Implementing a Calendar in Web Dynpro Java

Applies to:

Web Dynpro for Java applications for SAP NetWeaver CE 7.1.

Summary

The Tutorial describes how to create and implement a calendar application using the newly introduced Calendar UI elements. This tutorial focuses on creating a team calendar displaying the schedule for several team members and a weekly view for a selected member. In addition you learn to implement popup windows to create, edit and delete entries and to use CCTS-based Core Data Types.

Details

Level of complexity: Intermediate

Time required for completion: 99 min.

Sample Project

You can download the running example here: Implementing Calendar in Web Dynpro Java.zip

Author(s): SAP NetWeaver Product Management User Interaction, Stefanie Bacher
With grateful thanks to Nicole Bohrmann, who created this tutorial during her internship in our team.

Company: SAP AG

Created on: 21 July 2007

Author Bio

Stefanie Bacher works as an information developer within the SAP NetWeaver Product Management User Interaction Team. She focuses on knowledge distribution of Web Dynpro for Java.

Table of Contents

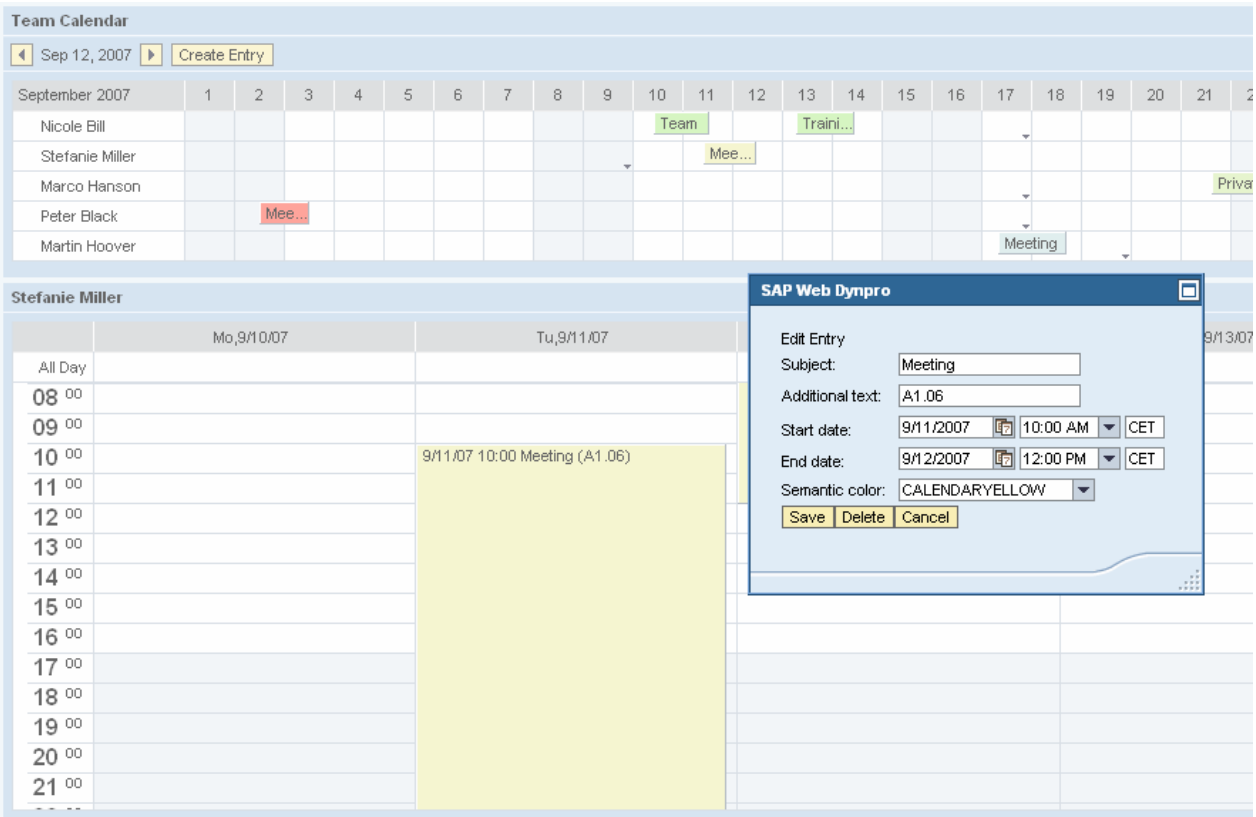
Introduction

The **team calendar** is the basis of this project. The context must therefore be adapted to this type of calendar. As the team calendar is created according to the tree node principle, its structure – which displays the team members in a list – dictates that the context must have a recursive node. This node then makes various team members available.

Selecting a day in the team calendar will display the selected week for this person in a weekly calendar below. **Navigation** between the two calendar types is made by triggering an event. This event then passes two previously defined parameters to the action in order to define the person and the date for the other calendar that the user has clicked. When a selection is made in the team calendar, a corresponding period is then displayed for the relevant person in the weekly calendar.

In addition you will learn to **create new entries, and change or delete existing ones**. The function for changing the calendar is made possible by **parameter mapping**. The data to be changed is stored in a temporary storage location. The data is not created or written to the corresponding calendar storage location until this is confirmed by pressing the relevant button. Parameter mapping is performed to access the edited entry and its index.

Data to be changed is entered in a **popup** and passed on to the calendar. This popup is created in a window and can only be closed again by the window that called it. This is done by triggering an action in the closing window. Event handling for this window is then delegated to the component controller.



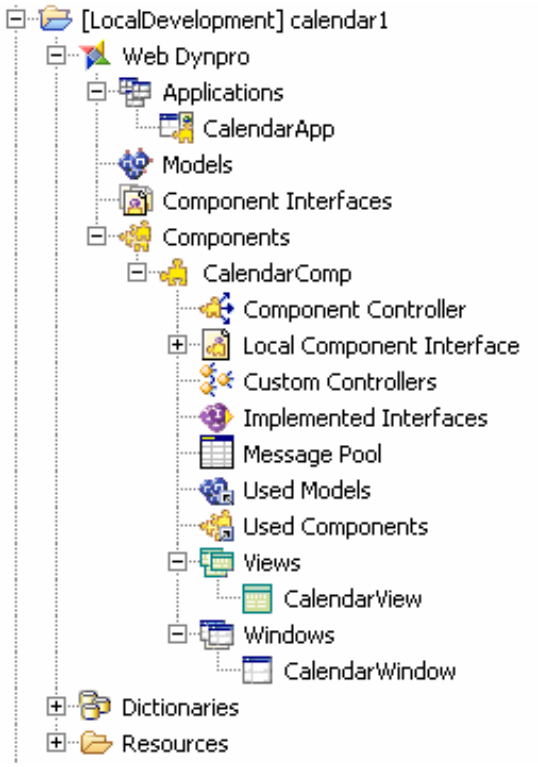
In this tutorial, you learn how to:

- Create the layout for the various calendar types
- Create the context for a team calendar using *Recursion Node*
- Define the navigation between the various elements
- Create and implement parameter mapping
- Work with core data types
- Map the context between a view and the component controller

- Create new views and windows
- Use event handling
- Implement popups

Creating a Web Dynpro Project and Application

Create a Web Dynpro Project called **Calendar**. Name the application **CalendarApp** and the package *com.sap.examples.calendarapp*. The component is called **CalendarComp**, and the package is *com.sap.examples.calendarcomp.*



Designing the Layout

At first you will create a team calendar. The appointments for each team member are displayed in a vertical list for a calendar month. The vertical display is based on the tree principle.

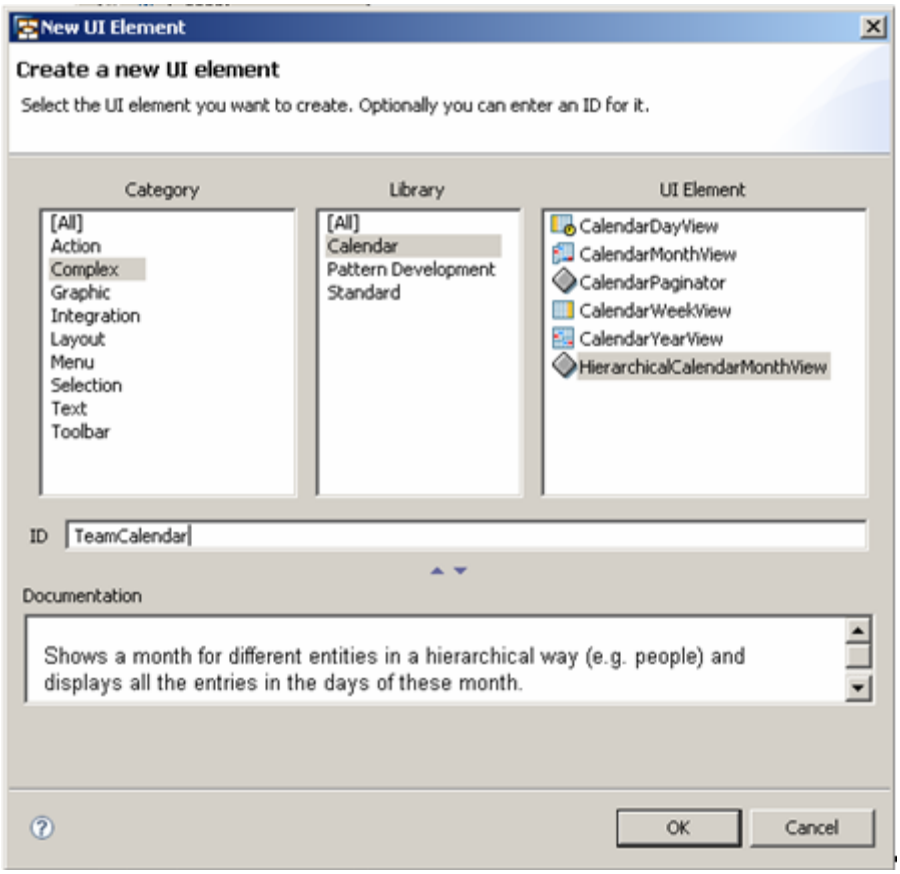
To provide more information about an appointment for a given team member, you also need to design a weekly calendar that displays the appointment in more detail in the weekly calendar upon selection of the member and corresponding time.

