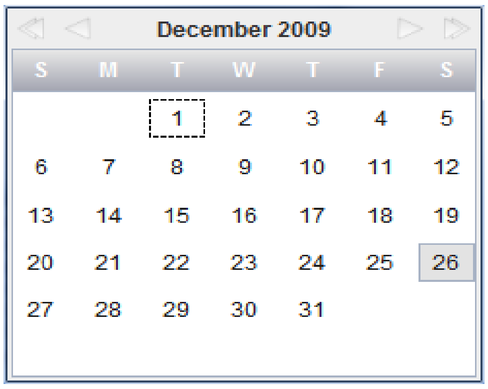
Custom Swing Components



Calendar

V1.2

Table of Contents

1 Document Version

Version	*Date*	*Author*
1.0	26th August 2010	R. Liebowitz
1.1	4th October 2010	R. Liebowitz

2 Release Notes

Changes in version 1.2

- Component now depends on version 1.2 of framework.
  - Improved clipping behaviour.
  - Enhancements to painters.
  - Added additional shape providers: Star, Circle and Square
  - Created the default gloss for java swing components.

3 Licenses

By purchasing a component from Custom Swing Components, the purchaser has agreed to abide by Custom Swing Components' license agreement including its terms and conditions. The details of which are repeated below.

3.1 Custom Swing Components

The terms and conditions appear on the following page and are included as part of the license.

Terms
The purchase of a license for a component from Custom Swing Components entitles you to the following:

You may use the component on as many projects as you desire. There are no restrictions regarding the number of deployments, and you may bundle the component as part of your software, be it commercial or personal.

Once you have purchased a component, there is no limit on how many times you may download the Java archive or its supporting documentation.

3 months free email support for the purchased component. Additional support will require the purchase of an extended support contract.

Receive bug fixes and updates for the version of the component purchased. Note that this only includes increments of the component's minor version. To illustrate this, if you purchase version 1.0 of a specific component, you are entitled to all future minor updates (i.e. 1.1, 1.2 ... 1.n).

In the event that a major version is released within 3 months of you purchasing the previous version, you will be automatically entitled to the new version. To illustrate this, if you purchase version 1.0 of a specific component, you are entitled to version 2.0 for free, if version 2.0 is released within 3 months of your purchase of version 1.0.

Restrictions
Restrictions apply to all types of licenses:

You may not directly resell licensed component to other individuals or entities. To illustrate this, you may not sell the Java archive to third parties. Please note that this does not restrict you from including the component in commercial software; it prevents you directly selling the archive to other third parties.

If the deployment of your software directly exposes the API of the component to third party developers, there may be an additional deployment fee. To illustrate this, if you sell a product whose primary target is developers, they will gain access to the licensed component and be able to use it in their own software. Please note that this does not restrict you from including the component in commercial software; it is intended to prevent other third party developers from making use of components that they have not purchased.

A license may not be automatically transferred. An enterprise license is granted to a named enterprise and may not be transferred to another enterprise. A developer license is granted to a named developer and may not be transferred to another developer. Custom Swing Components does not support a floating license. Please contact us directly if you need to transfer a license.

License Types
At present there are 3 types of licenses available: developer, enterprise and source code.

A developer license entitles a single named developer to make use of the licensed components.

An enterprise license entitles all developers within the enterprise to make use of the licensed components.

A source code license is available and applies on an enterprise scale; please contact us for more details.

4 Dependencies

The Calendar is dependent on a number of additional Java archives (jars). These archives must be present on the classpath for the component to function. The dependent archives have been bundled in the same zip file as this document.

The dependencies are located in the 'bin' folder and have been assembled into two sub directories, '*aggregated*' and '*separate*'.

- The '*aggregated*' folder contains a single java archive which includes the Calendar as well as all its dependent archives.
- The '*separate*' folder contains the Calendar and each of its dependent archives in separate files.

Dependency	*Filename*	*Version*	*Purpose*
Custom Swing Components Framework	framework-1.2.jar	1.2	This contains infrastructure and utilities required by all Custom Swing Components.

5 Calendar Usage Guide

The Calendar is a powerful component intended to allow the user to select either a single or multiple dates.

The code snippets below should cover the most common uses of the component, however for a more in depth explanation of the API it is recommended that you consult the javadocs. All the code snippets included are part of an example that can be found in the zip file.

