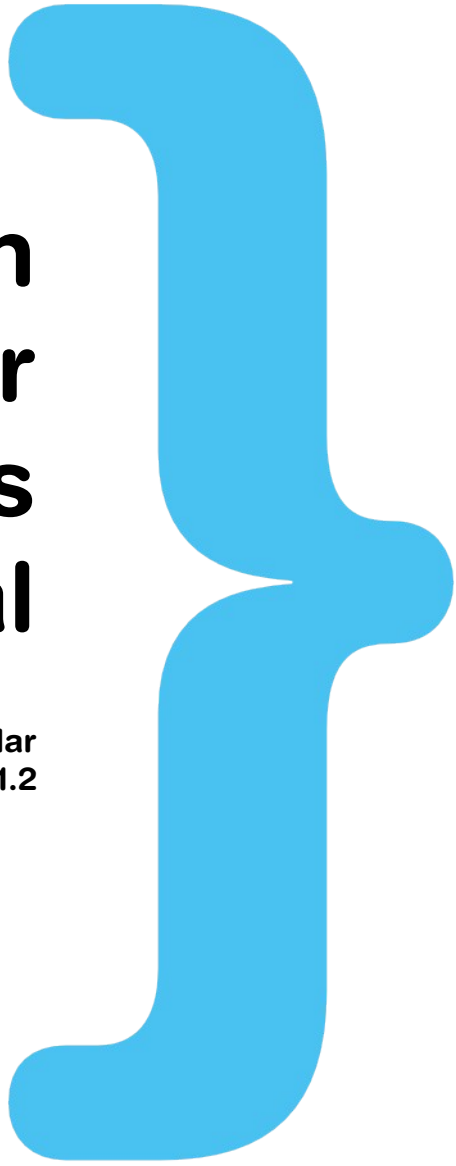
Vaadin Calendar User's Manual

Vaadin Calendar
1.2



Vaadin Calendar is a tool for displaying and managing events.

Publication date of this manual version is 2011-07-15.

Published by:
Vaadin Ltd
Ruukinkatu 2-4
20540 Turku
Finland

Table of Contents

1. Introduction to Vaadin Calendar

1.1. Overview

The Vaadin Calendar is a component for organizing and displaying events. It can be used to view and manage events in a monthly view or a weekly view. Calendar's API allows to manage its events, date range, even styling, localization and timezone. Calendar's data source may be anything, as its events are queried dynamically by the component. You may use containers, or any other data source by implementing Calendar's event provider.

1.2. Views

The Vaadin Calendar has two different type of views that are shown depending on the Calendar's date range. The Calendar displays a week by default. Calendar will be shown in a monthly view when date range is over than one week (seven days) long. Date range is always calculated in a accuracy of one millisecond.

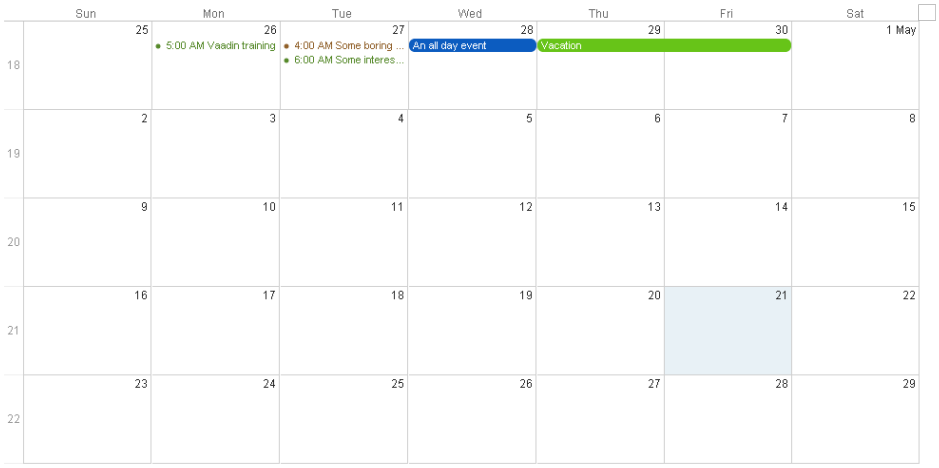


Figure 1: Monthly view with few all-day and normal events

The monthly view can easily be used to control all events, but it's best suited for longer events. Events that last for several days can be easily displayed and moved in the monthly view. In the figure above, you can see two longer events that are highlighted with a blue and green background color. Other markings are shorter day events that last less than a 24 hours. These events cant be moved.