Navigation in the team calendar is on a month-by-month basis using UI element *CalendarPaginator*.

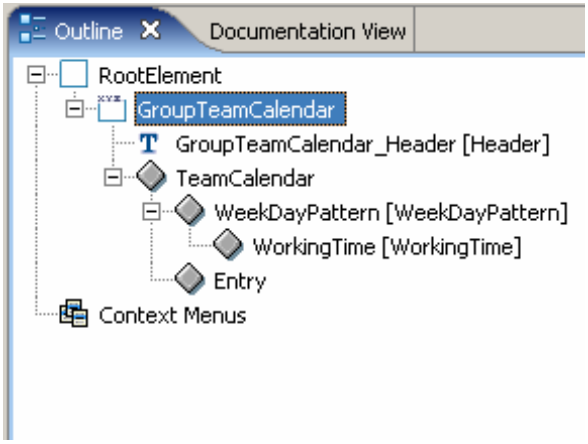
Designing the Team Calendar

1. Double-click the *CalendarView* to open it.
2. Remove the *DefaultTextView* element from the layout by making a right click on it and choosing *Delete* from the context menu.
3. Change the *layout* of the *RootElement* container. Select the *RootElement* and switch to the *Properties* tab. Change the *layout* property to *MatrixLayout*.
4. Insert a *Group* container in the *RootElement* container. Make a right click and choose *Insert Child*. In the wizard, select the *Group* element and enter **GroupTeamCalendar** as the ID.

5. Select *GroupTeamCalendar_Header* and enter **Team Calendar** as *text* property.
6. Make a right click on the *RootElement* and choose *Insert Child* to insert the *HierarchicalCalendarMonthView* element called **TeamCalendar** into the *GroupTeamCalendar* element.

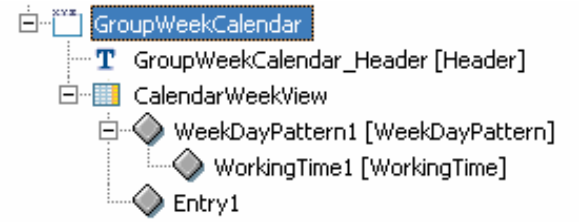


7. Make a right click on the *TeamCalendar* element and choose *Insert Entry*.
8. Make another right click on the *TeamCalendar* element and choose *Insert WeekDayPattern*.
9. Make a right click on the *WeekDayPattern* element and choose *Insert WorkingTime.* The structure of your team calendar now looks like the figure below:



Designing the Weekly Calendar

1. Create a *GroupContainer* in the *RootElement* container. Make a right click on *Insert Child* and choose the *Group* element. Enter **GroupWeekCalendar** as the ID and click *OK*.
2. Change the *layoutData* property in the *GroupWeekCalendar* to **MatrixHeadData**
3. Make a right click on *GroupWeekCalendar* and choose *Insert Child* to insert the *CalendarWeekView* element.
4. Set the *firstDayofWeek* property to **monday**.
5. Make a right click on the *CalendarWeekView* element and choose *Insert Entry*.
6. Make another right click on the *CalendarWeekView* element and choose *Insert WeekDayPattern*.
7. Make a right click on the *WeekDayPattern* element and choose *Insert WorkingTime* to insert the *WorkingTime* element into the *WeekDayPattern* element.

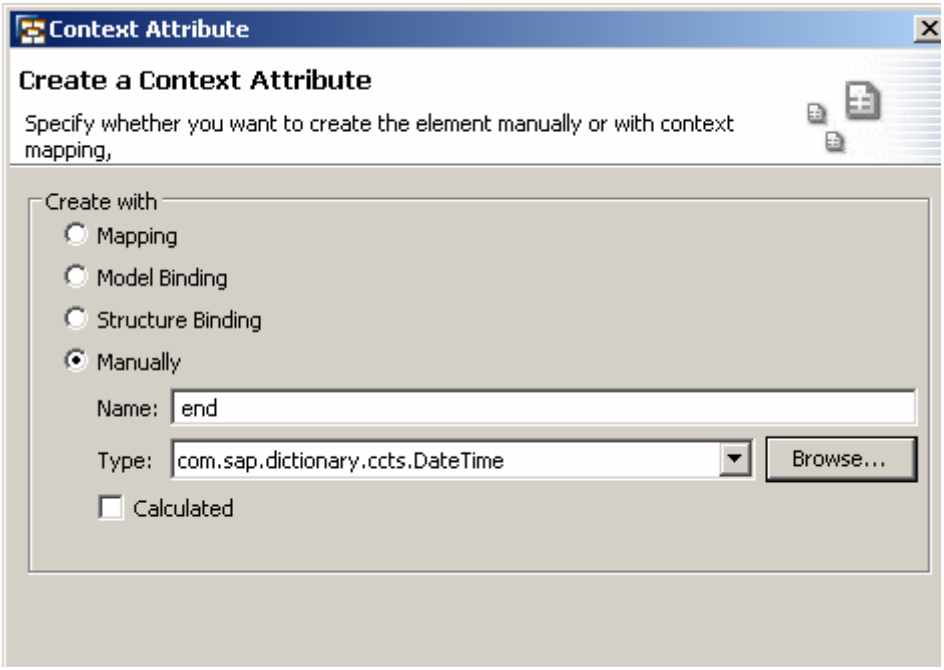


Creating the Context

The team calendar is based on the tree node principle, displaying the various team members in a vertical list. To enable the team calendar display, a recursion node is therefore required in the context. Since every member has his/her own entries and properties, the singleton property must be observed, as there is a certain relationship between the parent nodes and child nodes.

Creating the Context for the *TeamNode*

1. Switch to the *Context* tab and make a right click on the *Context* and choose *New → Node* from the context menu. Select *Manually* and enter **Team** as the name
2. Make a right click on the *Team* node and choose *New → Recursion Node*. Select *Manually*, enter **Child** as the name and choose the *Browse…* button. Select the *Team* node and click *Finish*.
3. Create a new node with ID **Entries** in the *Team* node.
4. Select the Entries node and change the *Singleton* property to **false** in the *Properties* view.
5. To create an attribute in the *Entries* node, open the context menu on *Entries* and choose *New → Attribute*. Select *Manually* and enter **additionalText** as the name and **string** as the type.
6. In the *Entries* node, create another *attribute* and enter **end** as the name. To specify the type, click the **Browse...** button and select *Core Data Type* and subsequently *DateTime*.

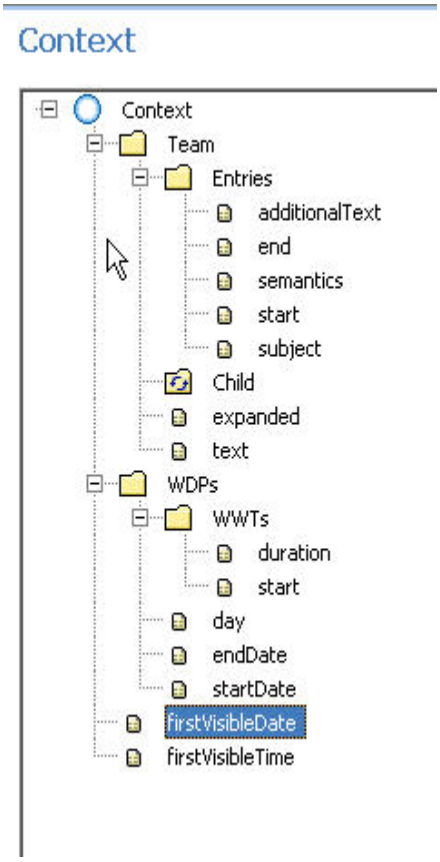


7. Repeat the last step to create another attribute named **start** of type *DateTime*
8. Create two more attributes in the Entries node:
    - **semantics** of type *TableCellDesign.* This is a *Simple Type* located in folder *com.sap.ide.webdynpro.uielementdefinitions*.
    - **subject** of type *string*
9. Insert the following two attributes into the *Team* node:
    - **expanded** of type *boolean*
    - **text** of type *string*

Creating the Context for the *WeekDayPatternNode*

1. Create a **WDPs** node directly below the Context root and insert the following attributes:
    - **day** of type *com.sap.ide.webdynpro.uielementdefinitions.DayOfWeek*
    - **endDate** of type *com.sap.dictionary.ccts.Date*
    - **startDate** of type *com.sap.dictionary.ccts.Date*
2. Insert a node called **WWTs** into the *WDPs* node
3. Create the following attributes inside the *WWTs* node:
    - **duration** of type *com.sap.dictionary.ccts.Duration*
    - **start** of type *com.sap.dictionary.ccts.Time*
4. Select the Context node and create the following attributes directly below the root node.
    - **firstVisibleDate** of type *com.sap.dictionary.ccts.Date*
    - **firstVisibleTime** of type *com.sap.dictionary.ccts.Time*

Your context is now built up and looks like the figure below:



Data Binding

To fill the team calendar display correctly and to obtain the entries from it, you need to bind the UI elements from the layout to the relevant context elements. The process of binding the UI elements to the context is known as data binding.