5.1 How to launch a swing application in thread safe manner

All interaction with the Calendar should occur on the Event Dispatch Thread (EDT), in order to enforce thread safety and prevent errors.

It is recommended that all swing applications be launched using the SwingUtilities class to ensure that right from the start, your application is interacting with the EDT. The code below will launch a JFrame in a thread safe manner.

```
/**
 * This is the correct way to launch a swing application.
 * @param args
 */
public static void main(String[] args) {
      SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                  CodeExamples codeExample = new CodeExamples();
                  frame = new JFrame();
                  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                  Container panel = frame.getContentPane();
                  panel.setLayout(new BorderLayout());
                  panel.add(codeExample, BorderLayout.CENTER);
                  frame.pack();
                  frame.setSize(500,300);
                  frame.setVisible(true);
            }
      });
}
```

5.2 How to create a Calendar

Whenever you create any type of component that is internationalizable or localizable (such as formatted text field, Date Picker a Calendar) it is of the utmost importance that you are aware of the locale and timezone applied to your component. It is for this reason that it is preferable to make use of a constructor that receives a time zone and a locale as parameters. This leaves nothing to chance.

If you make use of the default empty constructor, your Calendar will be constructed using your system's locale, system's time zone and a Calendar model with no business logic. The code below constructs a new Calendar and passes the locale and time zone to the component.

```java
/**
 * Although you can create a calendar without supplying parameters to
 * the constructor this is not ideal. It is far better to specify the exact
 * timezone and locale for the calendar.
 * For the purpose of this example I will use the System default timezone
 * and locale.
 * @return
 */
private JSCCalendar howToCreateACalendar() {
      TimeZone timeZone = TimeZone.getDefault();
      Locale locale = Locale.getDefault();
      return new JSCCalendar(timeZone, locale);
}
```

If you wish to change the locale or time zone at a later stage, you should use the available setter methods.

```java
/**
 * Simply call the setter methods on the calendar to change the timezone
 * and locale.
 */
private void howToChangeTheLocaleAndTimezone(JSCCalendar calendar) {
      calendar.setLocale(Locale.UK);
      calendar.setTimeZone(TimeZone.getTimeZone("GMT"));
}
```

5.3 How to change the UI (skin) of the Calendar

The Calendar ships with three skins that can be applied to the Calendar:

- SteelCalendarUI (Default)
- DarkSteelCalendarUI
- BasicCalendarUI

Changing to a new skin is very easy as the code below demonstrates.

```
/**
 * You can change the look and feel of the component by changing its ui.
 * In this example we will change the UI to the DarkSteelUI
 * @param calendar the calendar
 */
private void howToChangeTheLookAndFeel(JSCCalendar calendar) {
      //We create a new instance of the UI
      DarkSteelCalendarUI newUI = (DarkSteelCalendarUI)
                                  DarkSteelCalendarUI.createUI(calendar);
      //We set the UI
      calendar.setUI(newUI);
}
```

5.4 How to move the Calendar to a different date

If you wish to move the Calendar to display a specific month and year one need only call the setDisplayDate method on the Calendar's model. Calling this method will update the Calendar's UI to render the new date. Please note however that this will not force the Calendar to select the date, only display it.

The code below will create a new date of December 2009, using a regular java.util.Calendar and then apply the date to the Calendar component. Note that the java.util.Calendar makes use of the time zone and locale from the Calendar component, this ensures that the date supplied from the Calendar will have the correct time zone information.

```
/**
 * You can change the rendered date of the calendar by calling the
 * setDisplayDate method on the calendarModel.
 * @param calendar
 */
private void howToMoveToASpecificDate(JSCCalendar jscCalendar) {
      //lets make the jsccalendar move to December 2009
      Calendar calendar = Calendar.getInstance(jscCalendar.getTimeZone(),
                          jscCalendar.getLocale());
      calendar.set(Calendar.MONTH, Calendar.DECEMBER);
      calendar.set(Calendar.YEAR, 2009);
      Date december2009 = calendar.getTime();

      jscCalendar.getCalendarModel().setDisplayDate(december2009);
}
```

5.5 How to add business rules to your Calendar

The business logic relating to what dates are selectable and what dates are not are stored in the Calendar's model and not in the Calendar itself. This separation of logic allows developers to customise which dates are selectable and which dates are not.

