **Add Library**. Click **Next** and select the workspace JRE.
  7. Change to the **Projects** tab and add the SWT projects.
  8. Click **Finish** to create the project. It should compile with no errors.

## Section 9. The basics of CompositeTable

The easiest way to learn the CompositeTable application program interface (API) is to review the snippets included in the download. Open CompositeTableSnippet1 and scroll down to the `main` method. You can see that the widget is created, and Header and Row objects are created. These objects determine how the CompositeTable displays headers and rows. If you look at the Header implementation, you see that it contains two Label widgets positioned using a GridLayout. Run the example, and the window is similar to that shown below.

**Figure 9. CompositeTableSnippet1**



Next, change the Header class, as shown below.

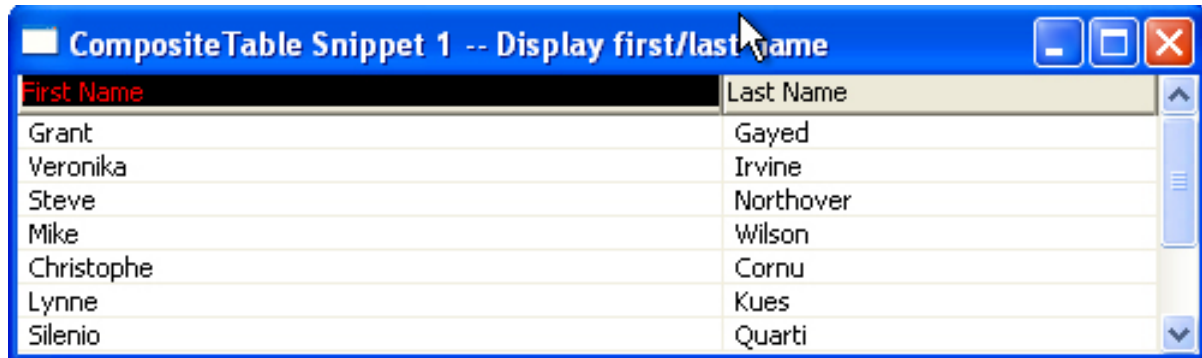
### Listing 3. Header changes

```
public Header(Composite parent, int style) {
    super(parent, style);
    setLayout(new GridLayout(new int[] { 160, 100 }, false));
    Label first = new Label(this, SWT.NULL);
    first.setText("First Name");
    first.setForeground(Display.getCurrent().getSystemColor(SWT.COLOR_RED));
    first.setBackground(Display.getCurrent().getSystemColor(SWT.COLOR_BLACK));
    new Label(this, SWT.NULL).setText("Last Name");
}
```

```
}
```

Running the example now shows the Header changes, as displayed in Figure 10.

**Figure 10. Change the header**



First Name	Last Name
Grant	Gayed
Veronika	Irvine
Steve	Northover
Mike	Wilson
Christophe	Cornu
Lynne	Kues
Silenio	Quarti

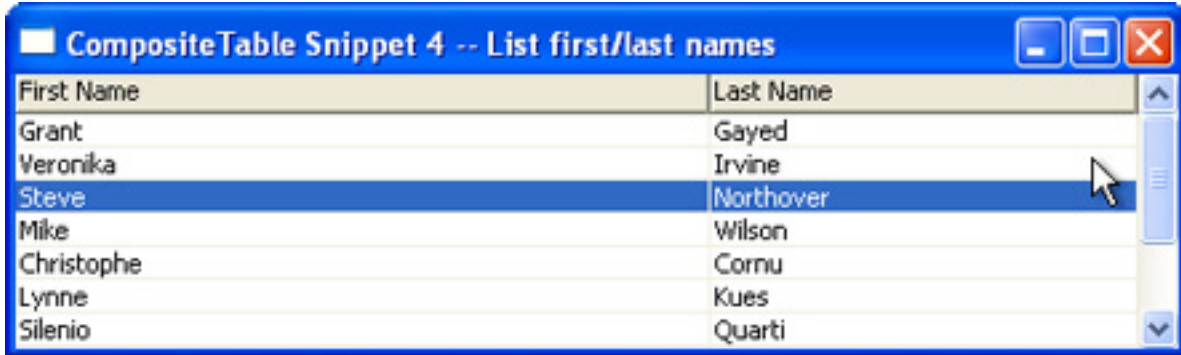
The other key component to the display of a `CompositeTable` is the `Row` object. Reviewing the `Row` class definition in `CompositeTableSnippet1` reveals two `Text` widgets positioned using `GridLayout`. The final piece of the puzzle is the content provider added in the `main` method. For each row, the `CompositeTable` calls the `refresh` method, passing in a row offset and a `Row` object. The method is then responsible for retrieving the appropriate model data and setting it on the `Row` object for display.

---

## Section 10. Implement common date functions with `CompositeTable`

Because the `CompositeTable`'s display is controlled by user-supplied composites for rows and headers, common table functions must be implemented differently from if you were using a Swing-based renderer paradigm. `CompositeTableSnippet4` demonstrates traditional table selection by row. Run the snippet to see this functionality in action, as shown below.

**Figure 11. `CompositeTableSnippet4`**



Looking at the implementation, the Row object extends `AbstractSelectableRow`. If you drill into this class, you see it implements a number of listeners. Its constructor adds the Row object as it's created as a listener to its parent, which is the Composite Table itself. Looking at the `focusGained()` and `focusLost()` methods, you can see that the row changes its background color corresponding to its state. Composite Table functions, such as selection, are handled by the rows themselves instead of the parent table.

Sortable rows are another interesting standard table feature implementation. To try this, run `CompositeTableSnippet5`, which displays as shown below.

**Figure 12. CompositeTableSnippet5**



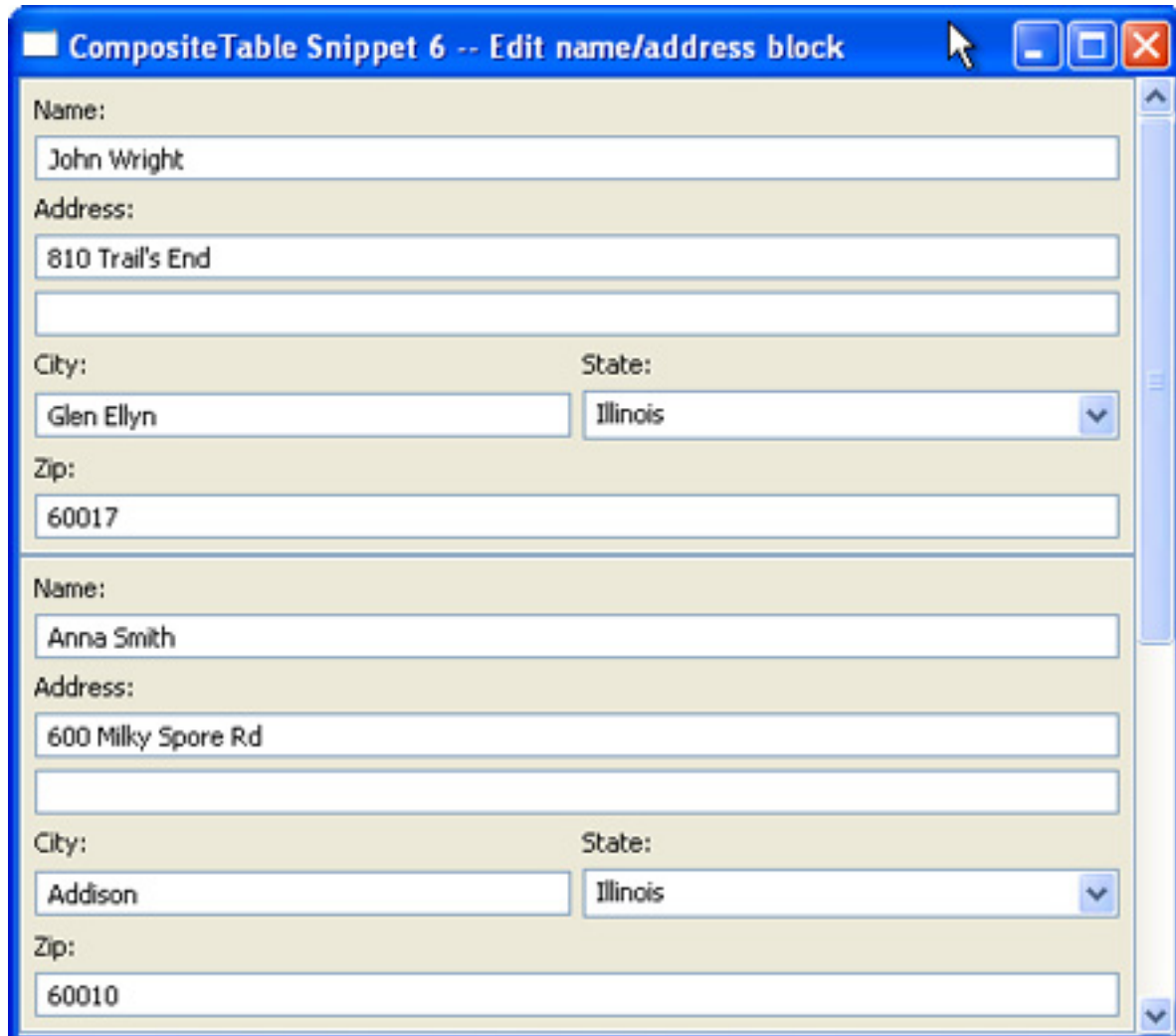
Looking at the implementation, the Header object extends `AbstractNativeHeader`. The `AbstractNativeHeader` includes a standard SWT Table with no rows, as shown in the constructor. When the columns are initialized in the `initializeColumns()` method, a selection listener is added to each column. When a column is selected, it calls the `sortOnColumn()` method. Switching back to the implementation in Header in `CompositeTableSnippet05`, you can see it sorts the data based on the column and then refreshes the table, causing the data to be presented in an updated state.

## Section 11. Nontraditional displays using

## CompositeTable

The examples you have seen so far have been traditional-looking tables. However, this doesn't have to be the case. Running CompositeTableSnippet6 displays the window shown below.