In the *CalendarView*, switch to the *Layout* tab and map the properties of the various UI elements to the relevant context elements.

TeamCalendar:

1. Click on UI element *TeamCalendar* in the Outline view and switch to the *Properties* tab in the lower screen section.
2. Select the *dataSource* property and click on *Bind*. Select the *Team* folder and confirm with *OK*.
3. Repeat this procedure for every element listed in the table below.

Property	**Value**
TeamCalendar	
dataSource	Team
expanded	Team.expanded
firstVisibleDate	firstVisibleDate
text	Team.text
WeekDayPattern	
dataSource	WDPs
day	WDPs.day
endDate	WDPs.endDate

startDate	WDPs.startDate
WorkingTime	
duration	WDPs.WWTs.duration
start	WDPs.WWTs.start
Entry	
additionalText	Team.Entries.additionalText
dataSource	Team.Entries
end	Team.Entries.end
semantics	Team.Entries.semantics
start	Team.Entries.start
subject	Team.Entries.subject

GroupWeekCalendar:

1. Click the *GroupWeekCalendar_Header* element and bind the *text* property to **Team.text**. The user can then see the name of the selected team member as title of the *CalendarWeekView*.

CalendarWeekView:

1. Click the *CalendarWeekView* element and bind its property elements to the context elements in the table below.

Property	**Value**
CalendarWeekView	
firstVisibleTime	firstVisibleTime
firstVisibleDate	firstVisibleDate
WeekDayPattern	
dataSource	WDPs
day	WDPs.day
endDate	WDPs.endDate
startDate	WDPs.startDate
WorkingTime	
duration	WDPs.WWTs.duration
start	WDPs.WWTs.start
Entry	
additionalText	Team.Entries.additionalText
dataSource	Team.Entries
end	Team.Entries.end
semantics	Team.Entries.semantics
start	Team.Entries.start
subject	Team.Entries.subject

Filling the Calendar with Some Data

To be able to initialize the calendar date and fill the calendar, this calendar first needs to be implemented in method **`wdDoInit()`.**

The source code for this is too long and complicated, however. We therefore call method **`fillCalendar()`** that has been specially created for this purpose. This calls further methods that contain certain parts of the implementation. The way in which the implementation is split up serves to enhance the transparency and reusability of the various elements.

1. To create the method `fillCalendar()` and the other helper methods used inside, switch to the *Java Editor*. If the *Java Editor* is not open, select the view, open the context menu and choose

Open -> Java Editor. There is a section called *others*, where you insert the following methods and arrays between *//@@begin others* and *//@@end*:

```
private void fillCalendar() {
   // initialize Calendar date and time initialize
   final Locale locale = WDResourceHandler.getCurrentSessionLocale();
   TimeZone tz = TimeZone.getDefault();
   CctCode timeZoneCode = new CctCode("CET", null, null, null, null, null);
   Calendar calendar = Calendar.getInstance().getInstance(locale);
   CctDate firstVisibleDate = new CctDate(new
      java.sql.Date(calendar.getTimeInMillis()));
   Time myTime = Time.valueOf("08:00:00");
   CctTime firstVisibleTime = new CctTime(myTime);
   // determine the startdate and starttime for the calendar
   wdContext.currentContextElement().setFirstVisibleDate(firstVisibleDate);
   wdContext.currentContextElement().setFirstVisibleTime(firstVisibleTime);
   // create Teammember
   String names[] = new String[5];
   names[0] = "Nicole Bill";
   names[1] = "Stefanie Miller";
   names[2] = "Marco Hanson";
   names[3] = "Peter Black";
   names[4] = "Martin Hoover";
   // create member names
   for (int j = 0; j < 5; j++) {
      createName(names[j]);
      fillEntries(j, timeZoneCode);
   }
}
```

```
private void createName(String name) {
  // add new member to Team node
  IPrivateCalendarView.ITeamElement team =
      wdContext.createAndAddTeamElement();
  team.setText(name);
  team.setExpanded(false);
}
```

```
private void fillEntries(int h, CctCode tzc) {
  switch (h) {
  case 0:
     for (int m = 0; m < 3; m++) {
     CctDateTime startDate = new CctDateTime(calendarEntries[m][0], tzc,
       false);
     CctDateTime endDate = new CctDateTime(calendarEntries[m][1], tzc,
       false);
     WDTableCellDesign cellColour = WDTableCellDesign.POSITIVE;
     createEntry(startDate, endDate, entrySubject[m][0],
       entrySubject[m][1], cellColour, h);
     }
     break;
  case 1:
     for (int k = 3; k < 5; k++) {
       CctDateTime startDate = new CctDateTime(calendarEntries[k][0], tzc,
          false);
```

```
        CctDateTime endDate = new CctDateTime(calendarEntries[k][1], tzc,
           false);
        WDTableCellDesign cellColour = WDTableCellDesign.CALENDAR_AQUA;
        createEntry(startDate, endDate, entrySubject[k][0],
           entrySubject[k][1], cellColour, h);
     }
     break;
  case 2:
     for (int l = 5; l < 8; l++) {
        WDTableCellDesign cellColour = WDTableCellDesign.CALENDAR_GREEN;
        CctDateTime startDate = new CctDateTime(calendarEntries[l][0], tzc,
           false);
        CctDateTime endDate = new CctDateTime(calendarEntries[l][1], tzc,
           false);
        createEntry(startDate, endDate, entrySubject[l][0],
           entrySubject[l][1], cellColour, h);
     }
     break;
  case 3:
     for (int l = 8; l < 11; l++) {
        WDTableCellDesign cellColour = WDTableCellDesign.CALENDAR_METAL;
        CctDateTime startDate = new CctDateTime(calendarEntries[l][0], tzc,
           false);
        CctDateTime endDate = new CctDateTime(calendarEntries[l][1], tzc,
           false);
        createEntry(startDate, endDate, entrySubject[l][0],
           entrySubject[l][1], cellColour, h);
     }
     break;
  case 4:
     for (int l = 11; l < 13; l++) {
        WDTableCellDesign cellColour = WDTableCellDesign.CALENDAR_TEAL;
        CctDateTime startDate = new CctDateTime(calendarEntries[l][0], tzc,
           false);
        CctDateTime endDate = new CctDateTime(calendarEntries[l][1], tzc,
           false);
        createEntry(startDate, endDate, entrySubject[l][0],
           entrySubject[l][1], cellColour, h);
     }
     break;
  }
}
```

2. To have some data, create these two arrays, again in the `others` section:

```
private Timestamp calendarEntries[][] = new Timestamp[13][2];
  {
     // dates for the different team members
     calendarEntries[0][0] = Timestamp.valueOf("2007-09-10 10:00:00.000");
     calendarEntries[0][1] = Timestamp.valueOf("2007-09-11 13:00:00.000");
     calendarEntries[1][0] = Timestamp.valueOf("2007-09-13 06:45:00.000");
     calendarEntries[1][1] = Timestamp.valueOf("2007-09-14 11:00:00.000");
     calendarEntries[2][0] = Timestamp.valueOf("2007-09-17 05:00:00.000");
     calendarEntries[2][1] = Timestamp.valueOf("2007-09-17 18:00:00.000");
     calendarEntries[3][0] = Timestamp.valueOf("2007-09-09 15:00:00.000");
     calendarEntries[3][1] = Timestamp.valueOf("2007-09-09 15:30:00.000");
     calendarEntries[4][0] = Timestamp.valueOf("2007-09-11 10:00:00.000");
     calendarEntries[4][1] = Timestamp.valueOf("2007-09-12 12:00:00.000");
     calendarEntries[5][0] = Timestamp.valueOf("2007-09-21 15:00:00.000");
```

```
      calendarEntries[5][1] = Timestamp.valueOf("2007-09-27 15:30:00.000");
      calendarEntries[6][0] = Timestamp.valueOf("2007-09-26 10:00:00.000");
      calendarEntries[6][1] = Timestamp.valueOf("2007-09-26 12:00:00.000");
      calendarEntries[7][0] = Timestamp.valueOf("2007-09-17 13:00:00.000");
      calendarEntries[7][1] = Timestamp.valueOf("2007-09-17 15:30:00.000");
      calendarEntries[8][0] = Timestamp.valueOf("2007-09-02 12:00:00.000");
      calendarEntries[8][1] = Timestamp.valueOf("2007-09-03 12:15:00.000");
      calendarEntries[9][0] = Timestamp.valueOf("2007-09-17 16:00:00.000");
      calendarEntries[9][1] = Timestamp.valueOf("2007-09-17 17:00:00.000");
      calendarEntries[10][0] = Timestamp.valueOf("2007-09-17 17:45:00.000");
      calendarEntries[10][1] = Timestamp.valueOf("2007-09-17 19:00:00.000");
      calendarEntries[11][0] = Timestamp.valueOf("2007-09-17 08:00:00.000");
      calendarEntries[11][1] = Timestamp.valueOf("2007-09-18 17:00:00.000");
      calendarEntries[12][0] = Timestamp.valueOf("2007-09-19 13:00:00.000");
      calendarEntries[12][1] = Timestamp.valueOf("2007-09-19 14:30:00.000");
  }
  private String entrySubject[][] = new String[13][2];
  {
      // Subjects for the different calendar entries
      entrySubject[0][0] = "Team meeting";
      entrySubject[0][1] = "Walldorf";
      entrySubject[1][0] = "Training";
      entrySubject[1][1] = "Heidelberg";
      entrySubject[2][0] = "Kick off";
      entrySubject[2][1] = "Palo Alto";
      entrySubject[3][0] = "Jour Fixe";
      entrySubject[3][1] = "Walldorf, WDF43, E0.4";
      entrySubject[4][0] = "Meeting";
      entrySubject[4][1] = "A1.06";
      entrySubject[5][0] = "Private Appointment";
      entrySubject[5][1] = "Coffee Corner";
      entrySubject[6][0] = "anniversary";
      entrySubject[6][1] = "Rot22 (E6.09)";
      entrySubject[7][0] = "Lunch";
      entrySubject[7][1] = "Canteen";
      entrySubject[8][0] = "Meeting discussion Roll Out";
      entrySubject[8][1] = "coffee corner";
      entrySubject[9][0] = "One to One";
      entrySubject[9][1] = "ABC 345, H2, 23";
      entrySubject[10][0] = "Teamevent";
      entrySubject[10][1] = "Walldorf";
      entrySubject[11][0] = "Meeting";
      entrySubject[11][1] = "Hamburg";
      entrySubject[12][0] = "Training";
      entrySubject[12][1] = "BAC";
  }
```