When a new Calendar is constructed, internally it creates a DefaultCalendarModel to ensure that it is able to function. This model however contains no business rules and therefore will not restrict the users selection of dates.

There are two methods available for developers when it comes to applying business rules. A developer may choose to either create an entirely new Calendar model, or override the isDateSelectable method on the DefaultCalendarModel.

For the purpose of this example we will create an entirely new Calendar model as this offers a better learning experience and gives the reader a greater insight into the responsibilities of the Calendar model. The code below will create a new model that does not allow the user to select a date that falls on a holiday or on a weekend.

```
/**
 * The CalendarModel is used to apply custom business rules to the calendar.
 * The CalendarModel is the location where your business rules should live.
 * It is also responsible for creating the text used in the calendar's cells
 * @param calendar
 */
private void howToAddBusinessRules(JSCCalendar calendar) {

AbstractCalendarModel newRules = new AbstractCalendarModel() {

@Override
public boolean isDateSelectable(Date date) {
      //this is the method that determines if a day may be selected.
      //Your business logic should live here.

      //so we return true if the date is not a holiday and is not a weekend.
      return !isDateHoliday(date) && !isDateWeekend(date);
}

@Override
public String getTextForHeading(DayOfWeek dayOfWeek) {
      //this will return the first letter of the day of the week for the heading
      Character firstLetter =
                        dayOfWeek.getDisplayChar(getCalendar().getLocale());
      return firstLetter.toString().toUpperCase();
}

@Override
public String getTextForCell(Date date) {
      //this will return the day of the month for each cell
      Calendar calendar = createCalendar(date);
      return Integer.toString(calendar.get(Calendar.DAY_OF_MONTH));
}
@Override
public boolean isDateSelected(Date date) {
      //please note the utility method areDatesEqual,
      //it compares two dates to see if they occur on the
      //same day.
      for (Date selectedDate: getSelectedDates()) {
            if (areDatesEqual(selectedDate, date)) {
                  return true;
            }
      }
      return false;
}

@Override
public boolean isDateHoliday(Date date) {
      //please note the utility method areDatesEqual,
      //it compares two dates to see if they occur on the
      //same day.
      for (Holiday holiday: getHolidays()) {
            if (areDatesEqual(holiday.getDate(), date)) {
                  return true;
            }
      }
      return false;
}

@Override
public Holiday getHolidayForDate(Date date) {
      //get the holiday object for the supplied date.
      for (Holiday holiday: getHolidays()) {
            if (areDatesEqual(holiday.getDate(), date)) {
                  return holiday;
            }
      }
      return null;
}

@Override
public boolean isDateWeekend(Date date) {
      Calendar calendar = createCalendar(date);
      DayOfWeek dayOfWeek =
            DayOfWeek.getDayOfWeek(calendar.get(Calendar.DAY_OF_WEEK));
      return getWeekendDays().contains(dayOfWeek);
}
};

//apply the calendarModel with the new rules to the calendar.
calendar.setCalendarModel(newRules);
```

You will notice in the code above that the model is also responsible for determining what text should appear in the heading and cells of the Calendar. The code above will return the first letter of the day of the week for each of the heading cells, and the numeric day of the week for the other cells.

The Calendar model is also responsible for keeping track of holidays.

5.6 How to listen to changes on the Calendar

If the developer needs to be informed of the changes occurring on the Calendar's model, such as when a new date is selected or removed, he may do so by attaching a CalendarSelectionListener to the Calendar's model.

```
/**
 * You listen to changes on the calendar by using a CalendarSelectionListener
 * @param calendar
 */
private void howToListenToChangesOnTheCalendar(JSCCalendar calendar) {
      calendar.addCalendarSelectionListener(new CalendarSelectionListener() {

@Override
public void selectedDatesChanged(CalendarSelectionEvent calendarSelectionEvent) {
      //the calendar that has changed.
      JSCCalendar calendar = calendarSelectionEvent.getCalendar();

      //the selected dates the calendar holds.
      List<Date> selectedDates = calendarSelectionEvent.getSelectedDates();

      //the type of event that has occurred.
      CalendarSelectionEventType selectionEventType =
                        calendarSelectionEvent.getCalendarSelectionEventType();

      switch (selectionEventType) {
      case DATE_REMOVED: {
            //put your logic here to react to a date being removed.
            System.out.println("A date has been removed");
            break;
      }
      case DATE_SELECTED: {
            //put your logic here to react to a date being selected/added
            System.out.println("A date has been selected");
            break;
      }
      case DATES_SELECTED: {
            //put your logic here to react to multiple dates being selected/added
            System.out.println("Dates have been selected");
            break;
      }
      case DATES_CLEARED: {
            //put your logic here to react to all dates being removed
            System.out.println("All dates have been cleared");
            break;
      }
      case DISPLAY_DATE_CHANGED: {
            //put your logic here to react to the calendar displaying a new date
            System.out.println("Display date moved");
            break;
            }
      }
      }
});
}
```