When date range is over seven days long, the weekly view will be used. Weekly view can show any number of days from one to seven.

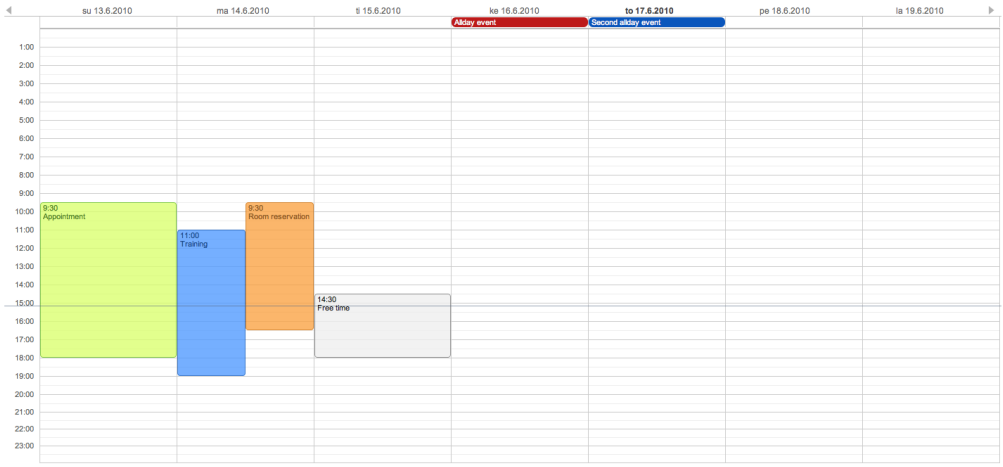


Figure 2: Weekly view

In the figure above you can see four normal day events and also all-day events at the top of the time line grid.

1.3. Calendar event

The Vaadin Calendar has an interface **CalendarEvent** for the events it displays. The concrete class of the event is decided by the implementor of the **CalendarEventProvider** that is used. By default, Calendar uses a **BasicEventProvider** to provide events. The **BasicEventProvider** uses instances of **BasicEvent**.

The Vaadin Calendar doesn't depend on any particular data source implementation. Events are queried by the Calendar from the provider that just has to implement **CalendarEventProvider**. It is up to the event provider that calendar gets the correct events.

Event query happens by asking the event provider for all the events between two dates. The range of these dates is guaranteed to be at least as long as the start and end dates set for the component. The component can however ask for a longer range to ensure correct rendering. In particular, all start dates are expanded to the start of the day, and all end dates are expanded to the end of the day.

Calendar event needs always a start time and an end time. These are the only obligatory fields for the component's UI. In addition, an event can also be set as an all-day event by setting the all-day field of the event. You can also set the description of an event, which is displayed as a tooltip in the UI.

If the all-day field of the event is true, then the event is always rendered as an all-day event. In the monthly view, this means that no start time is displayed in the UI and the event has an colored background. In the weekly view, all-day events are displayed in the upper part of the screen, and rendered similarly to the monthly view. In addition, when an event's time range is 24 hours or longer, it is rendered as an all-day event in the monthly view.

When event's time range is equal or less than 24 hours, with the accuracy of one millisecond, then the event is considered as a normal day event. Normal event has a start and end times that may be on different days.

1.4. Interaction

The Vaadin Calendar's date/week captions and events are clickable and the clicks can be listened by the server. Also date/time range selections, event dragging and event resizing can be listened by the server. By the API, programmer has a full control over the UI events.

Weekly view has navigation buttons to navigate forward and backward in the time. These actions are also listened by the server. Custom navigating is can be implemented using event handlers, which are described in section 2.7.2.

Most of the handlers related to Calendar events have sensible default handlers. These are found in the **com.vaadin.ui.handler** package. The default handlers and their functionalities are described below.