5. To define a daily working time – in this case 9 hours starting at 8.00 o'clock – add the following method again to the *others* section:

```
private void createWT() {
  // determine start-and End dates for Workingtime
  // WDPs
  Calendar endC = Calendar.getInstance();
  endC.set(2010, Calendar.JANUARY, 1, 8, 0, 0);
  CctDate endDate = new CctDate(new
    java.sql.Date(endC.getTimeInMillis()));
  Calendar startC = Calendar.getInstance();
  startC.set(2007, Calendar.JANUARY, 1, 8, 0, 0);
```

```
  CctDate startDate = new CctDate(new
    java.sql.Date(startC.getTimeInMillis()));
  // determine starttime and duration for Workingtime
  // WWTs
  CctDuration duration = new CctDuration("P0Y0M0DT9H0M0S");
  Time myTime = Time.valueOf("08:00:00");
  CctTime startWT = new CctTime(myTime);
  // Context
  // WDPs
  for (int i = 2; i < 7; i++) {
    IPrivateCalendarView.IWDPsElement newWDPSElement =
       wdContext.createAndAddWDPsElement();
    newWDPSElement.setDay(WDDayOfWeek.valueOf(i));
    newWDPSElement.setStartDate(startDate);
    newWDPSElement.setEndDate(endDate);
  }
  // WWts
  IPrivateCalendarView.IWWTsElement newWWTsElement =
  wdContext.createAndAddWWTsElement();
  newWWTsElement.setDuration(duration);
  newWWTsElement.setStart(startWT);
  wdContext.nodeTeam().nodeEntries().setLeadSelection(-1);
}
```

```
      private void createEntry(CctDateTime start, CctDateTime end, String
subject, String additionalText,
                  WDTableCellDesign colour, int h) {
            // create new Entries and add to nodeEntries
            wdContext.nodeTeam().setLeadSelection(h);
            IPrivateCalendarView.IEntriesElement newEntryElement =
wdContext.createAndAddEntriesElement();
            newEntryElement.setStart(start);
            newEntryElement.setEnd(end);
            newEntryElement.setSemantics(colour);
            newEntryElement.setSubject(subject);
            newEntryElement.setAdditionalText(additionalText);
            wdContext.nodeTeam().nodeEntries().setLeadSelection(-1);
  }
```

3. To call these methods, write the following code in the `wdDoInit()` method:

```
  public void wdDoInit()
  {
    //@@begin wdDoInit()
         fillCalendar();
         createWT();
    //@@end
  }
```

To remove errors – if you got some displayed -, open the context menu and choose *Source -> Organize Imports*

Creating and Implementing an Action

To coordinate how the entries and team members are displayed in the *CalendarWeekView*, you need an action that updates the start date for the *WeekCalendar* and sets the selected date in the team calendar. This action also has to display the entries for the selected team member in the calendar. It needs to call these from the index for the team member.

Creating an Action and Mapping an Event

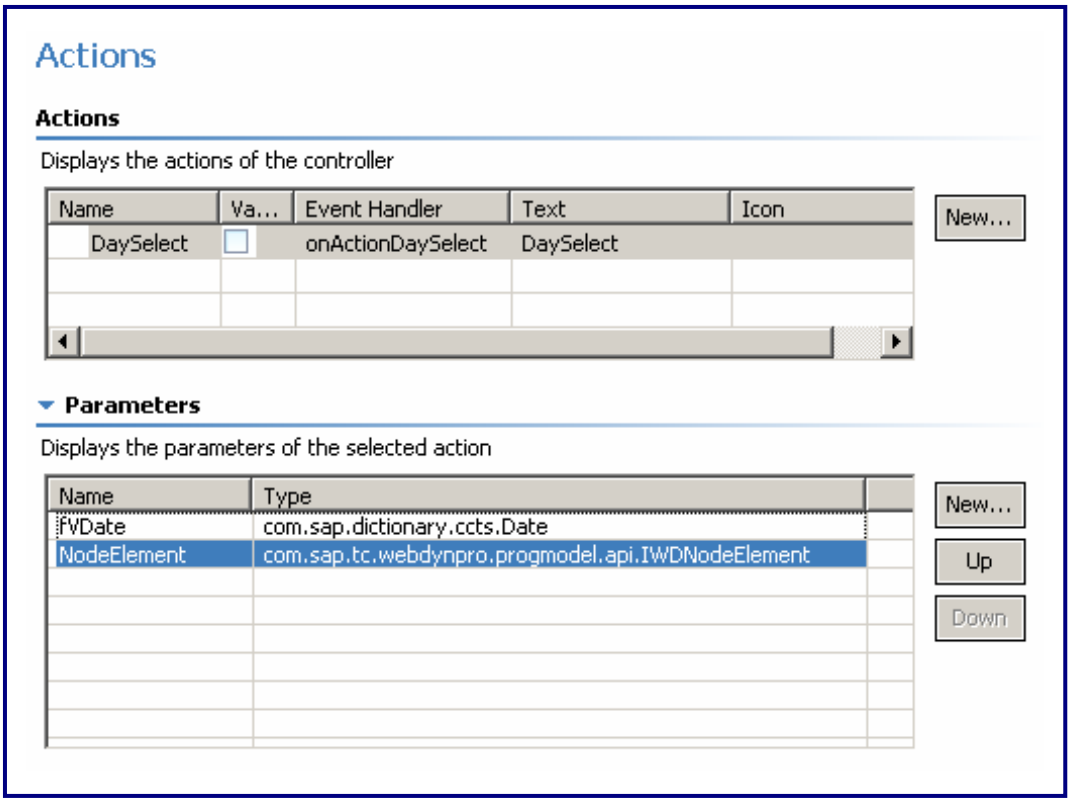
As the starting date selected from the team calendar is required in order to display the *CalendarWeekView* properly, a number of parameters need to be integrated in this action beforehand. You can do this in the event handler.

When the date is selected from the team calendar, the action must place this date in the parameter that has been created. This allows you to define this date as the starting date for the *CalendarWeekView*. This process is referred to as parameter mapping. To obtain the precise element without having to change the lead selection in the node entries, you can use the *Event* parameter. This parameter is used here to obtain and forward the selected date.

When the date is selected from the team calendar, this action also needs to be able to recognize which team member has been selected. It can be recognized and defined using the generic event parameter *nodeElement*. Here, you store the index for the selected node *Team* in the *nodeElement* parameter and can then call it in the action and set it as the lead selection for the *CalendarWeekView*. This ensures that only entries that are stored for this team member will be displayed.

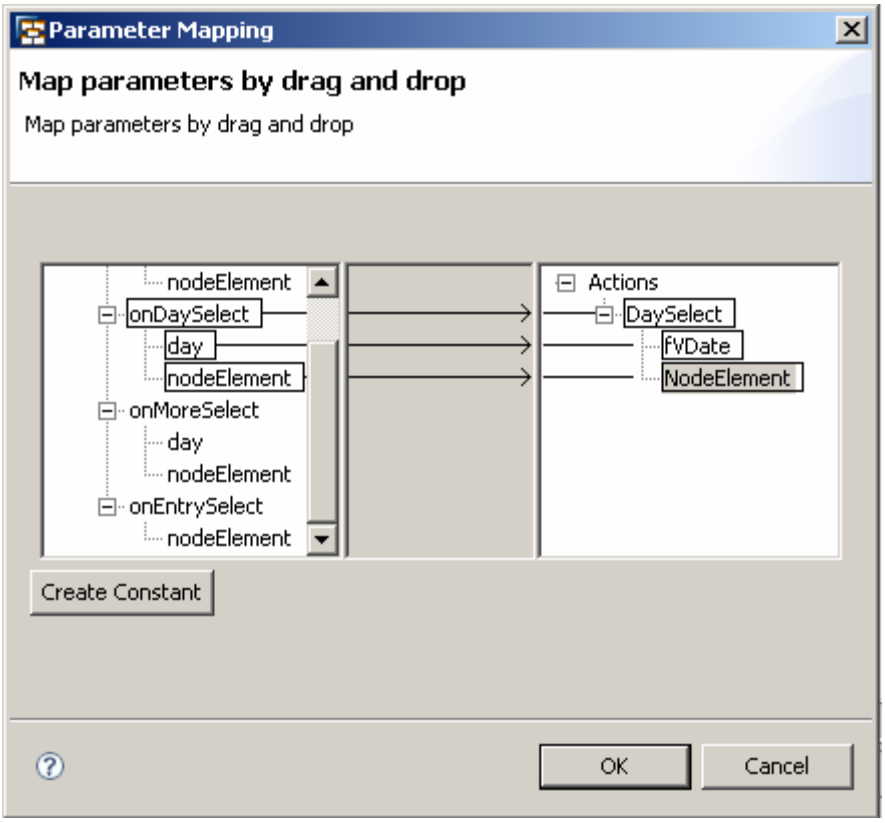
Create an Action

1. To call the *CalendarView* in View Designer, double-click the *CalendarView* node in the project.
2. Select the *Action* tab.
3. Press *New*. You can create a new action in the help window.
4. Enter **DaySelect** as the name for the new action. Quit the event handler using the standard options. Now click *Next* and press *New* in the next window.
5. Enter **fVDate** as the *name*.
6. Press *Browse…,* select *Core Date Type* and subsequently *Date*.
7. Create another parameter called **nodeElement**. Press *Browse…,* select *Java Native Type*, click *Browse...*, enter *IWDNodeElement* and confirm. The new action - **DaySelect** – and its event handler - **onActionDaySelect** – are now displayed in the action list.