**Figure 13. A composite-based table row**



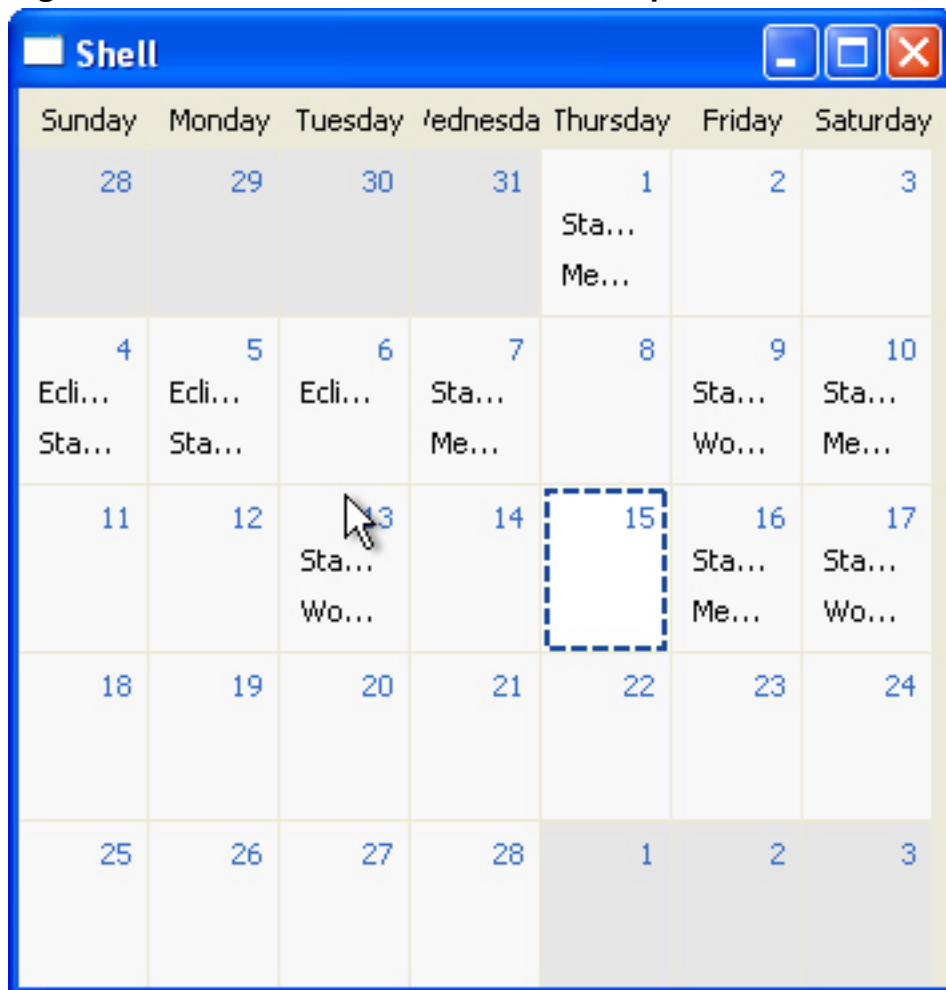
The screenshot shows a window titled "CompositeTable Snippet 6 -- Edit name/address block". The window contains two rows of a table, each with a light beige background. The first row has the following fields: "Name:" with the value "John Wright", "Address:" with the value "810 Trail's End", "City:" with the value "Glen Ellyn", "State:" with a dropdown menu showing "Illinois", and "Zip:" with the value "60017". The second row has the following fields: "Name:" with the value "Anna Smith", "Address:" with the value "600 Milky Spore Rd", "City:" with the value "Addison", "State:" with a dropdown menu showing "Illinois", and "Zip:" with the value "60010". The window has a blue title bar and standard Windows window controls (minimize, maximize, close) on the right side.

The implementation of the Row object doesn't simply contain a number of text widgets like the previous examples. Instead, it's a complicated multiline label and text widget layout. The table's refresh method sets the data into the appropriate text widgets.

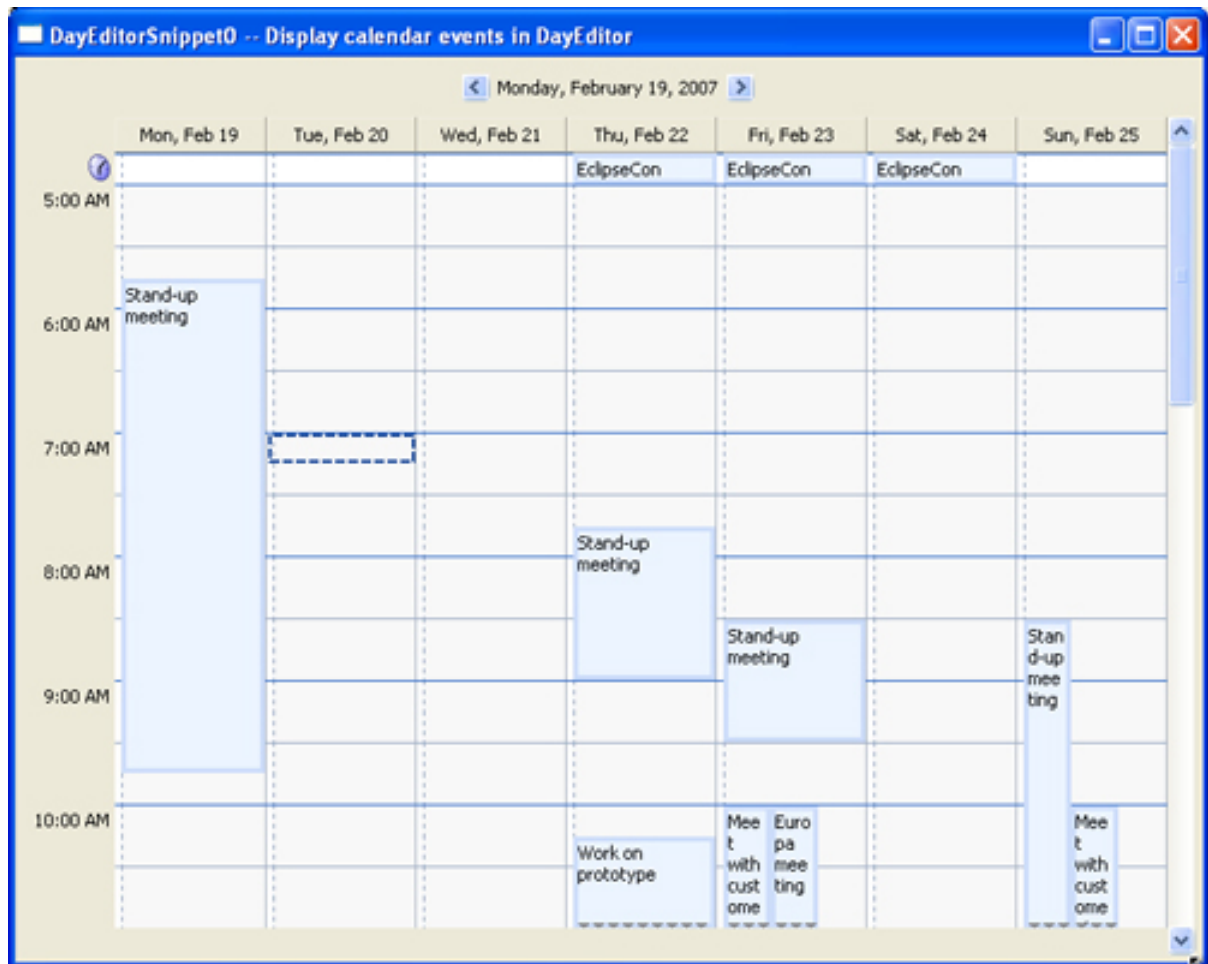
Although this tutorial doesn't go into detail explaining their implementations, DayEditorSnippet0 and MonthCalendarSnippet demonstrate how you can use the

CompositeTable widget to display more complicated nontraditional views of data. These are shown in figures 14 and 15.

**Figure 14. Present a month view with CompositeTable**



**Figure 15. Present a day view**



## Section 12. The Nebula Grid widget

The [Nebula Grid widget](#) is a tabular component that significantly adds to the capabilities found in the standard SWT Table component. This section details the usage of the Grid, including features such as column grouping, column spanning, checkbox cells, tree items, and cell selection. The Grid widget is in alpha status, and its API is subject to change.

### A simple Grid example

The code in Listing 4 represents a simple example of a Grid. A Grid is created and added to the shell with both horizontal and vertical scroll bars. The next line sets the column headers visible.

GridColumn s are created and added to the Grid, designating the columns that will be

shown, their titles, and their associated widths. GridItems represent a row in the tabular data. Each GridItem is populated by setting text at the appropriate index.

#### Listing 4. GridExample1

```
public class GridExample1 {
    public static void main(String...
args) {
        Display display = new
Display();
        Shell shell = new
Shell(display);
        FillLayout();
        shell.setLayout(new
FillLayout());
        Car car1 = new
Car(133, "2007", "Honda",
"CR-V", Car.CarType.SUV, 322, \
"Glacier Blue", true);
        Car car2 = new
Car(134, "2002", "BMW",
"M Roadster", Car.CarType.CONVERTIBLE, \
40233, "Red", false);
        Car car3 = new
Car(135, "2002", "Acura",
"RSX", Car.CarType.COUCPE, \
53283, "Black", false);

        Grid grid = new Grid(shell, SWT.BORDER |
SWT.V_SCROLL | SWT.H_SCROLL);
        grid.setHeaderVisible(true);

        GridColumn idColumn =
new GridColumn(grid, SWT.NONE);
        idColumn.setText("Car
Number");
        idColumn.setWidth(100);

        GridColumn yearColumn = new
GridColumn(grid, SWT.NONE);
        yearColumn.setText("Year");
        yearColumn.setWidth(50);

        GridColumn makeColumn = new
GridColumn(grid, SWT.NONE);
        makeColumn.setText("Make");
        makeColumn.setWidth(100);

        GridColumn modelColumn = new
GridColumn(grid, SWT.NONE);
        modelColumn.setText("Model");
        modelColumn.setWidth(100);

        GridColumn typeColumn = new
GridColumn(grid, SWT.NONE);
        typeColumn.setText("Type");
        typeColumn.setWidth(100);

        addCarRow(car1, grid);
        addCarRow(car2, grid);
        addCarRow(car3, grid);

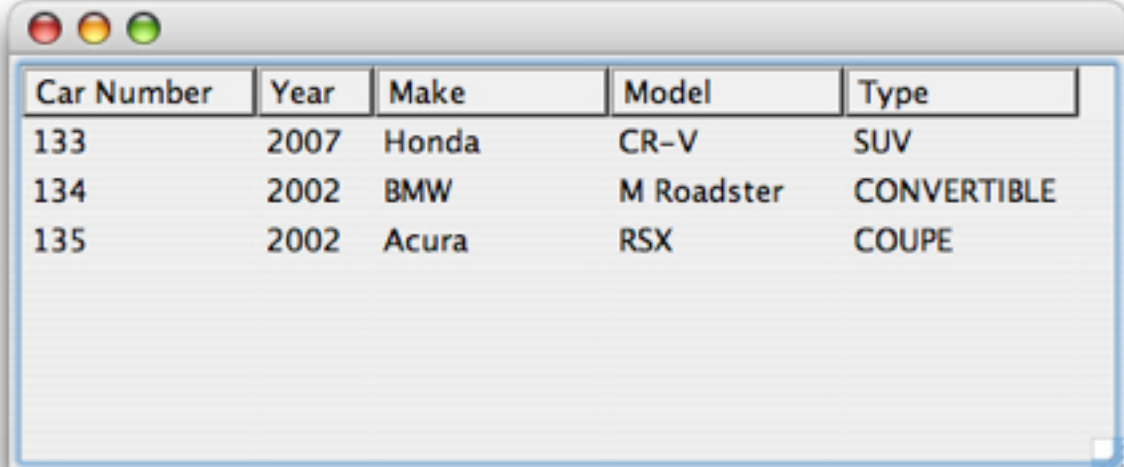
        shell.setSize(475, 200);
        shell.open();
        while (!shell.isDisposed())
        {
            if
```

```
(!display.readAndDispatch())
display.sleep();
    }
    display.dispose();
}

private static void addCarRow(Car
car, Grid grid) {
    GridItem item = new
GridItem(grid, SWT.NONE);
item.setText(String.valueOf(car.getCarNumber()));
item.setText(1,car.getYear());
    item.setText(2,
car.getMake());
    item.setText(3,
car.getModel());
    item.setText(4,
car.getCarType().toString());
}
}
```