5.7 How to change the appearance of the cells in the Calendar

The logic that determines how the cells in a Calendar are rendered resides in the CalendarCellRenderer. All 3 UI's come with their own implementations of CalendarCellRenderer. If you wish to change the appearance of the cells or headings, you can either:

- Tweak the fields of an existing cell renderer using the available getter and setter methods.
- Override the methods getHeadingCellRendererComponent or getCellRendererComponent.
- Create your own CalendarCellRenderer implementation.

The code below will create an entirely new CalendarCellRenderer that will render an icon for Christmas rather than render a normal number (The icon is loaded during the CellRenderers constructor and not each time the renderer is painted). All the information required to correctly render your cell will be passed into the method via its parameters.

```
/**
 * The implementation of CalendarCellRenderer returns a component which is rendered
 * in the calendar.
 * I recommend that your CalendarCellRenderer extends JLabel and returns itself.
 * See the standard swing TableCellRenderer for more information on how
 * cellRenderers work.
 * @param calendar
 */
private void howToChangeTheAppearanceOfTheCells(JSCCalendar calendar) {
      calendar.setCalendarCellRenderer(new CustomCellRenderer());
}

private class CustomCellRenderer extends JLabel implements CalendarCellRenderer {

private Icon xmasIcon;

public CustomCellRenderer() {
      Image image;
      try {
            image =
 GraphicsUtilities.loadCompatibleImage(Thread.currentThread().getContextClassLoader(
).getResourceAsStream("demo.png"));
            xmasIcon = new ImageIcon(image);
      } catch (IOException e) {
            e.printStackTrace();
      }
}
@Override
public JComponent getHeadingCellRendererComponent(JSCCalendar calendar,String text)
{
      //configure your customCellRenderer based on the supplied information, ie
      //String text
      setHorizontalAlignment(JLabel.CENTER);
      setText(text);
      setOpaque(false);
      setBorder(BorderFactory.createEmptyBorder(0,0,0,0));
      setForeground(Color.BLACK);
      setIcon(null);
      //our headings will have a transparent background and black foreground
      return this;
}
```

```
@Override
public JComponent getCellRendererComponent(CellRendererComponentParameter
                  parameterObject) {
      //configure your customerCellRenderer based on the information encapsulated in
      //the parameterObject.
      //you must choose how to render your component based on your business rules
      //and the following parameters.

      //is your current cell that you are rendering a holiday
      boolean isHoliday = parameterObject.isHoliday;
      //is your current cell that you are rendering a weekend
      boolean isWeekend = parameterObject.isWeekend;
      //is the mouse over tthe current cell you are rendering
      boolean isMouseOver = parameterObject.isMouseOver;
      //is your current cell that you are rendering today
      boolean isToday = parameterObject.isToday;
      //is your current cell that you are rendering already selected
      boolean isSelected = parameterObject.isSelected;
      //is your current cell that you are rendering able to be selected
      boolean isSelectable = parameterObject.isAllowSelection();
      //is your current cell that you are rendering currently the keyboard focus cell
      boolean hasKeyboardFocus = parameterObject.isHasFocus();
      //is your current cell that you are rendering in this current month
      boolean isCurrentMonth = parameterObject.isCurrentMonth;
      //the text of the cell
      String text = parameterObject.getText();
      //the date of the cell
      Date date = parameterObject.getDate();
      //the calendar
      JSCCalendar calendar = parameterObject.getCalendar();


      //for this example all dates will render the same except for the unselectable
      //dates. We defined the unselectable dates in the calendarModel as weekends and
      //holidays.

      setHorizontalAlignment(JLabel.CENTER);
      setIcon(null);
      setText(text);
      setOpaque(false);

      if (isSelectable) {
            setForeground(Color.BLACK);
      } else {
            setText(text);
            setForeground(Color.LIGHT_GRAY);
      }

      //if we are rendering the month of December, some of the cell in the calendar
      //can relate to days in January and December. in this example we will not draw
      //them
      if (!isCurrentMonth) {
            //removing the text will make them appear empty
            setText("");
      }
      //if its Christmas, lets use a festive icon
      if (isHoliday) {
            Holiday holiday = calendar.getCalendarModel().getHolidayForDate(date);
            if (holiday.getDescription().contains("Christmas")) {
                  setIcon(xmasIcon);
                  setText("");
            }
      }

      //selected dates will receive a black border
      if (isSelected) {
            setBorder(BorderFactory.createLineBorder(Color.BLACK));
      } else {
            setBorder(BorderFactory.createEmptyBorder(0,0,0,0));
      }

      //the cell which has keyboard focus will receive a gray border
      if (hasKeyboardFocus) {
            setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
      }

      return this;
      }
};
```