Mapping the Action to the Event:

1. Switch to the *Layout* tab and select the *TeamCalendar* element in the *Outline* view.
2. Select the *Properties* tab on the bottom edge and map the event **onDaySelect** to action handler *DaySelect*. If the event handler does not appear in the drop down list, save your data and try again.
3. Make a right click in the *Outline* View on UI element *TeamCalendar* and choose *Parameter Mapping.*
4. Using drag and drop, map action parameter **fVDate** to *TeamCalendar* parameter **day** and map action parameter **nodeElement** to *TeamCalendar* parameter **nodeElement.**



Implementing the Action

1. Switch to the Java Editor, navigate to the *onActionDaySelect* event handler and insert the following code:

```
public void onActionDaySelect
      (com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent,
      com.sap.dictionary.runtime.container.CctDate fVDate,
      com.sap.tc.webdynpro.progmodel.api.IWDNodeElement NodeElement){
      //@@begin onActionDaySelect(ServerEvent)
      // set firstVisibleDate = selected Date  and set Index for the
            selected Member
      wdContext.currentContextElement().setFirstVisibleDate(fVDate);
      wdContext.nodeTeam().setLeadSelection(nodeElement.index());
      //@@end
 }
```

Implementing the *CalendarPaginator*

To facilitate navigation between months in the team calendar, an additional UI element can be implemented. This UI element is known as the *CalendarPaginator*. It has two different events that move either one month forward or one month back, depending on how the corresponding action was implemented.

Inserting the *CalendarPaginator* in the Layout

1. Go back to the View Editor and select the *Layout* tab.

2. Make a right click on the *GroupTeamCalendar* element and choose *Insert Toolbar*.

3. Make a right click on the *Toolbar* UI element, choose *Insert ToolBarItem* and select the *CalendarPaginator*.

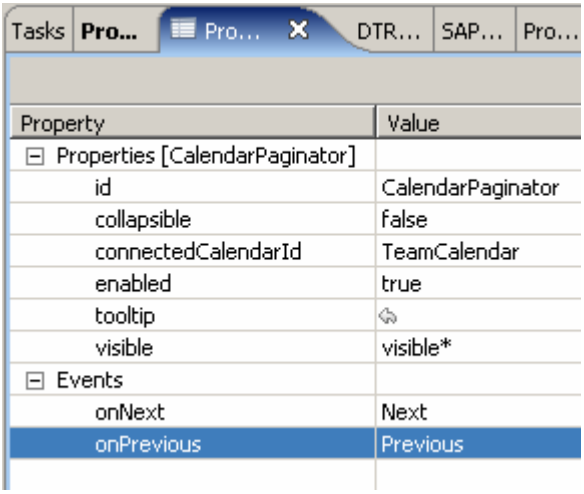
4. Switch to the *Properties* tab in the lower display area and enter **TeamCalendar** as *connectedCalendarId*. The value must be exactly the id property of your team calendar.

Caution: Do not be surprised if *CalendarPaginator: rendering problem: null* appears in the layout display. Continue as described below.

Imlementing Actions for the *CalendarPaginator*

1. Switch to the *Action* tab and create two new actions called **Next** and **Previous**.

2. Switch to the *Layout* tab, select the *CalendarPaginator* and bind the events *onNext* and *onPrevious* to the corresponding event handlers.



3. Switch to the Java Editor, navigate to `onActionNext()` and insert the following code:

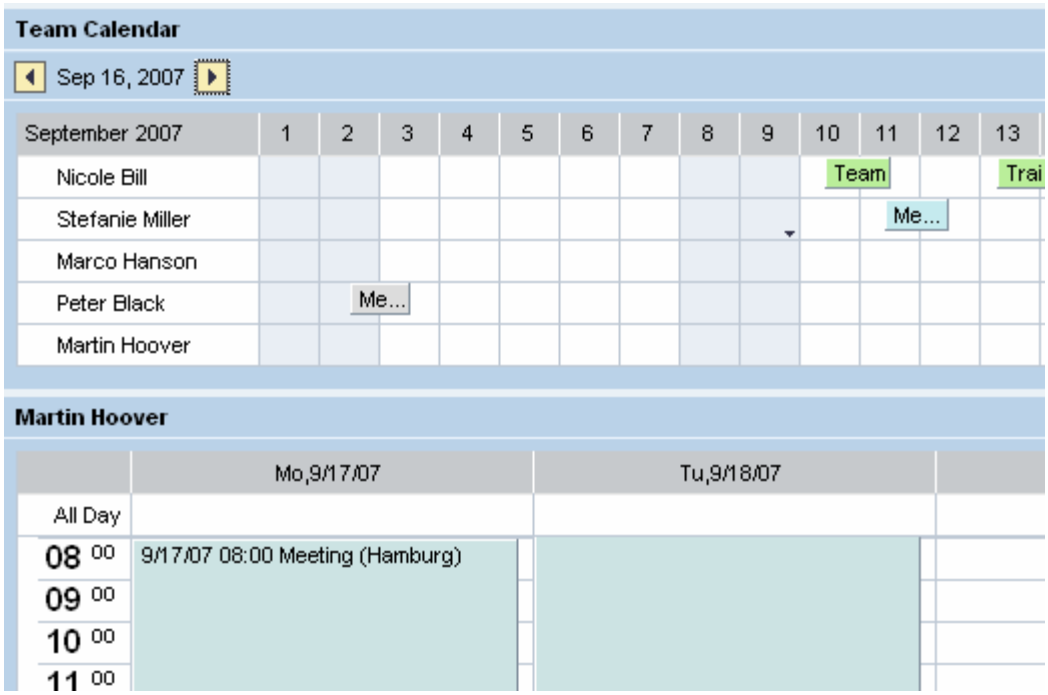
```
public void onActionNext(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent
wdEvent )
  {
  //@@begin onActionNext(ServerEvent)
  CctDate firstVisibleDate =
    wdContext.currentContextElement().getFirstVisibleDate();
  Calendar calendar = Calendar.getInstance();
  calendar.setTimeInMillis(firstVisibleDate.getContent().getTime());
  calendar.add(Calendar.MONTH, 1);
  CctDate newDate = new CctDate(new Date(calendar.getTimeInMillis()));
```

```
  wdContext.currentContextElement().setFirstVisibleDate(newDate);
  //@@end
}
```

4. Choose *Source -> Organize Imports* from the context menu and select **java.sql.Date**
5. Navigate to `onActionPrevious()` and insert the following code:

```
public void onActionPrevious(
       com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent ){
  //@@begin onActionPrevious(ServerEvent)
  CctDate firstVisibleDate =
    wdContext.currentContextElement().getFirstVisibleDate();
   Calendar calendar = Calendar.getInstance();
   calendar.setTimeInMillis(firstVisibleDate.getContent().getTime());
   calendar.add(Calendar.MONTH, -1);
   CctDate newDate = new CctDate(new Date(calendar.getTimeInMillis()));
   wdContext.currentContextElement().setFirstVisibleDate(newDate);
   //@@end
 }
```

Now the first big steps to implement a Calendar are done. To test your application, you can choose *Deploy new Archive and Run* from the context menu of the **CalendarApp**.



The team calendar displays the various team members. By clicking on the day in the upper team calendar, you can display the selected team member and the corresponding time in more detail in the corresponding row in the lower weekly calendar. Depending on your selection, this therefore allows you to switch the member, the date, or both. Below the title of the team calendar, you will find the toolbar with the *CalendarPaginator* that allows navigation in the team calendar. Using this element, you can jump either a month forward or a month back in the team calendar. Navigating to September of 2007 will display the entries.

Now we suggest that you have a break, before you carry on implementing this tutorial ;-)

Editing Entries using Popup Windows

In this part of the tutorial you will implement the ability to create, edit and delete entries using popups. You will enhance the context and map the view contexts to the controller context to be able to retrieve the data from different views.

Enhancing the Context

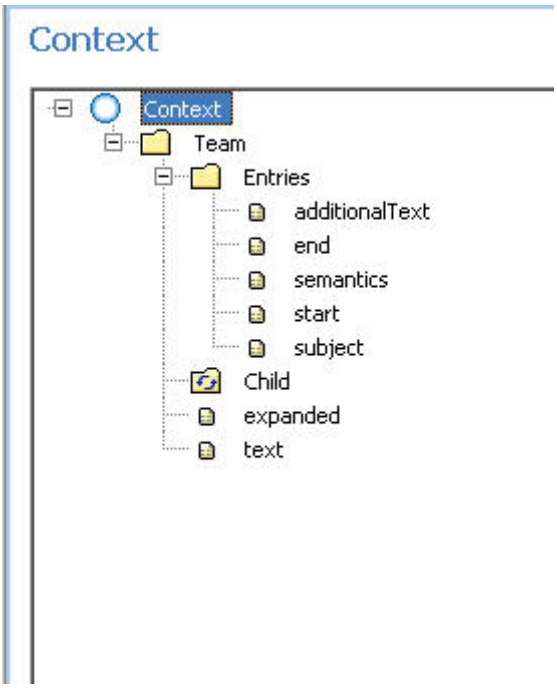
To create, change or delete entries in the calendar, you need to create a temporary storage location. After confirmation by choosing *Save*, the action is triggered that updates or replaces the relevant entry in the main node with the data in the temporary storage location.