Run the example by selecting **Run > Run As > SWT Application**. Figure 16 shows the result rendered in Mac OS X.

**Figure 16. GridExample1**



Car Number	Year	Make	Model	Type
133	2007	Honda	CR-V	SUV
134	2002	BMW	M Roadster	CONVERTIBLE
135	2002	Acura	RSX	COUPE

The functionality demonstrated in this example is also available in the SWT Table widget.

---

## Section 13. Checkboxes, row headers, changing cell color, and cell selection

Row headers let you label each row in the Grid. By default, the header for each row

is the ordinal number of the row in the Grid. To display row headers, call `setRowHeaderVisible(true)` on the Grid. Each GridItem can specify its own header. This is accomplished by calling `setHeaderText("Header text")` on the item.

You can designate a GridColumn as a column that contains a checkbox, which can be useful for rendering the value of a Boolean field. The code below creates a GridColumn that contains a checkbox.

```
GridColumn availableColumn = new GridColumn(grid, SWT.CHECK);
```

The checked status of the checkbox in a particular cell in the Grid is determined by its associated GridItem. You can set the status by calling the `setChecked(int, boolean)` method of the appropriate GridItem and passing the index of the column and the desired Boolean value. You can retrieve the value of the checkbox by calling the `getChecked(int)` method on the GridItem. Text can also be present in a checkbox cell. This can be accomplished by calling the `setText(int, String)` method on the GridItem, and passing the index of the column of the checkbox item and the text to be displayed.

You can independently change the color of the background and text, as well as the font of the text of each cell in the Grid. To do so, call the methods in Listing 5 on the appropriate GridItem.

### Listing 5. Change colors and fonts

```
item.setBackground(index, Color)
item.setFont(index, Font);

item.setForeground(index, Color);
```

A nice feature of the Grid that can enable spreadsheet-like behavior is the ability to enable cell selection. Single cells, a group of cells, entire columns, and entire rows can be selected. By default, a Grid is set to allow the selection of a single row at a time. The selection of multiple rows can be specified by ORing `SWT.MULTI` with the rest of the style integer supplied in the Grid constructor. An example appears below.

```
Grid grid = new Grid(shell, SWT.MULTI | SWT.V_SCROLL | SWT.H_SCROLL);
```

To enable the selection of single cells or groups of cells, call `setCellSelectionEnabled(true)` on the Grid. By default, the cells in all columns are selectable. To disallow the selection of cells in a column, call the `setCellSelectionEnabled` method on the appropriate GridColumn and pass it a value of false. To determine which cells are currently selected, call the

`getCellSelection()` method on the Grid. An array of Point objects containing the coordinates of each selected cell is returned.

Listing 6 demonstrates the use of row headers, checkboxes, cell colors, and cell selection. Experiment with the behavior of cell selection. Clicking a column header should select all the cells in that column. Clicking a row header should select all the cells in that row. Note that none of the cells in the first column, Car Number, should be selectable.

## Listing 6. GridExample2

```
public class GridExample2 {
    public static void main(String... args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout(new FillLayout());
        Grid grid = new Grid(shell, SWT.BORDER | SWT.V_SCROLL | SWT.H_SCROLL);

        grid.setHeaderVisible(true);
        grid.setRowHeaderVisible(true); // show Row Headers
        grid.setSelectionEnabled(true); //allow Cell Selection
        Car car1 = new Car(133, "2007", "Honda",
            "CR-V", Car.CarType.SUV, 322, "Glacier Blue", true);
        Car car2 = new Car(134, "2002", "BMW", \
            "M Roadster", Car.CarType.CONVERTIBLE, \
40233, "Red", false);
        Car car3 = new Car(135, "2002", "Acura",
            "RSX", Car.CarType.COUCPE, 53283, "Black", false);

        GridColumn idColumn = new GridColumn(grid, SWT.NONE);
        idColumn.setText("Car Number");
        idColumn.setWidth(100);
        //don't allow cells in the idColumn to be selected
        idColumn.setSelectionEnabled(false);

        GridColumn yearColumn = new GridColumn(grid, SWT.NONE);
        yearColumn.setText("Year");
        yearColumn.setWidth(50);

        GridColumn makeColumn = new GridColumn(grid, SWT.NONE);
        makeColumn.setText("Make");
        makeColumn.setWidth(100);

        GridColumn modelColumn = new GridColumn(grid, SWT.NONE);
        modelColumn.setText("Model");
        modelColumn.setWidth(100);

        GridColumn typeColumn = new GridColumn(grid, SWT.NONE);
        typeColumn.setText("Type");
        typeColumn.setWidth(100);

        GridColumn availableColumn = new GridColumn(grid, SWT.CHECK);
        availableColumn.setText("Available");
        availableColumn.setWidth(75);

        GridItem item1 = new GridItem(grid, SWT.NONE);
        item1.setHeaderText("Row Header");
        item1.setText(0, String.valueOf(car1.getCarNumber()));
        item1.setText(1, car1.getYear());
        item1.setText(2, car1.getMake());
        item1.setText(3, car1.getModel());
        item1.setText(4, car1.getCarType().toString());
    }
}
```

```

//set whether the check box in this column and row is checked
item1.setChecked(5, car1.isAvailable());

GridItem item2 = new GridItem(grid, SWT.NONE);
item2.setText(0, String.valueOf(car2.getCarNumber()));
item2.setText(1, car2.getYear());
item2.setText(2, car2.getMake());
item2.setText(3, car2.getModel());
item2.setText(4, car2.getCarType().toString());
item2.setChecked(5, car2.isAvailable());
//set background to blue
item2.setBackground(4, new Color(null, 0, 0, 255));
//change font
item2.setFont(4, new Font(null, "Arial", 12, SWT.BOLD | SWT.ITALIC));
//set text color to red
item2.setForeground(4, new Color(null, 255, 0, 0));

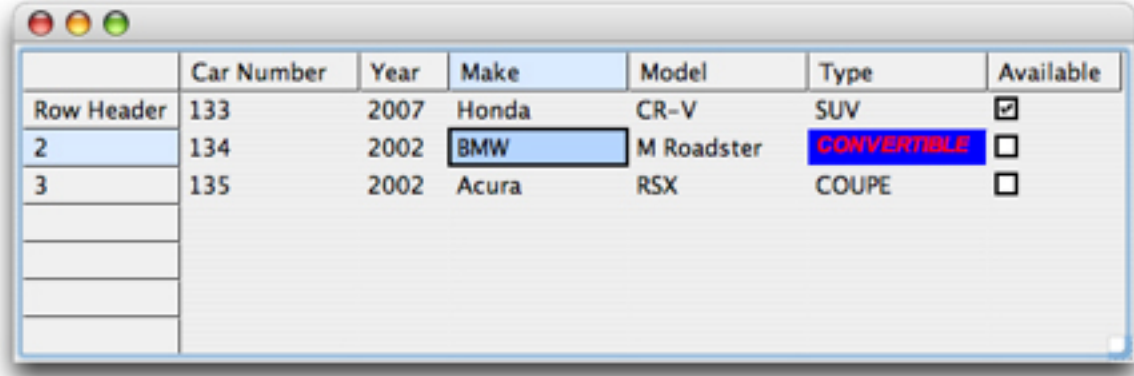
GridItem item3 = new GridItem(grid, SWT.NONE);
item3.setText(0, String.valueOf(car3.getCarNumber()));
item3.setText(1, car3.getYear());
item3.setText(2, car3.getMake());
item3.setText(3, car3.getModel());
item3.setText(4, car3.getCarType().toString());
item3.setChecked(5, car2.isAvailable());

shell.setSize(625, 200);
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}
}

```

Running the example in Mac OS X provides the result shown below.

**Figure 17. GridExample2**



	Car Number	Year	Make	Model	Type	Available
Row Header	133	2007	Honda	CR-V	SUV	<input checked="" type="checkbox"/>
2	134	2002	BMW	M Roadster	CONVERTIBLE	<input type="checkbox"/>
3	135	2002	Acura	RSX	COUPE	<input type="checkbox"/>

## Section 14. Create trees of Grid items and column spans

The Grid lets you create trees of GridItems, signifying a relationship between items and subitems. Items can be nested  $n$  layers deep, although simple one- or two-layer nestings are much more common. To create a tree of items:

1. On the appropriate GridColumn, call the `setTree()` method, passing `true` as the value. This designates the column as a column that allows a tree of subitems. The UI control for toggling the view of branch objects appears in this column.
2. Create a GridItem that is the root of the tree. This GridItem should have the Grid as its parent.
3. Create GridItems that are the next branch of the tree. To do so, pass the reference to the parent object (in this case, the root of the tree) to the constructor of each branch GridItem. Repeat this step for each level of the tree. The parent object passed to the constructor of each item is the GridItem object one level above it in the tree.