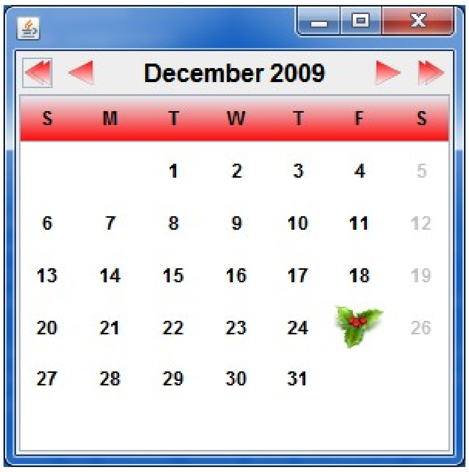
5.8 Customizing the UI itself

If you are not able to achieve the desired result by manipulating an existing CalendarCellRender, or by creating your own. You can always customize the Calendar by altering the settings on the Calendar's UI directly.

The code below will show you how to access the Calendar's UI and tweak some of the internal components.

```
/**
 * If you are still not happy with the level of customisation the
 * CalendarCellRenderer
 * gives you, you may change the UI directly. Each UI will have different properties
 * It is recommended that you check the javadoc for each ui to discover what getters
 * and setters are available.
 * In this example will will tweak the DarkSteelCalendarUI
 * @param calendar
 */
private void howToChangeTheAppearanceOfTheOverallUI(JSCCalendar calendar) {
      //All current calendarUI's are subclasses of of StandardFormatCalendarUI.
      //A StandardFormatCalendarUI comprises the following pieces
      //1. A month and year panel located at the top of the component
      //2. A cell panel located below the month and year panel containing the cells
      //3. The cell panel contains a background painter
      //4. The month and year panel contains a background painter
      //Each of the above classes including the StandardFormatCalendarUI
      //can be tweaked or replaced to create your ideal calendar.

      DarkSteelCalendarUI calendarUI = (DarkSteelCalendarUI) calendar.getUI();
      DarkSteelMonthAndYearPanel monthAndYearPanel = (DarkSteelMonthAndYearPanel)
            calendarUI.getMonthAndYearPanel();
      DarkSteelCellPanel cellPanel = (DarkSteelCellPanel) calendarUI.getCellPanel();
      DarkSteelCellPanelBackgroundPainter cellPanelBackgroundPainter =
            (DarkSteelCellPanelBackgroundPainter) cellPanel.getBackgroundPainter();

      //The DarkSteelCalendarUI does not make use of a backgroundPainter on the
      //monthAndYearPanel.
      //that is why the blankPainter was assigned. However any painter could be
      // applied in its place.
      BlankPainter blankPainter = (BlankPainter)
            monthAndYearPanel.getBackgroundPainter();

      //we will change the font on the month and year panel to black, bold and
      //slightly larger
      Font newFont = monthAndYearPanel.getDateLabel().getFont()
            .deriveFont(Font.BOLD).deriveFont(16f);
      monthAndYearPanel.getDateLabel().setForeground(Color.BLACK);
      monthAndYearPanel.getDateLabel().setFont(newFont);
      //these images will replace the standard images used for the buttons with
      //larger red images
      ImageIcon nextMonth = new ImageIcon(loadImage("demoNextMonth.png"));
      ImageIcon nextYear = new ImageIcon(loadImage("demoNextYear.png"));
      ImageIcon prevMonth = new ImageIcon(loadImage("demoPrevMonth.png"));
      ImageIcon prevYear = new ImageIcon(loadImage("demoPrevYear.png"));

      ImageIcon rolloverNextMonth = new
            ImageIcon(loadImage("demoRolloverNextMonth.png"));
      ImageIcon rolloverNextYear = new
            ImageIcon(loadImage("demoRolloverNextYear.png"));
      ImageIcon rolloverPrevMonth = new