To allow the entry in the calendar to be updated, it needs to be addressed by its index. This means that both the temporary storage location and an attribute need to be created for the index.

To navigate with the popup windows, you need an attribute that can store and replace the current window.

Creating the Context in the Component Controller

1. Open the *Context* tab in the *CalendarView*.

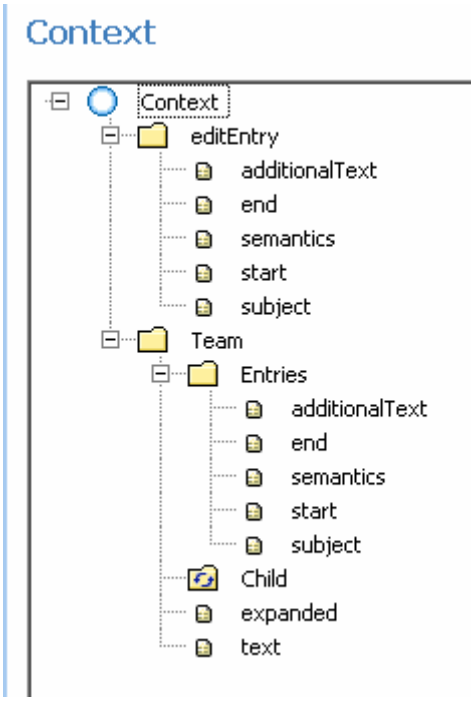


2. Make a right click on the *Team* node and select *Copy* from the context menu. Switch to the *Component Controller* and open the *Context* tab. Make a right click on the *Context* and select *Paste* from the help menu.

3. Create a new *Node* in the Component Controller under the Context node called **editEntry** with the following attributes:

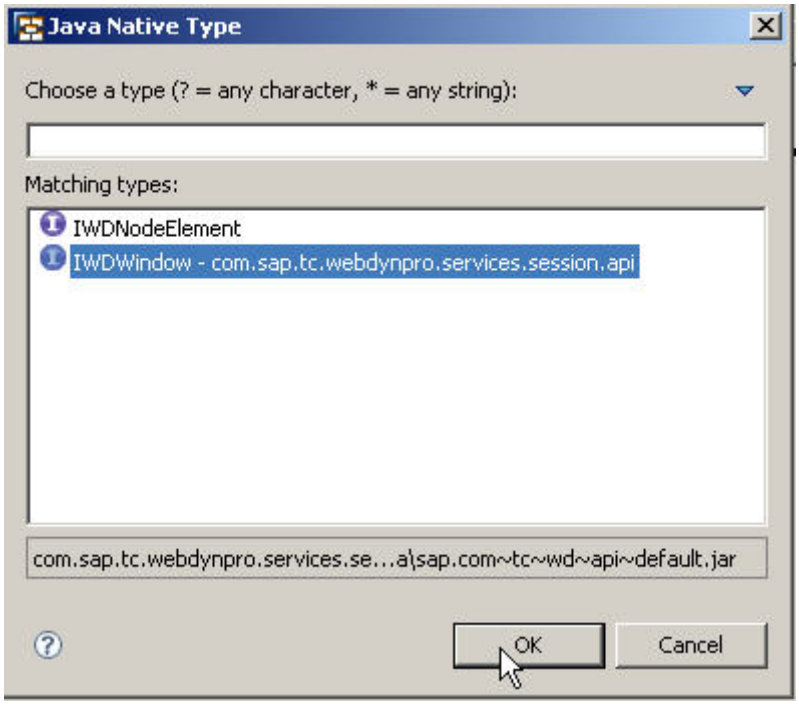
Name	Type
additionalText	string
end	com.sap.dictionary.ccts.DateTime
semantics	com.sap.ide.webdynpro.uielementdefinitions.TableCellDesign
start	com.sap.dictionary.ccts.DateTime
subject	string

4. Change the *Collection Cardinality* property for the *editEntry* node to *1…n* and save.

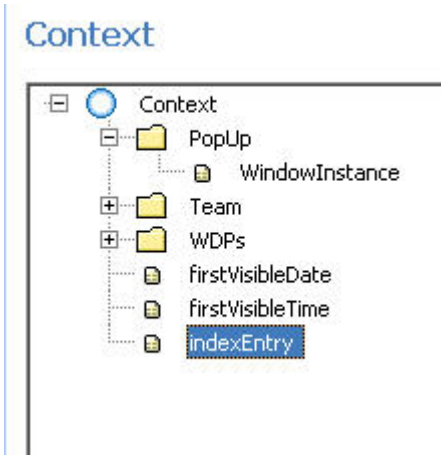


Creating the Context in *CalendarView*:

1. Switch to the *Context* tab in the *CalendarView*.
2. Create a new *node* and name it **PopUp**. Change the *Collection Cardinality* property to *1...1*.
3. Create an attribute here with the *name* **WindowInstance**. Press *Browse...* and select *Java Native Type* from the type selection.
4. Press *Browse…* and enter **IWDWindow**. Select *IWDWindow* and quit the menu by choosing *OK.*



5. Create a new attribute for the context. Give this attribute the name **indexEntry** and select *type* **integer** and confirm.

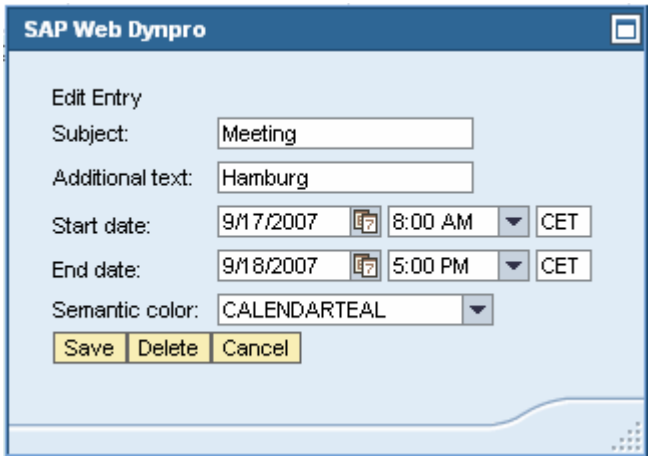


Creating New Views and Windows

To make the data change in different popups, you need to create two views and windows. Once you have created these elements, the various views are embedded into the relevant windows.

The popup is called by an action that coordinates calling the relevant window. To close the window, however, an event is required, as the called window can only be destroyed by its parent component (the calling window). Event handling is therefore required in the child component. This can use server-side events to forward the event handling to the Component Controller. It is therefore possible to subscribe the event handler to an event.

The figure below shows the *EditEntryWindow*, built up of the *EditEntryView*.

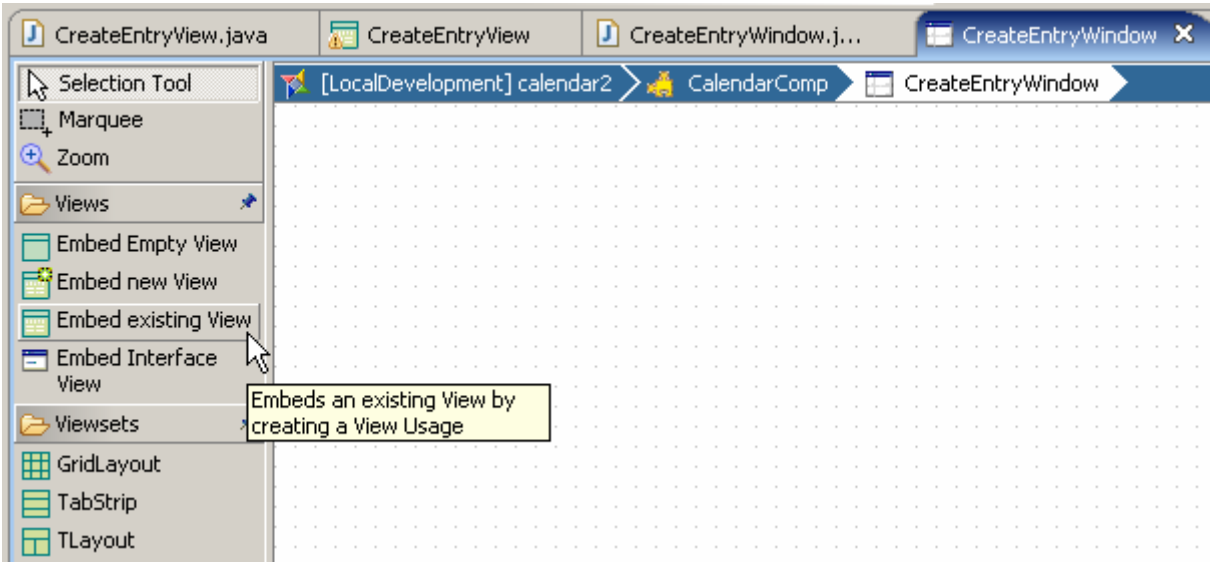


Creating New Views:

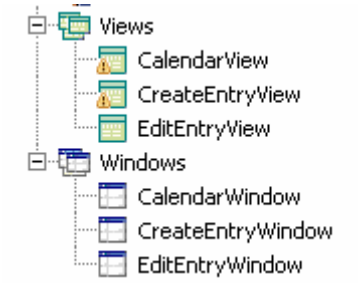
1. In *Web Dynpro Explorer*, make a right click on *Views* and select *Create View*.
2. Enter **EditEntryView** as the *name*. Confirm by choosing *Finish.* Confirm the next window too by choosing *OK*.
3. Create another view and name it **CreateEntryView**.