A data field in a GridItem can also be set to span a number of columns. To do so, call the `setColumnSpan(int, int)` method of the appropriate GridItem. The first argument is the index of the column that will be affected; the second argument specifies the number of subsequent columns that will be spanned.

In Listing 7, trees of GridItems are created to group cars into a specific rental category. The category column is set to span across all columns. Its background color and font are also changed.

### Listing 7. GridExample3

```
public class GridExample3 {
    public static void main(String... args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout(new FillLayout());

        Grid grid = new Grid(shell, SWT.BORDER |
            SWT.V_SCROLL | SWT.H_SCROLL);
        grid.setHeaderVisible(true);

        Car car1 = new Car(133, "2007", "Chevy",
            "Cobalt", Car.CarType.COUCPE, \
4321, "Yellow", true);
        Car car2 = new Car(134, "2007", "Chevy",
            "Cobalt", Car.CarType.COUCPE, \
4321, "Yellow", true);
        Car car3 = new Car(135, "2006", "Ford",
            "Focus", Car.CarType.COUCPE, \
15343, "Red", true);
        Car car4 = new Car(136, "2006", "Chrysler",
            "Sebring", Car.CarType.SEDAN, \
12932, "Black", false);
        Car car5 = new Car(137, "2002", "Ford",
```

```

4342, "Red", true);

        "Mustang", Car.CarType.COUCPE, \

        GridColumn rentalTypeColumn = new GridColumn(grid, SWT.NONE);
rentalTypeColumn.setText("Rental Grade");
rentalTypeColumn.setWidth(100);
rentalTypeColumn.setTree(true);

GridColumn idColumn = new GridColumn(grid, SWT.NONE);
idColumn.setText("Car Number");
idColumn.setWidth(100);

GridColumn yearColumn = new GridColumn(grid, SWT.NONE);
yearColumn.setText("Year");
yearColumn.setWidth(50);

GridColumn makeColumn = new GridColumn(grid, SWT.NONE);
makeColumn.setText("Make");
makeColumn.setWidth(100);

        GridColumn modelColumn = new GridColumn(grid, SWT.NONE);
modelColumn.setText("Model");
modelColumn.setWidth(100);

GridColumn typeColumn = new GridColumn(grid, SWT.NONE);
typeColumn.setText("Type");
typeColumn.setWidth(100);

        GridColumn availableColumn = new GridColumn(grid,
                SWT.CHECK | SWT.CENTER);
availableColumn.setText("Available");
availableColumn.setWidth(75);

GridColumn compactItem = new GridItem(grid, SWT.CENTER);
compactItem.setText(0, "Compact");

        compactItem.setFont(new Font(null,
                "Arial", 18, SWT.BOLD | SWT.ITALIC));
compactItem.setColumnSpan(0, 6);
compactItem.setBackground(0, new Color(null, 0, 255, 0));

GridColumn item1 = new GridItem(compactItem, SWT.NONE);
item1.setText(1, String.valueOf(car1.getCarNumber()));
item1.setText(2, car1.getYear());
item1.setText(3, car1.getMake());
item1.setText(4, car1.getModel());
item1.setText(5, car1.getCarType().toString());
item1.setChecked(6, car1.isAvailable());

GridColumn item2 = new GridItem(compactItem, SWT.NONE);
item2.setText(1, String.valueOf(car2.getCarNumber()));
item2.setText(2, car2.getYear());
item2.setText(3, car2.getMake());
item2.setText(4, car2.getModel());
item2.setText(5, car2.getCarType().toString());
item2.setChecked(6, car2.isAvailable());

GridColumn item3 = new GridItem(compactItem, SWT.NONE);
item3.setText(1, String.valueOf(car3.getCarNumber()));
item3.setText(2, car3.getYear());
item3.setText(3, car3.getMake());
item3.setText(4, car3.getModel());
item3.setText(5, car3.getCarType().toString());
item3.setChecked(6, car3.isAvailable());

GridColumn midSizedItem = new GridItem(grid, SWT.NONE);
midSizedItem.setText(0, "Mid-Sized");

```

```

        midSizedItem.setFont(new Font(null,
                                     "Arial", 18, SWT.BOLD | SWT.ITALIC));
        midSizedItem.setColumnSpan(0,6);
        midSizedItem.setBackground(0, new Color(null, 0, 255, 255));

        GridItem item4 = new GridItem(midSizedItem, SWT.NONE);
        item4.setText(1, String.valueOf(car4.getCarNumber()));
        item4.setText(2, car4.getYear());
        item4.setText(3, car4.getMake());
        item4.setText(4, car4.getModel());
        item4.setText(5, car4.getCarType().toString());
        item4.setChecked(6, car4.isAvailable());

        GridItem item5 = new GridItem(midSizedItem, SWT.NONE);
        item5.setText(1, String.valueOf(car5.getCarNumber()));
        item5.setText(2, car5.getYear());
        item5.setText(3, car5.getMake());
        item5.setText(4, car5.getModel());
        item5.setText(5, car5.getCarType().toString());
        item5.setChecked(6, car5.isAvailable());

        shell.setSize(700, 200);
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}

```

Running the example provides the result shown below.

**Figure 18. GridExample3**

Rental Grade	Car Number	Year	Make	Model	Type	Available
<b>Compact</b>						
	133	2007	Chevy	Cobalt	COUPE	<input checked="" type="checkbox"/>
	134	2007	Chevy	Cobalt	COUPE	<input checked="" type="checkbox"/>
	135	2006	Ford	Focus	COUPE	<input checked="" type="checkbox"/>
<b>Mid-Sized</b>						
	136	2006	Chrysler	Sebring	SEDAN	<input type="checkbox"/>
	137	2002	Ford	Mustang	COUPE	<input checked="" type="checkbox"/>

## Section 15. Column groups

The Nebula Grid lets you group related columns into GridColumnGroups. This is accomplished by creating a GridColumnGroup object, specifying the Grid as its parent in the constructor and a style of SWT.NONE. You then create the columns in the group, passing the reference to the newly created GridColumnGroup to the

constructor of each `GridColumn` as the parent object.

The example in Listing 8 builds on the previous example by creating two column groups: one for basic automobile information and one for details about the car. A few detail fields have been added, as well.

### Listing 8. GridExample4

```
public class GridExample4 {
    public static void main(String... args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout(new FillLayout());

        Grid grid = new Grid(shell, \
            SWT.BORDER | SWT.V_SCROLL | SWT.H_SCROLL);
        grid.setHeaderVisible(true);
        Car car1 = new Car(133, "2007", "Chevy",
            "Cobalt", Car.CarType.COUCPE, \
            4321, "Yellow", true);
        Car car2 = new Car(134, "2007", "Chevy",
            "Cobalt", Car.CarType.COUCPE, \
            4321, "Yellow", true);
        Car car3 = new Car(135, "2006", "Ford",
            "Focus", Car.CarType.COUCPE, \
            15343, "Red", true);
        Car car4 = new Car(136, "2006", "Chrysler",
            "Sebring", Car.CarType.SEDAN, \
            12932, "Black", false);
        Car car5 = new Car(137, "2002", "Ford",
            "Mustang", Car.CarType.COUCPE, \
            4342, "Red", true);

        GridColumn rentalTypeColumn = new GridColumn(grid, SWT.NONE);
        rentalTypeColumn.setText("Rental Grade");
        rentalTypeColumn.setWidth(100);
        rentalTypeColumn.setTree(true);

        GridColumnGroup carGroup = new GridColumnGroup(grid, SWT.NONE);
        carGroup.setText("Automobile");

        GridColumn yearColumn = new GridColumn(carGroup, SWT.NONE);
        yearColumn.setText("Year");
        yearColumn.setWidth(50);

        GridColumn makeColumn = new GridColumn(carGroup, SWT.NONE);
        makeColumn.setText("Make");
        makeColumn.setWidth(100);

        GridColumn modelColumn = new GridColumn(carGroup, SWT.NONE);
        modelColumn.setText("Model");
        modelColumn.setWidth(100);

        GridColumnGroup carDetailsGroup =
            new GridColumnGroup(grid, SWT.NONE);
        carDetailsGroup.setText("Car Details");

        GridColumn idColumn = new GridColumn(carDetailsGroup, SWT.NONE);
        idColumn.setText("Car Number");
        idColumn.setWidth(100);

        GridColumn typeColumn = new GridColumn(carDetailsGroup, SWT.NONE);
        typeColumn.setText("Type");
        typeColumn.setWidth(100);
    }
}
```

```

GridColumn mileageColumn =
    new GridColumn(carDetailsGroup, SWT.NONE);
mileageColumn.setText("Mileage");
mileageColumn.setWidth(100);

GridColumn colorColumn = new GridColumn(carDetailsGroup, SWT.NONE);
colorColumn.setText("Color");
colorColumn.setWidth(100);

GridColumn availableColumn =
    new GridColumn(grid, SWT.CHECK | SWT.CENTER);
availableColumn.setText("Available");
availableColumn.setWidth(75);

GridItem compactItem = new GridItem(grid, SWT.CENTER);
compactItem.setText(0, "Compact");
compactItem.setFont(new Font(null, "Arial", 16, SWT.ITALIC));
compactItem.setColumnSpan(0, 8);
compactItem.setBackground(0, new Color(null, 0,255,0));

addCarRow(car1, compactItem);
addCarRow(car2, compactItem);
addCarRow(car3, compactItem);

GridItem midSizedItem = new GridItem(grid, SWT.NONE);
midSizedItem.setText(0, "Mid-Sized");
midSizedItem.setFont(new Font(null, "Arial", 16, SWT.ITALIC));
midSizedItem.setColumnSpan(0,8);
midSizedItem.setBackground(0, new Color(null, 0, 255, 255));

addCarRow(car4, midSizedItem);
addCarRow(car5, midSizedItem);

shell.setSize(850, 200);
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}

private static void addCarRow(Car car, GridItem parentItem) {
    GridItem item1 = new GridItem(parentItem, SWT.NONE);
    item1.setText(1, car.getYear());
    item1.setText(2, car.getMake());
    item1.setText(3, car.getModel());
    item1.setText(4, String.valueOf(car.getCarNumber()));
    item1.setText(5, car.getCarType().toString());
    item1.setText(6, String.valueOf(car.getMileage()));
    item1.setText(7, car.getColor());
    item1.setChecked(8, car.isAvailable());
}
}

```

Running the example provides the result shown below.