- **BasicBackwardHandler**. Handles clicking the back-button of the weekly view so that the viewed month is changed to the previous one.
- **BasicForwardHandler**. Handles clicking the forward-button of the weekly view so that the viewed month is changed to the next one.
- **BasicWeekClickHandler**. Handles clicking the week numbers int the monthly view so that the viewable date range is changed to the clicked week.
- **BasicDateClickHandler**. Handles clicking the dates on both the monthly view and the weekly view. Changes the viewable date range so that only the clicked day is visible.
- **BasicEventMoveHandler**. Handles moving the events in both monthly view and the weekly view. Events can be moved and their start and end dates are changed correctly, but only if the event implements **CalendarEventEditor** (implemented by **BasicEvent**).
- **BasicEventResizeHandler**. Handles resizing the events in the weekly view. Events can be resized and their start and end dates are changed correctly, but only if the event implements **CalendarEventEditor** (implemented by the **BasicEvent**).

All of these handlers are automatically set when creating a new Calendar. If you wish to disable some of the default functionality, then you can simply set the corresponding handler to null. This will prevent the functionality to ever appearing on the UI. For example, if you set the **EventMoveHandler** to null, the user will be unable to move events in the browser.

2. Using Vaadin Calendar

2.1. Overview

It is very easy to get from 0 to ready with the Vaadin Calendar. All you need to do is install the JAR as described below, and you are ready to go. By default the Vaadin Calendar uses a **BasicEventProvider** for events and displays a date range of one week. After the initial installation, you can start customizing the Calendar for your project.

2.2. Installing

Installing Vaadin Calendar doesn't differ from installing any other Vaadin add-on:

- Download the jar from http://vaadin.com/directory#addon/124
- Copy it to your web project's WEB-INF/lib directory.
- Recompile your widgetset. If you are using Eclipse and the Vaadin Eclipse Plugin (available at http://vaadin.com/eclipse) the widgetset should be compiled automatically and you are ready to go.
- For more instructions and information, refer to http://vaadin.com/directory/help/using-vaadin-add-ons.

Assuming that you already have Vaadin installed in your project, Vaadin Calendar is now installed and available in your project.

2.3. Date range: start and end dates

One of the first things you'll notice about the Vaadin Calendar is that it only displays one week at a time. We can easily change that by using start and end dates. The date range must be between one and 60 days. Here we set the calendar to show only one day, which is the current day.

```
cal.setStartDate(new Date());
cal.setEndDate(new Date());
```

Notice that although the range we set is actually 0ms long, the calendar still renders time from 00:00 to 23:59. This is normal, as the Vaadin Calendar is guaranteed to render *at least* the date range provided, but may expand it. This is more important when we implement our own Event Providers.

2.4. Event provider

The second thing you'll probably notice about the Calendar is that it is pretty empty. Lets add some events. In the beginning the default **BasicEventProvider** will suffice, but later on you'll probably want to create your own Event Provider as described in the next section.

As the **BasicEventProvider** is used by default, we can just retrieve from the Calendar and add an event to it.

```
BasicEvent event = new BasicEvent();
        java.util.Calendar calendar =
java.util.Calendar.getInstance();
        calendar.setTime(new Date());
        event.setStart(calendar.getTime());

        calendar.add(java.util.Calendar.HOUR, 3);
        event.setEnd(calendar.getTime());
        event.setCaption("FooBar");

        BasicEventProvider eventProvider = (BasicEventProvider)
cal.getEventProvider();
        eventProvider.addEvent(event);
```

This adds a new event that lasts for 3 hours. As the **BasicEventProvider** and **BasicEvent** implement some optional event interfaces provided by the calendar package, there is no need to refresh the calendar. Just create events, set their properties and add them to the Event Provider.

2.4.1. Customized Events and Event Provider

To begin customizing the Calendar Events for your project, you should start by creating an event class that implements the **CalendarEvent** interface. Below is the **BasicEvent** from the **com.vaadin.addon.calendar.event** package. It nicely presents the normal work required for an event.

```
public class BasicEvent implements CalendarEventEditor,
EventChangeNotifier {

   ...

   public String getCaption() {
        return caption;
    }

    public String getDescription() {
        return description;
    }

    public Date getEnd() {
        return end;
    }

    public Date getStart() {
        return start;
    }

    public String getStyleName() {
        return styleName;
    }

    public boolean isAllDay() {
        return isAllDay;
    }

    public void setCaption(String caption) {
        this.caption = caption;
```

```
        fireEventChange();
    }

    public void setDescription(String description) {
        this.description = description;
        fireEventChange();
    }

    public void setEnd(Date end) {
        this.end = end;
        fireEventChange();
    }

    public void setStart(Date start) {
        this.start = start;
        fireEventChange();
    }

    public void setStyleName(String styleName) {
        this.styleName = styleName;
        fireEventChange();
    }

    public void setAllDay(boolean isAllDay) {
        this.isAllDay = isAllDay;
        fireEventChange();
    }

    public void addListener(EventChangeListener listener) {
        ...
    }

    public void removeListener(EventChangeListener listener) {
        ...
    }

    protected void fireEventChange() {...}
}
```