Creating New Windows and Assigning the Corresponding Views

1. In *Web Dynpro Explorer*, make a right click on *Windows* and choose *Create Window*.
2. Enter **EditEntryWindow** as the name and click on *Finish.*
3. Now go to *Embed Existing View* on the window that now opens and draw a rectangle in the empty field in the middle.



4. Select *EditEntryView* and confirm.
5. Repeat this procedure and create a window called **CreateEntryWindow** and embed the *CreateEntryView* into.

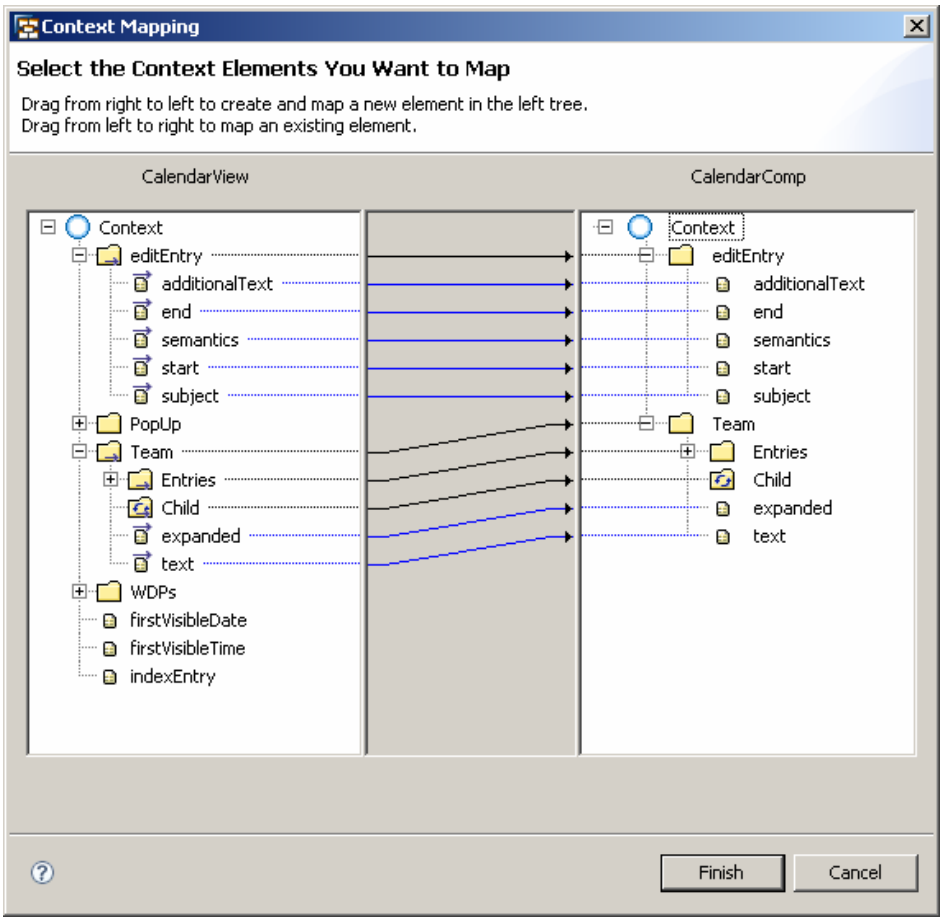


Context Mapping between the Component Controller and the Views

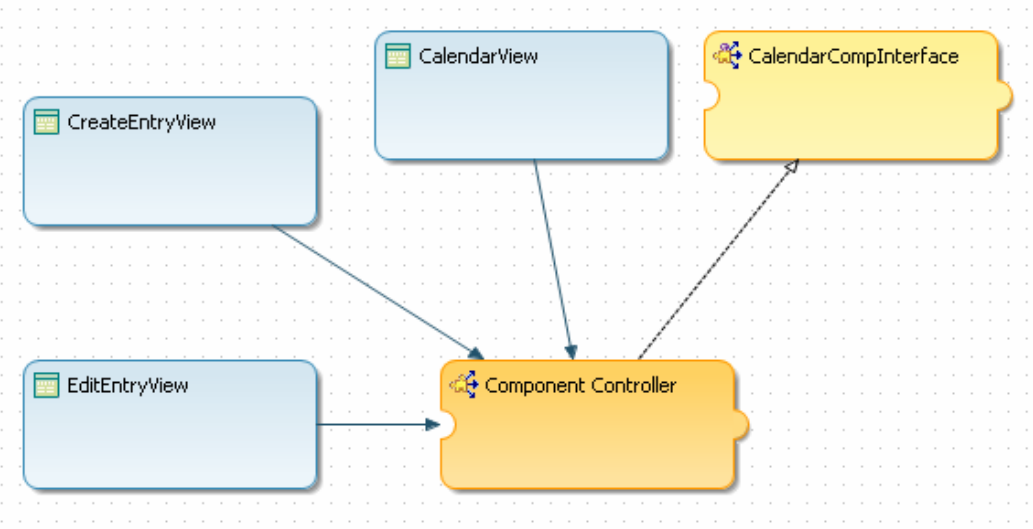
When new entries are created, or existing entries are changed, the data is placed in a temporary storage location (the *editEntry* node you just created) and is not passed on to the main storage location (the *Team* node and the *Entries* node below) until the user confirms.

Creation of new data or modification of existing data takes place in separate views. To allow this data to be passed on to the main nodes in the Component Controller, these views must be mapped with their contexts and the Component Controller context.

1. Double-click the *CalendarComp* in *Web Dynpro Explorer*.
2. Click on the *Create Data Link* arrow and draw a line from the *CalendarView* to the *Component Controller*. To do this, click on the *CalendarView*, drag and drop the line to the *Component Controller* and release the button.
3. In the window that now opens, drag and drop a line from the *Team* node in the *CalendarView* to the *Team* node in the *Component Controller*. Falsche Richtung!
4. Select all fields in the window that now appears and confirm by pressing *Finish*.
5. Now drag the *editEntry* nodes to the *CalendarView* context.
6. Confirm by pressing *Finish* and save your data.



7. Draw a Data Link from the *EditEntryView* to the Component Controller and map the *editEntry* node.
8. Repeat this procedure for the *CreateEntryView* and map the *editEntry* and *Team* nodes.



Layout

In this tutorial, a number of elements are added to the team calendar. This allows you to make subsequent changes to entries.

These events take place in popups. The layout for these popups is created by templates. This procedure simplifies the process of creating the individual layouts in the various views.

Each newly created view still contains buttons that are bound to the various actions. These actions are used to save, delete or create data.

Designing the *EditEntryView*

1. Open the *EditEntryView* and switch to the *Outline* tab.
2. Change the property *text* for the *DefaultTextView* to **Edit Entry.**
3. Open the ccontext menu on the RootElement and choose Apply Template.
4. Select Form, confirm with *Next* >, select the *editEntry* node and click *Next* > again.
5. Arrange the UI elements to the following sequence:
    - *subject*
    - *additionalText*
    - *start*
    - *end*
    - *semantics*
6. Enter the corresponding texts to the respective labels.

7. In the *RootElement*, create three buttons and call them **Save, Delete** and **Cancel**. Designing the CreateEntryView

1. Repeat this procedure for the *CreateEntryView.*
2. Change the property text for the *DefaultTextView* element to **Create Entry.**
3. Create a template called *Form* for the *editEntry* node and arrange the UI elements as you did in the *EditEntryView*.
4. Enter the corresponding texts to the respective labels.
5. In *TransContainer_0*, insert
    - A *DropDownByIndex* with ID **memberName**. Bind the `texts` property to the *text* attribute inside the *Team* node.
    - A *Label* and set its *text* property to **name** and its *layoutData* to **MatrixHeadData** and the *labelFor* property to **memberName**.

6. In the *RootElement*, create two buttons and call them **Save** and **Cancel**.

Enhancing the CalendarView Layout

1. Open the *CalendarView* and insert a *ToolBarItem* of type *ToolBarButton* to the toolbar.

Actions, Events and Methods

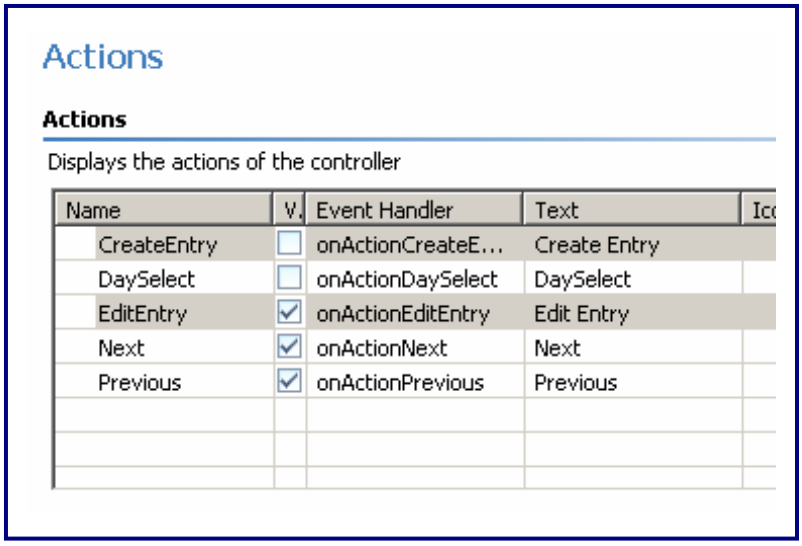
Actions are required in order to open popups for editing data after an event. An example of this kind of event is when an entry or member is selected in a calendar, or when a button is pressed.