### Figure 19. GridExample4

Rental Grade	Automobile			Car Details				Available
	Year	Make	Model	Car Number	Type	Mileage	Color	
<b>Compact</b>								
	2007	Chevy	Cobalt	133	COUPE	4321	Yellow	<input checked="" type="checkbox"/>
	2007	Chevy	Cobalt	134	COUPE	4321	Yellow	<input checked="" type="checkbox"/>
	2006	Ford	Focus	135	COUPE	15343	Red	<input checked="" type="checkbox"/>
<b>Mid-Sized</b>								
	2006	Chrysler	Sebring	136	SEDAN	12932	Black	<input type="checkbox"/>

## Create an expandable column group

In addition to grouping related columns, you can create a `GridColumnGroup` in a manner that allows it to be expanded and contracted, hiding some of its columns. To do so, pass a style of `SWT.TOGGLE` when constructing the `GridColumnGroup`. The `GridColumnGroup` can be set to begin in its expanded or contracted state by calling its `setExpanded()` method.

You must specify whether each `GridColumn` belonging to the group is a summary or a detail field (or both). This is accomplished by calling the `setDetail()` and `setSummary()` methods of `GridColumn`, respectively. By default, any column not explicitly set as a detail field is assumed to be both a summary and a detail column, and is displayed in the expanded and contracted state of the `GridColumnGroup`.

The example in Listing 9 creates a toggled `GridColumnGroup` that displays the details of each car. The group starts in the contracted state.

### Listing 9. GridExample5

```
public class GridExample5 {
    public static void main(String... args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout(new FillLayout());

        Grid grid = new Grid(shell,
            SWT.BORDER | SWT.V_SCROLL | SWT.H_SCROLL);
        grid.setHeaderVisible(true);

        Car car1 = new Car(133, "2007", "Chevy",
            "Cobalt", Car.CarType.COUPE, \
4321, "Yellow", true);
        Car car2 = new Car(134, "2007", "Chevy",
            "Cobalt", Car.CarType.COUPE, \
6435, "Yellow", true);
        Car car3 = new Car(135, "2006", "Ford",
            "Focus", Car.CarType.COUPE, \
15343, "Red", true);
        Car car4 = new Car(136, "2006", "Chrysler",
            "Sebring", Car.CarType.SEDAN, \
12932, "Black", false);
        Car car5 = new Car(137, "2002", "Ford",
            "Mustang", Car.CarType.COUPE, \
4342, "Red", true);
```

```
GridColumn rentalTypeColumn = new GridColumn(grid, SWT.NONE);
rentalTypeColumn.setText("Rental Grade");
rentalTypeColumn.setWidth(100);
rentalTypeColumn.setTree(true);

GridColumnGroup carGroup = new GridColumnGroup(grid, SWT.NONE);
carGroup.setText("Automobile");

GridColumn yearColumn = new GridColumn(carGroup, SWT.NONE);
yearColumn.setText("Year");
yearColumn.setWidth(50);

GridColumn makeColumn = new GridColumn(carGroup, SWT.NONE);
makeColumn.setText("Make");
makeColumn.setWidth(100);

GridColumn modelColumn = new GridColumn(carGroup, SWT.NONE);
modelColumn.setText("Model");
modelColumn.setWidth(100);

GridColumnGroup carDetailsGroup =
    new GridColumnGroup(grid, SWT.TOGGLE);
carDetailsGroup.setText("Car Details");
// set the group to start off contracted
carDetailsGroup.setExpanded(false);

GridColumn idColumn = new GridColumn(carDetailsGroup, SWT.NONE);
idColumn.setText("Car Number");
idColumn.setWidth(100);

GridColumn typeColumn = new GridColumn(carDetailsGroup, SWT.NONE);
typeColumn.setText("Type");
typeColumn.setWidth(100);
// all of other the columns in the group are detail fields
typeColumn.setDetail(true);
typeColumn.setSummary(false);

GridColumn mileageColumn =
    new GridColumn(carDetailsGroup, SWT.NONE);
mileageColumn.setText("Mileage");
mileageColumn.setWidth(100);
mileageColumn.setDetail(true);
mileageColumn.setSummary(false);

GridColumn colorColumn = new GridColumn(carDetailsGroup, SWT.NONE);
colorColumn.setText("Color");
colorColumn.setWidth(100);
colorColumn.setDetail(true);
colorColumn.setSummary(false);

GridColumn availableColumn =
    new GridColumn(grid, SWT.CHECK | SWT.CENTER);
availableColumn.setText("Available");
availableColumn.setWidth(75);

GridItem compactItem = new GridItem(grid, SWT.CENTER);
compactItem.setText(0, "Compact");
compactItem.setFont(new Font(null, "Arial", 16, SWT.ITALIC));
compactItem.setColumnSpan(0, 8);
compactItem.setBackground(0, new Color(null, 0,255,0));

addCarRow(car1, compactItem);
addCarRow(car2, compactItem);
addCarRow(car3, compactItem);

GridItem midSizedItem = new GridItem(grid, SWT.NONE);
midSizedItem.setText(0, "Mid-Sized");
midSizedItem.setFont(new Font(null, "Arial", 16, SWT.ITALIC));
midSizedItem.setColumnSpan(0,8);
```

```

        midSizedItem.setBackground(0, new Color(null, 0, 255, 255));

        addCarRow(car4, midSizedItem);
        addCarRow(car5, midSizedItem);

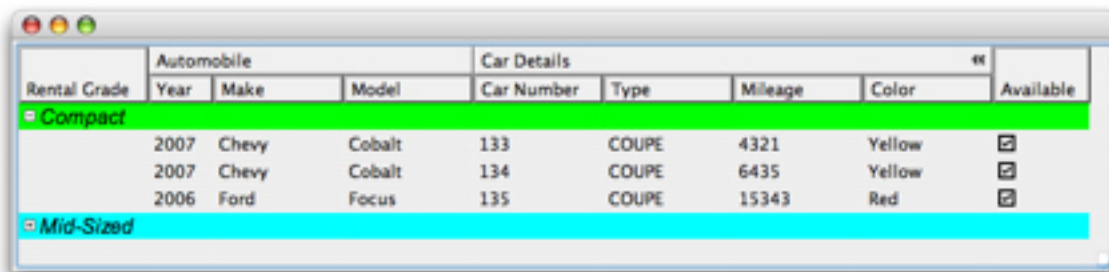
        shell.setSize(850, 200);
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }

    private static void addCarRow(Car car, GridItem parentItem) {
        GridItem item1 = new GridItem(parentItem, SWT.NONE);
        item1.setText(1, car.getYear());
        item1.setText(2, car.getMake());
        item1.setText(3, car.getModel());
        item1.setText(4, String.valueOf(car.getCarNumber()));
        item1.setText(5, car.getCarType().toString());
        item1.setText(6, String.valueOf(car.getMileage()));
        item1.setText(7, car.getColor());
        item1.setChecked(8, car.isAvailable());
    }
}

```

Running the example provides the result shown below (shown here in its expanded state).

**Figure 20. GridExample5**



**Add a summary column**

As mentioned, you can designate one or more GridColumns in a GridColumnGroup to be a summary field. This can be most useful for showing a summary value and allowing the user to toggle open the details. Listing 10 creates another GridColumnGroup that displays the total price a day of the rental when contracted. When expanded, the details of this total value are displayed, along with the Total Price column. Note how the Total Price column is designated as a summary and a detail column. If you send false as a parameter to the setDetail() method on this GridColumn, the Total Price column disappears when the group is expanded.

**Listing 10. GridExample6**

```

public class GridExample6 {

```

```

public static void main(String... args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setLayout(new FillLayout());

    //defaults to SWT.SINGLE - other options MULTI, NO_FOCUS, CHECK

        Grid grid = new Grid(shell, SWT.SINGLE |
                               SWT.BORDER | SWT.V_SCROLL | SWT.H_SCROLL);
    grid.setHeaderVisible(true);

        Car car1 = new Car(133, "2007", "Chevy",
                          "Cobalt", Car.CarType.COUCPE, \
4321, "Yellow", true, 122.00, .075);
        Car car2 = new Car(134, "2007", "Chevy",
                          "Cobalt", Car.CarType.COUCPE, \
6435, "Yellow", true, 122.00, .075);
        Car car3 = new Car(135, "2006", "Ford",
                          "Focus", Car.CarType.COUCPE, \
15343, "Red", true, 122.00, .075);
        Car car4 = new Car(136, "2006", "Chrysler",
                          "Sebring", Car.CarType.SEDAN, \
12932, "Black", false, 144.00, .075);
        Car car5 = new Car(137, "2002", "Ford",
                          "Mustang", Car.CarType.COUCPE, \
4342, "Red", true, 144.00, .075);

    GridColumn rentalTypeColumn = new GridColumn(grid, SWT.NONE);
    rentalTypeColumn.setText("Rental Grade");
    rentalTypeColumn.setWidth(100);
    rentalTypeColumn.setTree(true);

    GridColumnGroup carGroup = new GridColumnGroup(grid, SWT.NONE);
    carGroup.setText("Automobile");

    GridColumn yearColumn = new GridColumn(carGroup, SWT.NONE);
    yearColumn.setText("Year");
    yearColumn.setWidth(50);

    GridColumn makeColumn = new GridColumn(carGroup, SWT.NONE);
    makeColumn.setText("Make");
    makeColumn.setWidth(100);

    GridColumn modelColumn = new GridColumn(carGroup, SWT.NONE);
    modelColumn.setText("Model");
    modelColumn.setWidth(100);

    GridColumnGroup carDetailsGroup =
        new GridColumnGroup(grid, SWT.TOGGLE);
    carDetailsGroup.setText("Car Details");
    carDetailsGroup.setExpanded(false);

    GridColumn idColumn = new GridColumn(carDetailsGroup, SWT.NONE);
    idColumn.setText("Car Number");
    idColumn.setWidth(100);

    GridColumn typeColumn = new GridColumn(carDetailsGroup, SWT.NONE);
    typeColumn.setText("Type");
    typeColumn.setWidth(100);
    typeColumn.setDetail(true);
    typeColumn.setSummary(false);

    GridColumn mileageColumn =
        new GridColumn(carDetailsGroup, SWT.NONE);
    mileageColumn.setText("Mileage");
    mileageColumn.setWidth(100);
    mileageColumn.setDetail(true);
    mileageColumn.setSummary(false);