You may have noticed that there was some additional code in the **BasicEvent** that wasn't from the **CalendarEvent** interface. Namely **BasicEvent** also implements two additional interfaces:

- **CalendarEditor**. This adds setters for all the fields, and is required for some of the default handlers to work.
- **EventChangeNotifier**. This adds the possibility to listen for changes in the event, and enables the Calendar to render the changes immediately.

Start and end dates are mandatory, but caption, description style name are not. Style name is used as a part of the CSS class name for the event's DOM element.

Now you can create events for the Calendar. The Calendar needs also a data source which is known as an Event Provider for the Calendar. Lets create an example Event Provider example that implements **CalendarEventProvider.**

```
public class MyEventProvider implements CalendarEventProvider{
```

```
    public List<Event> getEvents(Date startDate, Date endDate){

        List<Event> events = new ArrayList<Event>();
        GregorianCalendar cal = new GregorianCalendar();
        cal.setTime(new Date());

        Date start = cal.getTime();
        cal.add(GregorianCalendar.HOUR, 5);
        Date end = cal.getTime();
        BasicEvent event = new BasicEvent();
        event.setCaption("My Event");
        event.setDescription("My Event Description");
        event.setStart(start);
        event.setEnd(end);
        events.add(event);

        return events;
    }
}
```

CalendarEventProvider has only one method to be implemented. Whenever the Calendar is painted, **CalendarEventProvider.getEvents(Date, Date)** method is called. In **MyEventProvider** example, only one event will be returned, that is five hours long and it starts from the current date and clock time.

It is important to notice, that the Calendar may query for dates beyond the range defined by start date and end date. Particularly, the Calendar may expand these dates to make sure the UI is rendered correctly.

In addition to the basic event interfaces, you can enhance the functionality of your Event and Event Provider classes by using **EventChange** and **EventSetChange** events. These will enable the Calendar component to know about changes in events and render itself accordingly. **BasicEvent** and **BasicEventProvider** include a simple example implementation of these interfaces.

2.5. Sizing

The Vaadin Calendar supports the dynamic size system of Vaadin, with both defined and undefined sizes. When using defined sizes, the Calendar calculates the correct height for the cells so that it fits to the size given. When using an undefined size Calendar, all the sizes come from CSS. In addition, when height is undefined a scrollbar is displayed in the weekly view to better fit the cells to the UI. See 2.6.1. for information about customizing the undefined sizes.

2.6. Styling

Calendar's default theme comes with the component. You may choose to overwrite the stylenames from the default theme file *calendar.css*. The Calendar's default theme is located in a folder named “public” under the src directory in the JAR file. Vaadin will find the CSS from inside the JAR package.

2.6.1. Undefined Style

Usually you don't need to overwrite any of the default styles, but an undefined size Calendar is a exception of this. Below is a list of stylenames that define the size of an undefined size Calendar (these are the defaults from *calendar.css*)

```
.v-calendar-month-sizedheight .v-calendar-month-day {
height: 100px;
}

.v-calendar-month-sizedwidth .v-calendar-month-day {
width: 100px;
}

.v-calendar-header-month-Hsized .v-calendar-header-day {
width: 101px;
}

/* for IE */
.v-ie6 .v-calendar-header-month-Hsized .v-calendar-header-day {
width: 104px;
}

/* for others */
.v-calendar-header-month-Hsized td:first-child {
padding-left: 21px;
}

.v-calendar-header-day-Hsized {
width: 200px;
}

.v-calendar-week-numbers-Vsized .v-calendar-week-number {
height: 100px;
line-height: 100px;
}

.v-calendar-week-wrapper-Vsized {
height: 400px;
overflow-x: hidden !important;
}

.v-calendar-times-Vsized .v-calendar-time {
height: 38px;
}

.v-calendar-times-Hsized .v-calendar-time {
width: 42px;
}

.v-calendar-day-times-Vsized .v-slot,.v-calendar-day-times-
Vsized .v-slot-even {
height: 18px;
}

.v-calendar-day-times-Hsized, .v-calendar-day-times-Hsized .v-
slot,.v-calendar-day-times-Hsized .v-slot-even {
```

```
width: 200px;
}
```