            ImageIcon(loadImage("demoRolloverPrevMonth.png"));
      ImageIcon rolloverPrevYear = new
            ImageIcon(loadImage("demoRolloverPrevYear.png"));

      monthAndYearPanel.getIncrementMonthButton().setIcon(nextMonth);
      monthAndYearPanel.getIncrementYearButton().setIcon(nextYear);
      monthAndYearPanel.getDecrementMonthButton().setIcon(prevMonth);
      monthAndYearPanel.getDecrementYearButton().setIcon(prevYear);


      monthAndYearPanel.getIncrementMonthButton().setRolloverIcon(rolloverNextMonth);
      monthAndYearPanel.getIncrementYearButton().setRolloverIcon(rolloverNextYear);
      monthAndYearPanel.getDecrementMonthButton().setRolloverIcon(rolloverPrevMonth);
      monthAndYearPanel.getDecrementYearButton().setRolloverIcon(rolloverPrevYear);

      //we will remove the selectedIcon
      monthAndYearPanel.getIncrementMonthButton().setPressedIcon(null);
      monthAndYearPanel.getIncrementYearButton().setPressedIcon(null);
      monthAndYearPanel.getDecrementMonthButton().setPressedIcon(null);
      monthAndYearPanel.getDecrementYearButton().setPressedIcon(null);

      //we will tweak the colours on the cellPanel background painter.
      cellPanelBackgroundPainter.setHeadingBackgroundEndGradientColor(Color.WHITE);
      cellPanelBackgroundPainter.setHeadingBackgroundEndGradientColor(Color.RED);

      //there are far too many options to tweak but the ones used above should
      //create a nice festive calendar.
}
```

The output of the above code and the custom CalendarCellRenderer written earlier produces the following result.



6 Date Picker Usage Guide

The Date Picker is a simple component used for selecting a single date. The Date Picker is a rather simplistic component, comprising a formatted text field and a button. When the button is activated a frame containing a Calendar is displayed. The user can either capture a date through the formatted text field or through the Calendar.

The Date Picker is very simplistic and acts as a utility component to assist the user in selected a single date.

6.1 How to create a Date Picker

When constructing a Date Picker it is important that you supply a correctly configured DateFormat and Calendar control to ensure the Date Picker behaviour aligns with your business logic. If you rely on the default constructor you will create a Date Picker whose internal Calendar will not contain any business logic, and a Date Format using the MEDIUM style of your system's locale.

```
/**
 * Ideally whenever you create a datePicker you should supply
 * a fully configured calendar(containing your formatting and business rules)
 * as well as the dateformat to use in the textfield.
 * @return
 */
private JSCDatePicker howToCreateADatePicker() {
      TimeZone timeZone = TimeZone.getDefault();
      Locale locale = Locale.getDefault();
      JSCCalendar calendar = new JSCCalendar(timeZone, locale);

      //in this example I will use the short dateFormat and the locale of your pc.
      DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.MEDIUM, locale);
      JSCDatePicker datePicker = new JSCDatePicker(dateFormat, calendar);

      return datePicker;
}
```