```

```

GridColumn colorColumn = new GridColumn(carDetailsGroup, SWT.NONE);
colorColumn.setText("Color");
colorColumn.setWidth(100);
colorColumn.setDetail(true);
colorColumn.setSummary(false);

        GridColumnGroup pricingGroup =
            new GridColumnGroup(grid, SWT.TOGGLE);
pricingGroup.setText("Pricing");
pricingGroup.setExpanded(false);

        GridColumn dailyRentalColumn =
            new GridColumn(pricingGroup, SWT.NONE);
dailyRentalColumn.setText("Daily Rental");
dailyRentalColumn.setWidth(100);
dailyRentalColumn.setDetail(true);
dailyRentalColumn.setSummary(false);

GridColumn taxColumn = new GridColumn(pricingGroup, SWT.NONE);
taxColumn.setText("Tax");
taxColumn.setWidth(100);
taxColumn.setDetail(true);
taxColumn.setSummary(false);

        GridColumn totalPriceColumn =
            new GridColumn(pricingGroup, SWT.NONE);
totalPriceColumn.setText("Total");
totalPriceColumn.setWidth(100);
totalPriceColumn.setDetail(true);
totalPriceColumn.setSummary(true);
        GridColumn availableColumn =
            new GridColumn(grid, SWT.CHECK |
SWT.CENTER);
availableColumn.setText("Available");
availableColumn.setWidth(75);

GridColumn compactItem = new GridItem(grid, SWT.CENTER);
compactItem.setText(0, "Compact");
compactItem.setFont(new Font(null, "Arial", 16, SWT.ITALIC));
compactItem.setColumnSpan(0, 11);
compactItem.setBackground(0, new Color(null, 0,255,0));

addCarRow(car1, compactItem);
addCarRow(car2, compactItem);
addCarRow(car3, compactItem);

GridColumn midSizedItem = new GridItem(grid, SWT.NONE);
midSizedItem.setText(0, "Mid-Sized");
midSizedItem.setFont(new Font(null, "Arial", 16, SWT.ITALIC));
midSizedItem.setColumnSpan(0,11);
midSizedItem.setBackground(0, new Color(null, 0, 255, 255));

addCarRow(car4, midSizedItem);
addCarRow(car5, midSizedItem);

shell.setSize(850, 200);
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}

private static void addCarRow(Car car, GridItem parentItem) {
    NumberFormat formatter = NumberFormat.getCurrencyInstance();
    GridItem item1 = new GridItem(parentItem, SWT.NONE);
    item1.setText(1, car.getYear());
}

```

```

        item1.setText(2, car.getMake());
        item1.setText(3, car.getModel());
        item1.setText(4, String.valueOf(car.getCarNumber()));
        item1.setText(5, car.getCarType().toString());
        item1.setText(6, String.valueOf(car.getMileage()));
        item1.setText(7, car.getColor());
        item1.setText(8, formatter.format(car.getDailyRentalFee()));
        item1.setText(9, formatter.format(car.calculateTax()));
        item1.setText(10, formatter.format(car.getTotalFee()));
        item1.setChecked(11, car.isAvailable());
    }
}

```

Running the example provides the result shown below (shown here in its contracted state).

**Figure 21. GridExample6**

Rental Grade	Automobile			Car Details	Pricing	Available
	Year	Make	Model	Car Number	Total	
Compact	2007	Chevy	Cobalt	133	\$131.15	<input type="checkbox"/>
	2007	Chevy	Cobalt	134	\$131.15	<input type="checkbox"/>
	2006	Ford	Focus	135	\$131.15	<input type="checkbox"/>
Mid-Sized						

## Section 16. The Nebula PGroup widget

The [Nebula PGroup widget](#), a subclass of the SWT Canvas, is a collapsible area that lets you group a number of SWT widgets together to form a meaningful collection of information for the user. This section details the usage of the PGroup widget. PGroup is currently in alpha status, and its API is subject to change.

### Create the PGroup and its styles

Listing 11 is a simple code example that creates a basic PGroup. The PGroup constructor takes a Composite parent object and an int signifying the style of the PGroup to be created.

In this case, you pass the default value of SWT.NONE. The other applicable style that can be passed to the constructor is SWT.SMOOTH, which rounds off the edges of the normally rectangular rendering of PGroup. You set the title of the PGroup by calling `setText()` and passing the appropriate value.

The example then creates a layout and a small number of widgets, which are added to the PGroup.

## Listing 11. PGroupExample1

```
public class PGroupExample1 {
    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout(new GridLayout());

        PGroup group = new PGroup(shell, SWT.NONE);
        group.setText("Owner Info");

        GridLayout gridLayout = new GridLayout();
        gridLayout.numColumns = 2;
        group.setLayout(gridLayout);
        GridData gridData = new GridData(GridData.HORIZONTAL_ALIGN_FILL);
        gridData.horizontalSpan = 3;
        group.setLayoutData(gridData);

        new Label(group, SWT.NULL).setText("Name:");
        final Text ownerName = new Text(group, SWT.SINGLE | SWT.BORDER);
        ownerName.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

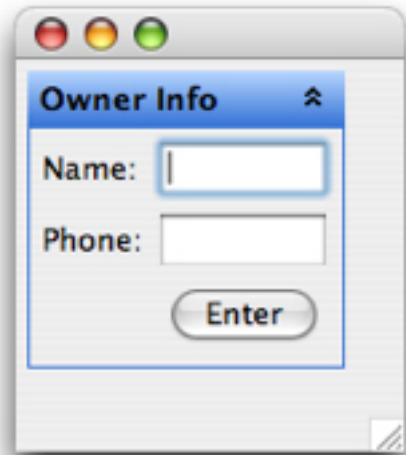
        new Label(group, SWT.NULL).setText("Phone:");
        final Text ownerPhone = new Text(group, SWT.SINGLE | SWT.BORDER);
        ownerPhone.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

        Button enter = new Button(group, SWT.PUSH);
        enter.setText("Enter");
        gridData = new GridData(GridData.HORIZONTAL_ALIGN_END);
        gridData.horizontalSpan = 3;
        enter.setLayoutData(gridData);

        shell.setSize(200, 200);
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

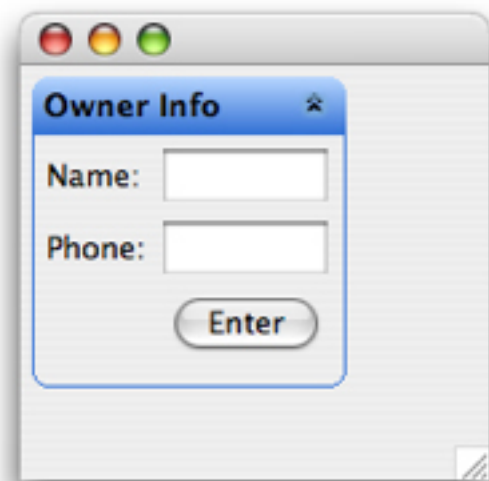
Running the example on Mac OS X provides the results shown below.

## Figure 22. PGroupExample1



Changing the style to `SWT.SMOOTH` when creating the `PGroup` changes the style slightly, as shown below (note the rounded edges of the `PGroup`).

**Figure 23. PGroupExample1 with a changed style**



## Change color

Changing the color of the background of the `PGroup` is straightforward. Call `setBackground()` on the group, giving it a display (or null) and an `SWT Color` object. Calling `setForeground()` on `PGroup` sets the color of the text in the `PGroup` Header bar. If there is a way to change the color of the header itself, we haven't been able to locate it.

Listing 12 adds the color changes described.

## Listing 12. PGroupExample2

```
public class PGroupExample2 {
    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout(new GridLayout());

        PGroup group = new PGroup(shell, SWT.NONE);

        // change text color in header to red
        group.setForeground(new Color(null, 255,0,0));

        // change the color of the background of the group
        group.setBackground(new Color(null, 111,111,0));

        group.setText("Owner Info");
        GridLayout gridLayout = new GridLayout();
        gridLayout.numColumns = 2;
        group.setLayout(gridLayout);
        GridData gridData = new GridData(GridData.HORIZONTAL_ALIGN_FILL);
        gridData.horizontalSpan = 3;
        group.setLayoutData(gridData);

        new Label(group, SWT.NULL).setText("Name:");
        final Text ownerName = new Text(group, SWT.SINGLE | SWT.BORDER);
        ownerName.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

        new Label(group, SWT.NULL).setText("Phone:");
        final Text ownerPhone = new Text(group, SWT.SINGLE | SWT.BORDER);
        ownerPhone.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

        Button enter = new Button(group, SWT.PUSH);
        enter.setText("Enter");
        gridData = new GridData(GridData.HORIZONTAL_ALIGN_END);
        gridData.horizontalSpan = 3;
        enter.setLayoutData(gridData);

        shell.setSize(200, 200);
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

Running the example provides the results shown below.

### Figure 24. PGroupExample2



## Change the toggle representation

You can change the representation of the toggle icon by changing the toggle renderer used by the PGroup. All toggle renderers must be a subclass of `org.eclipse.swt.nebula.widgets.pgroup.AbstractRenderer`. A few of these renderers are included in the PGroup JAR. These include `ChevronsToggleRenderer` (the default renderer used if one isn't set), `MinMaxToggleRenderer`, `TreeNodeToggleRenderer`, and `TwistedToggleRenderer`. Of course, you can create your own renderers if you wish. Changing the renderer is as simple as calling the `setToggleRenderer()` method on the PGroup instance and passing it an `AbstractRenderer` instance.

In addition to changing the toggle renderer, you can also designate whether the PGroup starts in expanded or contracted mode by calling the `setExpanded()` method on the PGroup instance and passing it the appropriate value.

Experiment with the code in Listing 13 by trying different toggle renderers.

### Listing 13. PGroupExample3

```
public class PGroupExample3 {
    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout(new GridLayout());

        PGroup group = new PGroup(shell, SWT.NONE);

        // start out expanded
        group.setExpanded(true); //default

        // Change Toggles
        // group.setToggleRenderer(new ChevronsToggleRenderer()); //default
        // group.setToggleRenderer(new MinMaxToggleRenderer());
        // group.setToggleRenderer(new TreeNodeToggleRenderer());
        // group.setToggleRenderer(new TwistedToggleRenderer());

        group.setLayout(new GridLayout());
    }
}
```

```

group.setText("Owner Info");
GridLayout gridLayout = new GridLayout();
gridLayout.numColumns = 2;
group.setLayout(gridLayout);
GridData gridData = new GridData(GridData.HORIZONTAL_ALIGN_FILL);
gridData.horizontalSpan = 3;
group.setLayoutData(gridData);

new Label(group, SWT.NULL).setText("Name:");
final Text ownerName = new Text(group, SWT.SINGLE | SWT.BORDER);
ownerName.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

new Label(group, SWT.NULL).setText("Phone:");
final Text ownerPhone = new Text(group, SWT.SINGLE | SWT.BORDER);
ownerPhone.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

Button enter = new Button(group, SWT.PUSH);
enter.setText("Enter");
gridData = new GridData(GridData.HORIZONTAL_ALIGN_END);
gridData.horizontalSpan = 3;
enter.setLayoutData(gridData);

shell.setSize(200, 200);
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}
}

```

Running the code with the MinMaxToggleRenderer gives the result shown below (with the PGroup contracted).