2.6.2. Event's style

Events can be styled with CSS by setting them some class name like *“color1”*, *“color2”*. Style name is retrieved for the event by **CalendarEvent.getStyleName()** method. The String returned by that method will be used as a part of the event's main element class name. *“color1”* would be identified with a class name *“v-calendar-event-color1”*. In CSS, this means that element can be styled with the following example CSS.

```
.v-calendar .v-calendar-event-color1 {
color: #4f8324;
}

.v-calendar .v-calendar-event-color1-all-day {
background-color: #61c114;
}

/* For week/day view */
.v-calendar .v-calendar-event-color1 .v-calendar-event-caption
{
color: #4f8324;
}

.v-calendar .v-calendar-event-color1 .v-calendar-event-content
{
border-color: #61c114;
background-color: #daff70;
}
```

2.7. Visible hours and days

As we saw in section 2.3., by setting the date range you can change the range of dates that are shown by the Calendar. But what if you wanted to show the whole month but hide weekends, or show only hours from 8 to 16 when viewing a whole week? That's where you can utilize the **setVisibleDays** and **setVisibleHours** -methods. To achieve what we just described, see the example code below.

```
calendar.setVisibleDays(1,5);
calendar.setVisibleHours(0,15);
```

After these calls, only weekdays from monday to friday will be rendered. And when in the Calendar is in the weekly view, only hours from 00:00 to 16:00 are shown. Note that the excluded times are never rendered so care should be taken when setting the date range. If the date range contains only dates / times that are excluded, nothing will be rendered. Also note that even if a date is not rendered because these settings, the EventProvider may still be queried for events for that date.

2.8. Drag and drop

In Vaadin, it is possible to drag and drop components and certain parts of components. Too see the full explanation of drag and drop in Vaadin, including the terms used in this section, see the chapter 11.13 of the Book of Vaadin.

The Vaadin Calendar can act as a drop target. In order to do this, it must have a drop handler. When the drop handler is set, the days in the monthly view and the time slots in the weekly view can receive drops. Other places (such as day names in the weekly view) cannot receive drops.

The Calendar uses its own **TargetDetails** implementation, **CalendarTargetdetails**. It holds information about the place (date and time) of the drop location. The drop location can be retrieved via the **getDropTime** -method. If the drop happened in the monthly view, the returned date doesn't have exact time information. If the drop happened in the weekly view, the returned date also contains the start time of the slot.

Below is a short example on creating a drop handler and using the drop information to create a new event. The whole source code can be found online with the name **CalendarDragAndDropDemo** (http://dev.vaadin.com/svn/addons/Calendar). Some of the less important code has been left out to make the code more readable and shorter.

```
private Calendar createDDCalendar() {
  Calendar calendar = new Calendar();
  calendar.setDropHandler(new DropHandler() {
    public void drop(DragAndDropEvent event) {
      CalendarTargetDetails details =
              (CalendarTargetDetails) event.getTargetDetails();

      TableTransferable transferable =
              (TableTransferable) event.getTransferable();

      createEvent(details, transferable);
      removeTableRow(transferable);
    }

    public AcceptCriterion getAcceptCriterion() {
      return AcceptAll.get();
    }

  });

  return calendar;
}


protected void createEvent(CalendarTargetDetails details,
  TableTransferable transferable) {
  Date dropTime = details.getDropTime();
  java.util.Calendar timeCalendar = details.getTargetCalendar()
                                    .getInternalCalendar();
  timeCalendar.setTime(dropTime);
  timeCalendar.add(java.util.Calendar.MINUTE, 120);
  Date endTime = timeCalendar.getTime();

  Item draggedItem = transferable.getSourceComponent().
                            getItem(transferable.getItemId());

  String eventType = (String)draggedItem.
                            getItemProperty("type").getValue();

  String eventDescription = "Attending: "
            + getParticipantString(
```

```
                (String[]) draggedItem.
                  getItemProperty("participants").getValue());

  BasicEvent newEvent = new BasicEvent();
  newEvent.setAllDay(!details.hasDropTime());
  newEvent.setCaption(eventType);
  newEvent.setDescription(eventDescription);
  newEvent.setStart(dropTime);
  newEvent.setEnd(endTime);

  BasicEventProvider ep = (BasicEventProvider) details
                      .getTargetCalendar().getEventProvider();
  ep.addEvent(newEvent);
}
```