6.2 How to set the date on the Date Picker

The date displayed by the Date Picker is know as the selected date, and can be set using the available setter method setSelectedDate. The code below will set the Date Picker's selected date to Christmas 2009.

```
/**
 * If you need to set or read a date from the datePicker simply use the
 * getSelectedDate and setSelectedDate methods
 * @param datePicker
 */
private void howToSetADateOnTheDatePicker(JSCDatePicker datePicker) {
      Calendar calendar =
            Calendar.getInstance(datePicker.getCalendar().getTimeZone(),
                  datePicker.getCalendar().getLocale());
      calendar.set(Calendar.DAY_OF_MONTH, 25);
      calendar.set(Calendar.MONTH, Calendar.DECEMBER);
      calendar.set(Calendar.YEAR, 2009);
      Date december2009 = calendar.getTime();

      //this will set a date
      datePicker.setSelectedDate(december2009);

      //this will read a date
      datePicker.getSelectedDate();
}
```

6.3 How to listen for changes on the Date Picker

In order to listen to changes on the Date Picker, we simply listen for changes on the Calendar model embedded in the wrapped Calendar component.

```
/**
 * You should listen to changes on the datePicker by listening to changes on
 * the calendar component that it wraps.
 * @param datePicker
 */
private void howToListenToChangesOnTheDatePicker(JSCDatePicker datePicker) {
      datePicker.getCalendar().addCalendarSelectionListener(new
CalendarSelectionListener() {

      @Override
      public void selectedDatesChanged(CalendarSelectionEvent calendarSelectionEvent)
{
      //the calendar that has changed.
      JSCCalendar calendar = calendarSelectionEvent.getCalendar();

      //the selected dates the calendar holds.
      List<Date> selectedDates = calendarSelectionEvent.getSelectedDates();

      //the type of event that has occurred.
      CalendarSelectionEventType selectionEventType =
            calendarSelectionEvent.getCalendarSelectionEventType();

      switch (selectionEventType) {
            case DATE_REMOVED: {
                  //put your logic here to react to a date being removed.
                  System.out.println("A date has been removed");
                  break;
            }
            case DATE_SELECTED: {
                  //put your logic here to react to a date being selected/added
                  System.out.println("A date has been selected");
                  break;
            }
            case DATES_SELECTED: {
                  //put your logic here to react to multiple dates being
                  //selected/added
                  System.out.println("Dates have been selected");
                  break;
            }
            case DATES_CLEARED: {
                  //put your logic here to react to all dates being removed
                  System.out.println("All dates have been cleared");
                  break;
            }
            case DISPLAY_DATE_CHANGED: {
                  //put your logic here to react to the calendar displaying a new
                  // date
                  System.out.println("Display date moved");
            }
      }
}
});
```

6.4 How to change the behaviour of the Date Picker

The Date Picker has a number of useful properties that can be set which will change how it behaves. The code below lists some of the most useful properties.

```
/**
 * There are a number of useful methods that you can access that changes the
 * behaviour of the datePicker.
 * @param datePicker
 */
private void howToAccessOtherSettings(JSCDatePicker datePicker) {
      //if the user deletes all the text in the formattedTextField,
      //the datePicker will store this information as a null date.
      //if this flag is true, the formatted textfield will force the user
      //to select a date.
      datePicker.setAllowNullDates(true);

      //this lets you choose the location the frame will appear relative
      //to the datePicker's button.
      datePicker.setFramePosition(FramePosition.SOUTH_WEST);

      //this will change the size of the calendar's frame
      datePicker.setFrameSize(new Dimension(250,250));

      //sets the title of the frame
      datePicker.setFrameTitle("Demo frame");

      //determines if the calendar should hide after the user
      //has selected a date.
      datePicker.setCloseFrameAfterSelection(true);

      //Decoration refers to the buttons at the top of the frame
      //setting decorate frame to false will hide the buttons
      //and the frame's title
      datePicker.setDecorateFrame(true);
}
```

6.5 How to access the nested components

The standard implementations of Date Picker comprise of a formatted text field, a button and a frame. If you need to gain access to these nested components you can do so through the Date Pickers UI.

```
/**
 * If you need to access the inner components of the datePicker
 * such as the button and the formattedTextfield you can do so through the UI.
 * @param datePicker
 */
private void howToAccessTheNestedComponents(JSCDatePicker datePicker) {
      //the standardFormatUI comprises a formattedTextfield and a button.
      StandardFormatDatePickerUI ui = (StandardFormatDatePickerUI)
            datePicker.getUI();

      AbstractButton btnShowCalendar = ui.getBtnShowCalendar();
      JFormattedTextField txtDate = ui.getTxtDate();
}
```