**Figure 25. PGroupExample3**

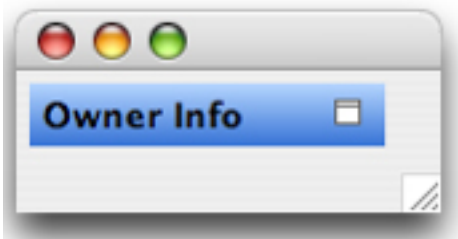


Figure 26 shows the result with the TreeNodeToggleRenderer.

**Figure 26. TreeNodeToggleRenderer**

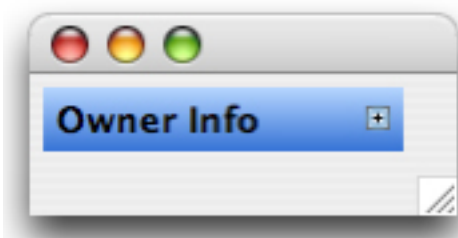
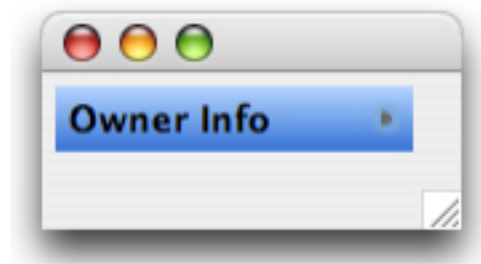


Figure 27 shows the result with the TwistedToggleRenderer.

**Figure 27. TwistedToggleRenderer****Change the group strategy**

In addition to changing the toggle renderer, you can choose to change the rendering strategy for the entire PGroup widget. All such rendering strategies must extend `org.eclipse.swt.nebula.widgets.pgroup.AbstractGroupStrategy`. A few of these strategies are included in the PGroup JAR: `RectangleGroupStrategy` (the default if no strategy is set), `SimpleGroupStrategy`, and `FormGroupStrategy`. Each changes the way the PGroup object is painted and sized. You can set a new rendering strategy on the PGroup instance by calling `setStrategy()` and passing an `AbstractGroupStrategy` instance.

The code in Listing 14 changes the rendering strategy to use a `SimpleGroupStrategy`.

**Listing 14. PGroupExample4**

```
public class PGroupExample4 {
    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout(new GridLayout());

        PGroup group = new PGroup(shell, SWT.NONE);

        // Optionally, change strategy
        // group.setStrategy(new RectangleGroupStrategy()); //default
        // group.setStrategy(new FormGroupStrategy());
        group.setStrategy(new SimpleGroupStrategy());

        group.setText("Owner Info");
        GridLayout gridLayout = new GridLayout();
        gridLayout.numColumns = 2;
        group.setLayout(gridLayout);
        GridData gridData = new GridData(GridData.HORIZONTAL_ALIGN_FILL);
        gridData.horizontalSpan = 3;
        group.setLayoutData(gridData);

        new Label(group, SWT.NULL).setText("Name:");
        final Text ownerName = new Text(group, SWT.SINGLE | SWT.BORDER);
        ownerName.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

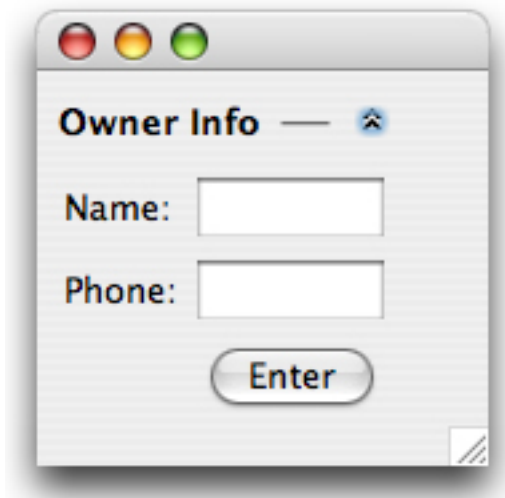
        new Label(group, SWT.NULL).setText("Phone:");
        final Text ownerPhone = new Text(group, SWT.SINGLE | SWT.BORDER);
        ownerPhone.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
    }
}
```

```
Button enter = new Button(group, SWT.PUSH);
enter.setText("Enter");
gridData = new GridData(GridData.HORIZONTAL_ALIGN_END);
gridData.horizontalSpan = 3;
enter.setLayoutData(gridData);

shell.setSize(200, 200);
shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}
```

Figure 28 shows the results of this code on Mac OS X.

**Figure 28. PGroupExample4**



---

## Section 17. The Nebula PShelf widget

The [Nebula PShelf widget](#) follows a selectable-accordion metaphor. Each selectable area or shelf can contain a Composite client area you can hide by selecting one of the other shelves. This section details the usage of the PShelf widget, which is currently in an alpha release status. Its API is, therefore, subject to change.

### A PShelf example

Creating a PShelf is straightforward: call the constructor, and supply a Composite parent object and an SWT style, represented by an `int` value. Each shelf is represented by a `PShelfItem` object. A reference to the parent PShelf must be

supplied to the PShelfItem constructor. Calling the `getBody()` method on the PShelfItem returns a Composite object that functions as the client area. A Layout is set, and you add other SWT widgets to this Composite area in the standard way.

The example in Listing 15 creates a PShelf with three shelves: one containing a Nebula Grid; one containing a Text object; and one containing some labels, some text fields, and a button.

### Listing 15. PShelfExample1

```
public class PShelfExample1 {
    public static void main (String [] args) {
        Display display = new Display ();
        Shell shell = new Shell (display);
        shell.setLayout(new FillLayout());

        PShelf shelf = new PShelf(shell, SWT.SIMPLE);

        PShelfItem item1 = new PShelfItem(shelf,SWT.NONE);
        item1.setText("First Item");
        item1.getBody().setLayout(new FillLayout());

        Grid grid = new Grid(item1.getBody(),
            SWT.BORDER | SWT.V_SCROLL | SWT.H_SCROLL);
        grid.setHeaderVisible(true);

        GridColumn column1 = new GridColumn(grid,SWT.NONE);
        column1.setText("Column 1");
        column1.setWidth(150);
        GridColumn column2 = new GridColumn(grid,SWT.NONE);
        column2.setText("Column 2");
        column2.setWidth(150);

        GridItem gridItem1 = new GridItem(grid,SWT.NONE);
        gridItem1.setText(0, "First Item");
        gridItem1.setText(1, "Some data");
        GridItem gridItem2 = new GridItem(grid,SWT.NONE);
        gridItem2.setText(0, "Second Item");
        gridItem2.setText(1, "Some other data");
        GridItem gridItem3 = new GridItem(grid,SWT.NONE);
        gridItem3.setText(0, "Third Item");
        gridItem3.setText(1, "Even more data");

        PShelfItem item2 = new PShelfItem(shelf,SWT.NONE);
        item2.setText("Second Item");
        item2.getBody().setLayout(new GridLayout());
        Text text = new Text(item2.getBody(),SWT.WRAP);
        text.setText("Blah blah blah");

        PShelfItem item3 = new PShelfItem(shelf, SWT.NONE);
        item3.setText("Third Item");

        GridLayout gridLayout = new GridLayout();
        gridLayout.numColumns = 2;
        item3.getBody().setLayout(gridLayout);
        GridData gridData = new GridData(GridData.HORIZONTAL_ALIGN_FILL);
        gridData.horizontalSpan = 3;
        item3.getBody().setLayoutData(gridData);

        new Label(item3.getBody(), SWT.NULL).setText("Name:");
    }
}
```

```

final Text ownerName =
    new Text(item3.getBody(), SWT.SINGLE | SWT.BORDER);
ownerName.setLayoutData(
    new GridData(GridData.FILL_HORIZONTAL));

    new Label(item3.getBody(), SWT.NULL).setText("Phone:");

final Text ownerPhone =
    new Text(item3.getBody(), SWT.SINGLE | SWT.BORDER);
ownerPhone.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

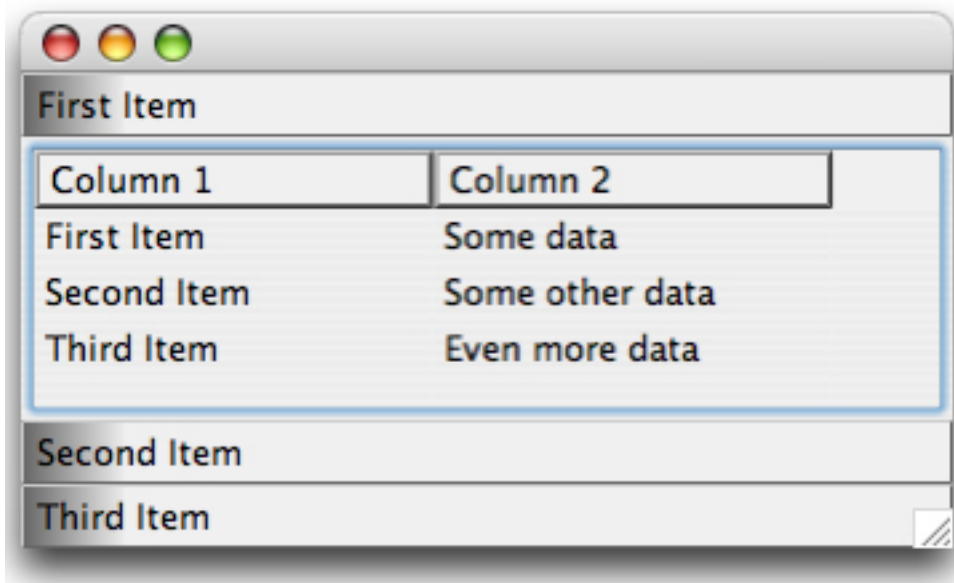
Button enter = new Button(item3.getBody(), SWT.PUSH);
enter.setText("Enter");
gridData = new GridData(GridData.HORIZONTAL_ALIGN_END);
gridData.horizontalSpan = 3;
enter.setLayoutData(gridData);

shell.setSize(350,200);
shell.open ();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch ()) display.sleep ();
}
display.dispose ();
}
}

```

Running the example in Mac OS X yields the result shown below.