2.9. Localization and miscellaneous

By default, Vaadin Calendar uses the system's default locale for its internal **Calendar**. Month and weekday names will be translated by the effective locale. Calendar's locale can be set by **setLocale(java.util.Locale locale)** method. Setting the locale will update also other location specific date and time settings like the first day of the week, timezone and time format. However, timezone and time format can be overridden by the Vaadin Calendar settings.

```
cal.setLocale(Locale.US);
```

You may change the format of weekly view's date captions with the **setWeeklyCaptionFormat(String dateFormatPattern)** method. You may read more about the allowed pattern from the JavaDoc of **java.text.SimpleDateFormat** class.

```
cal.setWeeklyCaptionFormat("dd-MM-yyyy");
```

Timezone can be changed by the method **setTimeZone(java.util.TimeZone)**. Setting timezone to null will reset timezone to the locale's default.

```
cal.setTimeZone(TimeZone.getTimeZone("Europe/Helsinki"));
```

Time may be in 24 or 12 hours format. It is defined by the locale, but may also be set by the method **setTimeFormat(TimeFormat)**. Settings it to null will reset time format to the locale's default.

```
cal.setTimeFormat(TimeFormat.Format12H);
```

2.10. Code examples for customizing the Vaadin Calendar

In the following example we will create an application that shows you the features of the Vaadin Calendar. Event provider and styling was described earlier, so now we concentrate on the Calendar API's other features.

The whole example application source code can be found online with the name **CustomizedCalendarDemo** (http://dev.vaadin.com/svn/addons/Calendar). Some of the less important code for this document has been left out to make the code more readable and shorter. Three dots are used to mark places for the missing code.

2.10.1. Calendar instance

Create a new Vaadin Calendar instance by the following code snippet. Here we use our own Event Provider, the earlier described **MyEventProvider.**

```
Calendar cal = new Calendar(new MyEventProvider());
```

This initializes the Calendar. To customize the viewable date range, we must set a start and end date to it.

There is only one visible event on the timeline, starting from the current time. That's what our event provider passes to the client. See the figure 3.