**Figure 29. PShelfExample1**



### Change the PShelf renderer

You can choose to change the renderer that realizes the PShelf. All such renderers must be subclasses of `org.eclipse.swt.nebula.widgets.pshelf.AbstractRenderer`. Two renderers -- `PaletteShelfRenderer` (the default) and `RedmondShelfRenderer` -- are supplied in the PShelf JAR file. You can set them on the PShelf instance by calling the `setRenderer()` method and providing an `AbstractRenderer` instance.

## Change colors

There are a number of ways you can specify color for the PShelf and the widgets it contains. To specify the background color for the selection areas of the PShelf, call the `setBackground()` method on the PShelf instance and provide a `Color` object. The `setForeground()` method changes the color of the text in these selection areas. You can set the color of each client area and the widgets in it using the usual `setBackground()` and `setForeground()` methods on the appropriate widget.

In addition, each supplied renderer has differing methods you can use to specify its color choices. For instance, the `PaletteShelfRenderer` supplies a `setShadeColor()` method that designates the color used in its shading gradient.

The `RedmondShelfRenderer` has quite a few methods you can use to designate how it uses color. You can set colors and color gradients for a shelf that is selected, not selected, or hovered over. Listing 16 uses a `RedmondShelfRenderer` and changes colors in various ways across the example. (Please note that these colors aren't particularly good choices, but they illustrate the concepts at hand.)

### Listing 16. PShelfExample2

```
public class PShelfExample2 {
    public static void main (String [] args) {
        Display display = new Display ();
        Shell shell = new Shell (display);
        shell.setLayout(new FillLayout());

        PShelf shelf = new PShelf(shell, SWT.SIMPLE);

        RedmondShelfRenderer renderer = new RedmondShelfRenderer();
        shelf.setRenderer(renderer);
        renderer.setFont(new Font(null, "Arial", 16, SWT.ITALIC));
        renderer.setSelectedFont(new Font(null, "Arial", 22, SWT.BOLD));

        //set foreground color to white:
        shelf.setForeground(new Color(null, 255, 255, 255));
        //set coloring gradient on shelf header
        renderer.setGradient1(new Color(null, 0, 0, 255));
        renderer.setGradient2(new Color(null, 0, 0, 175));

        //set a hover color
        renderer.setHoverGradient1(new Color(null, 255, 0, 0));
        renderer.setHoverGradient1(new Color(null, 110, 0, 0));

        //set a selected color
        renderer.setSelectedGradient1(new Color(null, 0, 255, 0));
        renderer.setSelectedGradient2(new Color(null, 0, 110, 0));

        PShelfItem item1 = new PShelfItem(shelf, SWT.NONE);
        item1.setText("First Item");
        item1.getBody().setLayout(new FillLayout());

        Grid grid = new Grid(item1.getBody(),
            SWT.BORDER | SWT.V_SCROLL | SWT.H_SCROLL);
        grid.setHeaderVisible(true);

        GridColumn column1 = new GridColumn(grid, SWT.NONE);
        column1.setText("Column 1");
    }
}
```

```

column1.setWidth(150);
GridColumn column2 = new GridColumn(grid,SWT.NONE);
column2.setText("Column 2");
column2.setWidth(150);

GridItem gridItem1 = new GridItem(grid,SWT.NONE);
gridItem1.setText(0, "First Item");
gridItem1.setText(1, "Some data");
GridItem gridItem2 = new GridItem(grid,SWT.NONE);
gridItem2.setText(0, "Second Item");
gridItem2.setText(1, "Some other data");
GridItem gridItem3 = new GridItem(grid,SWT.NONE);
gridItem3.setText(0, "Third Item");
gridItem3.setText(1, "Even more data");

PShelfItem item2 = new PShelfItem(shelf,SWT.NONE);
item2.setText("Second Item");
item2.getBody().setLayout(new GridLayout());
// Set background color of the body of the shelf
item2.getBody().setBackground(new Color(null, 110, 0, 0));

Text text = new Text(item2.getBody(),SWT.WRAP);
text.setText("Blah blah blah");
//set Background color on the text widget...
text.setBackground(new Color(null, 0,0,110));

PShelfItem item3 = new PShelfItem(shelf, SWT.NONE);
item3.setText("Third Item");

    GridLayout gridLayout = new GridLayout();
    gridLayout.numColumns = 2;
    item3.getBody().setLayout(gridLayout);
    GridData gridData = new GridData(GridData.HORIZONTAL_ALIGN_FILL);
    gridData.horizontalSpan = 3;
    item3.getBody().setLayoutData(gridData);

    new Label(item3.getBody(), SWT.NULL).setText("Name:");
final Text ownerName =
    new Text(item3.getBody(), SWT.SINGLE | SWT.BORDER);
    ownerName.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

    new Label(item3.getBody(), SWT.NULL).setText("Phone:");
final Text ownerPhone =
    new Text(item3.getBody(), SWT.SINGLE | SWT.BORDER);
    ownerPhone.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

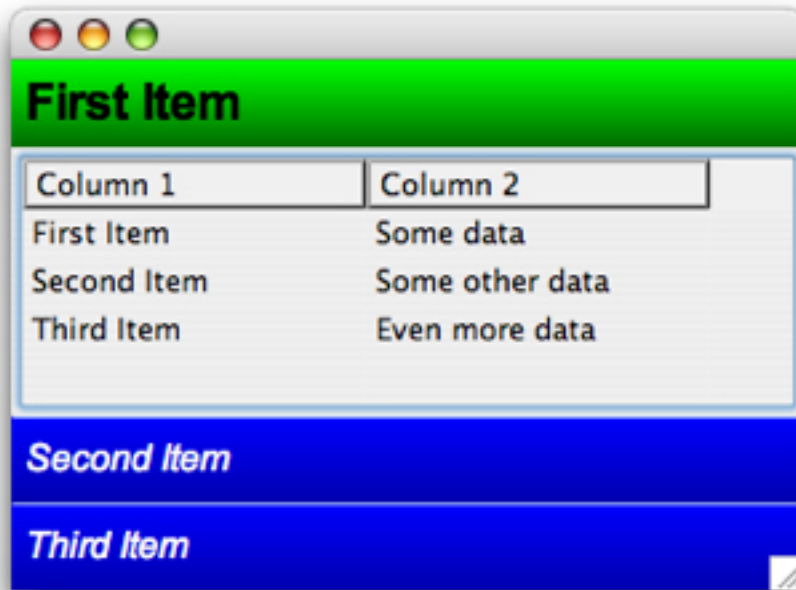
    Button enter = new Button(item3.getBody(), SWT.PUSH);
    enter.setText("Enter");
    gridData = new GridData(GridData.HORIZONTAL_ALIGN_END);
    gridData.horizontalSpan = 3;
    enter.setLayoutData(gridData);

shell.setSize(350,200);
shell.open ();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch ()) display.sleep ();
}
display.dispose ();
}
}

```

Running the example yields the result shown in Figure 30.

### Figure 30. PShelfExample2



---

## Section 18. Conclusion

The Standard Widget Toolkit (SWT) provides access to the native widgets of an operating system via Java technology, but the widgets available don't solve every need. The Eclipse Nebula project provides nine widgets to help answer common UI programming needs. This tutorial has demonstrated five of Nebula's nine widgets, including Grid, CDateTime, CompositeTable, PGroup, and PShelf. SWT includes support for advanced widgets, such as date selectors, shelves, and groups. Using the Composite Table and Grid, you can also generate complicated table layouts. Check out the [Resources](#) for more information.

# Resources

## Learn

- Start at the [Nebula Project: Supplemental Custom Widgets for SWT](#) home at eclipse.org to learn more about Nebula.
- Read the original SWT project proposal [Supplemental Widgets for SWT \(Nebula\)](#) document.
- Learn about creating SWT applications in the developerWorks article series "[A gentle introduction to SWT and JFace.](#)"
- Learn about migrating legacy Swing applications to the higher-performing SWT platform in "[Migrate your Swing application to SWT.](#)"
- Examine the strengths and weaknesses of SWT, Swing, or AWT in "[SWT, Swing, or AWT: Which is right for you?](#)"
- For an introduction to the Eclipse platform, see "[Getting started with the Eclipse Platform.](#)"
- Check out the "[Recommended Eclipse reading list.](#)"
- Browse all the [Eclipse content](#) on developerWorks.
- Expand your Eclipse skills by checking out IBM developerWorks' [Eclipse project resources](#).
- To listen to interesting interviews and discussions for software developers, check out [developerWorks podcasts](#).
- Stay current with developerWorks' [Technical events and webcasts](#).
- Check out upcoming conferences, trade shows, webcasts, and other [Events](#) around the world that are of interest to IBM open source developers.
- Visit the developerWorks [Open source zone](#) for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

## Get products and technologies

- Download [CDateTime](#) from Eclipse.org.
- Download the [SWTCalendar](#) from SourceForge.net.
- Download the [JPopupCalendar](#) from SourceForge.net and read the [installation instructions](#).
- Download the [Jaret Date Chooser](#).
- Download the [Eclipse Platform](#) and get started with Eclipse now.

- Check out the latest [Eclipse technology downloads](#) at IBM [alphaWorks](#).
- Innovate your next open source development project with [IBM trial software](#), available for download or on DVD.

## Discuss

- The [Eclipse Platform newsgroups](#) should be your first stop to discuss questions regarding Eclipse. (Selecting this will launch your default Usenet news reader application and open eclipse.platform.)
- The [Eclipse newsgroups](#) has many resources for people interested in using and extending Eclipse.
- Participate in [developerWorks blogs](#) and get involved in the developerWorks community.

## About the authors

### Scott Delap



Scott Delap is president of Rich Client Solutions Inc., a software consulting firm focusing on technologies such as Swing, Eclipse RCP, GWT, Flex, and Open Laszlo. He is actively involved in the Java community, speaking at events such as NFJS, QCon and JavaOne. He is also the Java editor of InfoQ.com and runs ClientJava.com, a portal focused on desktop Java development.

---

### Barry Livingston

Barry Livingston has been developing software for more than eight years. A specialist in rich client development, he is focused on client technologies such as Swing and Eclipse RCP. Active in the Java community, he has often partnered with Rich Client Solutions to provide high-quality rich client development and consulting services to a variety of clients.