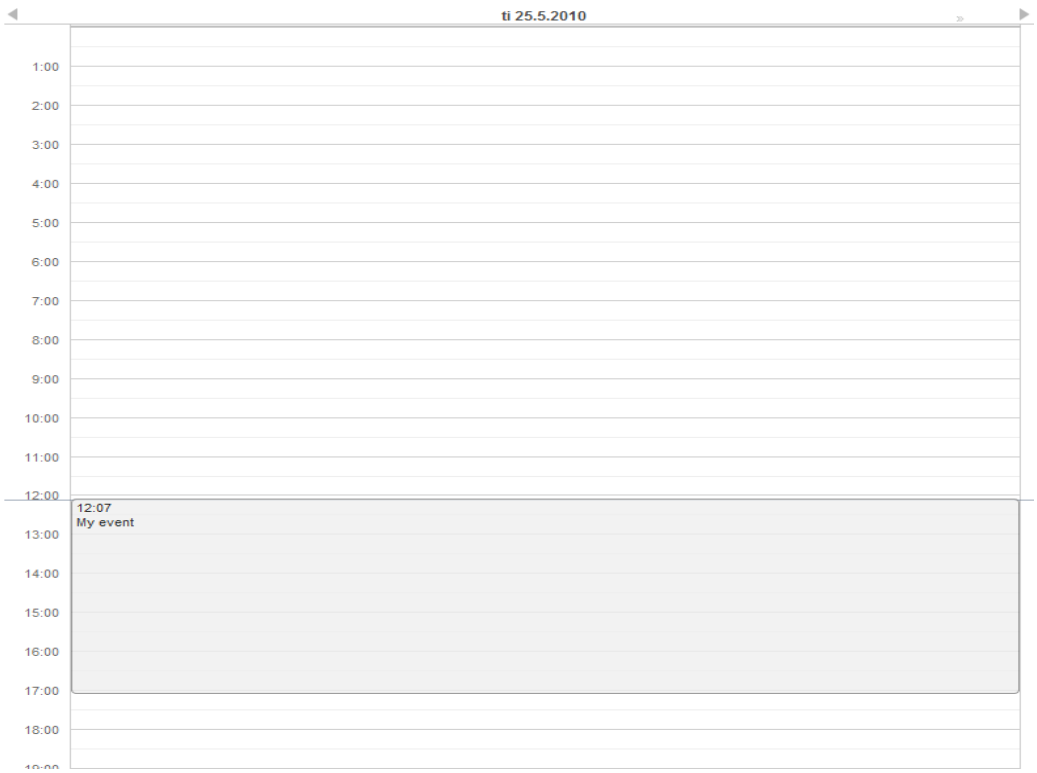


Figure 3: Weekly view with a single visible day and one event.

It would be nice to also be able to control the navigation forward and backward. The default navigation is provided by the default handlers, but perhaps we want to restrict the users so they can only navigate dates in the current year. Maybe we also want to pose some other restrictions to the clicking week numbers and dates.

These restrictions and other custom logic can be imposed by using custom handlers. The handlers can be found in the **com.vaadin.addon.calendar.ui.handler** package and can be easily extended. Note that if you don't want to extend the default handlers, you are free to implement your own. The interfaces are described in **CalendarComponentEvents**.

2.10.2. Backward and forward navigation

In the weekly view, there are navigation buttons in the top left and right corner of the component. See the figure 4.

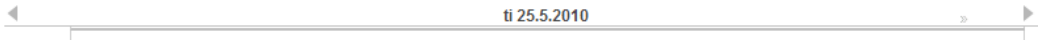


Figure 4: Backward and forward navigation buttons.

Backward and forward navigation is handled via **BackwardListener** and **ForwardListener**.

```
cal.setHandler(new BasicBackwardHandler() {

  protected void setDates(BackwardEvent event, Date start, Date
end) {

    java.util.Calendar calendar = event.getComponent()
                                    .getInternalCalendar();
    if (isThisYear(calendar, end)
         && isThisYear(calendar, start)) {
      super.setDates(event, start, end);
    }
  }
});
```

The forward-handler can be implemented identically. This handler restricts the setting of dates to the current year.

2.10.3. Date click handling

Date click handling by default switches the current date range to one day. The date click event is handled by a **DateClickHandler** . The following code snippet will the handle click event so that when the user clicks the date header in the weekly view, it will either show a single day or whole week depending on the current status.

```
cal.setHandler(new BasicDateClickHandler() {

  public void dateClick(DateClickEvent event) {
    Calendar cal = event.getComponent();
    long currentCalDateRange = cal.getEndDate().getTime()
                               - cal.getStartDate().getTime();

    if (currentCalDateRange < VCalendar.DAYINMILLIS) {
      // Change the date range to the current week
      cal.setStartDate(cal.getFirstDateForWeek(event.getDate())
);
```

```
      cal.setEndDate(cal.getLastDateForWeek(event.getDate()));

    } else {
      // Default behaviour, change date range to one day
      super.dateClick(event);
    }
  }
});
```

2.10.4. Week click handling

In the monthly view, on the left side of the date grid, there are clickable week numbers. You can handle the click event by setting a **WeekClickHandler** for the Calendar. The default handler changes the date range to be the clicked week.

In the following code snippet we add a week click handler that changes the Calendar's date range to one week only if the start and end dates of the week are in the current month.

```
cal.setHandler(new BasicWeekClickHandler() {

  protected void setDates(WeekClick event, Date start, Date
end) {
    java.util.Calendar calendar = event.getComponent()
                                  .getInternalCalendar();
    if (isThisMonth(calendar, start)
        && isThisMonth(calendar,  end)) {
      super.setDates(event, start, end);
    }
  }
});
```

2.10.5. Event click handling

Monthly and weekly view's Calendar events are clickable, but they have no default handler. Just like the date and week click handlers, event click handling is enabled by setting an **EventClickHandler** for the Calendar. The clicked event will be available by the **EventClick.getCalendarEvent()** method. Example code below.

```
cal.addListener(new EventClickListener() {

    public void eventClick(EventClick event) {
        BasicEvent e = (BasicEvent) event.getCalendarEvent();
          getMainWindow().showNotification(
                        "Event clicked: " + e.getCaption(),
                        e.getDescription());
    }
});
```

2.10.6. Event dragging

An event can be dragged to change its position. The default handler for event dragging sets the events start and end dates accordingly. To restrict event moving, we can set a customized move handler

Below we add a **EventMoveHandler** to the Calendar. The event handler will update the new position to the datasource, if the new dates are in the current month. Also some other changes must be made to our Event Provider class.

```
cal.setHandler(new BasicEventMoveHandler() {
  private java.util.Calendar javaCalendar;

  public void eventMove(MoveEvent event) {
    javaCalendar = event.getComponent().getInternalCalendar();
    super.eventMove(event);
  }


  protected void setDates(CalendarEventEditor event,
                          Date start, Date end) {
    if (isThisMonth(javaCalendar, start)
        && isThisMonth(javaCalendar, end)) {
      super.setDates(event, start, end);
    }
  }
});
```

Our Event Provider needs to be slightly changed so that it doesn't always create a new event when **getEvents** is called.

```
public static class MyEventProvider implements
CalendarEventProvider {

  private List<CalendarEvent> events = new
ArrayList<CalendarEvent>();

  public MyEventProvider() {
    events = new ArrayList<CalendarEvent>();
    GregorianCalendar cal = new GregorianCalendar();
    cal.setTime(new Date());

    Date start = cal.getTime();
    cal.add(GregorianCalendar.HOUR, 5);
    Date end = cal.getTime();
    BasicEvent event = new BasicEvent();
    event.setCaption("My Event");
    event.setDescription("My Event Description");
    event.setStart(start);
    event.setEnd(end);
    events.add(event);
  }

  public void addEvent(CalendarEvent BasicEvent) {
    events.add(BasicEvent);
  }

  public List<CalendarEvent> getEvents(Date startDate,
                                       Date endDate) {
    return events;
  }
}
```

After these changes users can move events around, but when they drop them the start and end dates are checked by the server. Note that as the server-side *must* move the event in order for it to render to the place it was dropped. The server can also reject moves by not doing anything when the event is received.

2.10.7. Drag selection handling

Drag selection works in both monthly and weekly view. To listen for drag selection, add **RangeSelectListener** to the Calendar. There is no default handler for range select.

In the code example below, we create an new event to the Calendar when any date range is selected. Drag selection will open a window where user is asked for a caption for the new event. After confirming, new event will be passed to our event provider and calendar will be updated. Note that as our Event Provider and Event classes do not implement the event change interface, we must refresh the Calendar manually after changing our events.

```
cal.setHandler(new RangeSelectHandler() {

  public void rangeSelect(RangeSelectEvent event) {
    BasicEvent calendarEvent = new BasicEvent();
    calendarEvent.setStart(event.getStart());
    calendarEvent.setEnd(event.getEnd());

    // Create popup window and add a form in it.
    VerticalLayout layout = new VerticalLayout();
    layout.setMargin(true);
    layout.setSpacing(true);

    final Window w = new Window(null, layout);
    ...

    // Wrap the calendar event to a BeanItem
    // and pass it to the form
    final BeanItem<CalendarEvent> item =
                         new BeanItem<CalendarEvent>(myEvent);

    final Form form = new Form();
    form.setItemDataSource(item);
    ...

    layout.addComponent(form);

    HorizontalLayout buttons = new HorizontalLayout();
    buttons.setSpacing(true);
    buttons.addComponent(new Button("OK", new ClickListener() {

        public void buttonClick(ClickEvent event) {
            form.commit();
            // Update event provider's data source
            provider.addEvent(item.getBean());
            // Calendar needs to be repainted
            cal.requestRepaint();
            getMainWindow().removeWindow(w);
        }
    }));
```

```
    ...
  }
});
```

2.10.8. Event resizing

An Event can be resized from both ends to change its start or end time. This offers the users a convenient way to change event times without the need to type anything. The default resize handler sets the events start and end times according to the resize.

In the code below, we set a custom handler for resize events. This handler prevents any event to be resized over 12 hours in length. Note that this does not prevent the user from resizing an event over 12 hours in the client. The resize will just be corrected by the server.

```
cal.setHandler(new BasicEventResizeHandler() {

  private static final long twelveHoursInMs = 12*60*60*1000;

  protected void setDates(CalendarEventEditor event,
                          Date start, Date end) {
    long eventLength = end.getTime() - start.getTime();
    if (eventLength <= twelveHoursInMs) {
      super.setDates(event, start, end);
    }
  }
});
```