Actions are also required in order to close the popup again. The action in the view controller triggers a method in the Component Controller. The method in the Component Controller then triggers an event again. This event triggers an action in the parent view controller, issuing the command to close the popup. This path is needed as only the parent view controller of the popup can issue the command to close this window. It is therefore possible to subscribe the event handler to an event.

To give an example: If the user clicks the *Save* button on the *EditEntryView* (the popup), the *onAction* event is triggered, and the corresponding event handler triggers the event *editEntry* of the *CalendarComp.* In the *CalendarComp* this event just triggers an event in the *CalendarView* controller: Now the *CalendarView* controller "knows" that the *Save* button in the popup was clicked and in the corresponding event handler the popup is closed and the data is written to the context.

Creating Actions in the CalendarView

1. Open the *CalendarView* and switch to the *Action* tab.
2. Create the following actions:
   - *EditEntry* with the text *Edit Entry* and the parameter **nodeElement** of type *IWDNodeElement*
   - *CreateEntry*. This action does not need parameters.

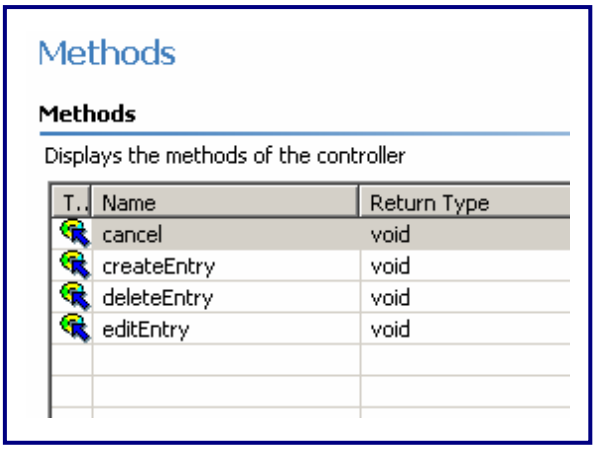


3. Switch to Layout tab and select the *CalendarWeekView in the Outline Tab*.
4. Assign the action *EditEntry* to the event *onEntrySelect*.
5. Assign the action *CreateEntry* to the *onAction* event of the *ToolBarButton*.

Methods and Events in the Component Controller

1. Open the *Methods* tab in the *Component Controller*.
2. Create the following methods, all with *Return Type* **void**:

- **editEntry**
- **createEntry**
- **deleteEntry**
- **cancel**



3. Switch to the *Events* tab and create the following events:
    - **editEntry**
    - **createEntry**
    - **deleteEntry**
    - **cancel**

Creating Actions for the CreateEntryView and Binding them to an Event

4. Switch to the *CreateEntryView,* open the *Actions* tab and create the following actions.
    - **Cancel**
    - **Save**
    - **SelectMember**.
      For this action create a parameter called **index** of type *integer*.
5. Switch to the *Layout* tab and assign the following events to these actions:
    - Event *onAction* of the Button *Cancel* to *Cancel*
    - Event *onAction* of the Button *Save* to *Save*
    - Event *onSelect* of the *DropDownByIndex* called memberName to *SelectMember*

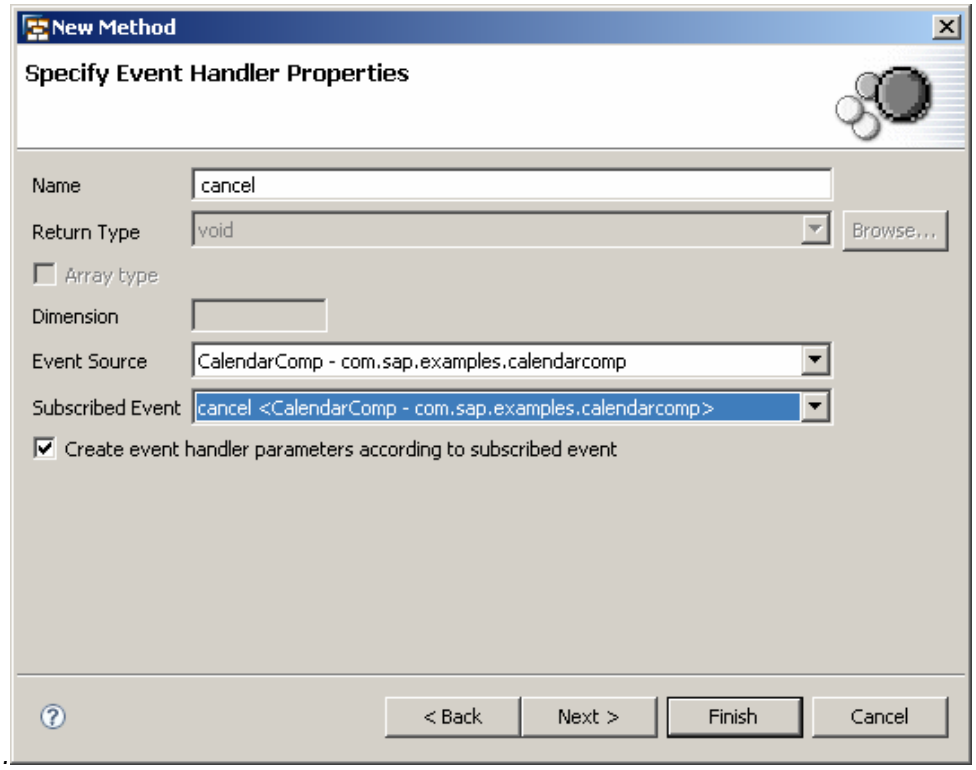
Creating Actions for the EditEntryViews and Binding them to an Event

6. Switch to the *EditEntryView* and open the *Action* tab.
7. Create the following actions:
    - **Delete**
    - **Cancel**
    - **Save**
8. Switch to the *Layout* tab and assign the *onAction* events of the buttons to the respective actions.

Subscribing Events in the *CalendarView*

9. Switch to the *CalendarView* and open the *Methods* tab.

10. Choose *New*, select *Event Handler* and choose *Next*.
11. Enter **cancel** as the *name*.
12. Select **CalendarComp** as the *Event Source* and **cancel** as the *Subscribed Event* and confirm.



13. Repeat this procedure for the following methods and save.

Name	Event Source	Subscribed Event
deleteEntry	CalendarComp	deleteEntry
createEntry	CalendarComp	createEntry
editEntry	CalendarComp	editEntry

Methods

Methods

Displays the methods of the controller

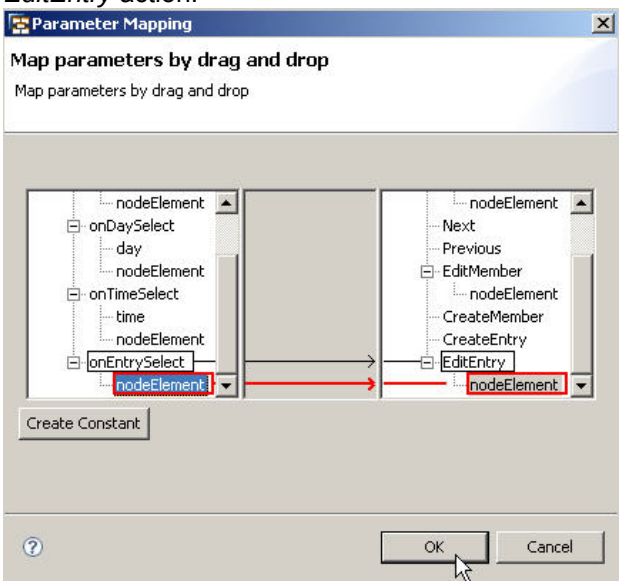
T..	Name	Ret...	Event Source	Subscribed Event
	cancel	void	CalendarComp - com.sap.examples.calendarcomp	cancel <CalendarComp - com.sap.examples.calendarcomp>
	createEntry	void	CalendarComp - com.sap.examples.calendarcomp	createEntry <CalendarComp - com.sap.examples.calendarcomp>
	deleteEntry	void	CalendarComp - com.sap.examples.calendarcomp	deleteEntry <CalendarComp - com.sap.examples.calendarcomp>
	editEntry	void	CalendarComp - com.sap.examples.calendarcomp	editEntry <CalendarComp - com.sap.examples.calendarcomp>
	onActionCreateEntry	void		
	onActionDaySelect	void		
	onActionEditEntry	void		
	onActionNext	void		
	onActionPrevious	void		

Parameter Mapping

To release the selected elements in the calendar, parameter mapping is required. The UI element's *nodeElement* is therefore mapped to the parameters of the action.

Action EditEntry

1. Switch to the *Layout* tab in the *CalendarView*.
2. In the *OutlineView*, make a right click to open the menu for UI element *CalendarWeekView*.
3. Choose *Parameter Mapping*.
4. Assign the *nodeElement* in the *onEntrySelect* event *to* the *nodeElement* parameter in the *EditEntry* action.



Implementation

For the popups to appear, they need to be created in the implementation by a command. The command is written in the implementation of several actions. For the sake of simplicity and better maintenance, a single method is written that can be called up in the various actions.

The process of closing the popups is also written to an extra method. To do this, the system calls up the events and methods that have been created. It is therefore stipulated in the implementation that the action in the child component triggers a method in the Component Controller. When an event is triggered in the Component Controller, this method then calls an action in the parent component in which the command to close the popup is implemented. This path is needed as only the parent component of the popup can issue the command to close this window.

Modification, deletion and creation of the data are also coordinated in the implementation.

CalendarView

1. Switch to the *Java Editor* of the *CalendarView* and insert the following code between *//@@begin others* and *//@@end*:

```
private void destroyWindow()
{
  IWDWindow window = wdContext.currentPopUpElement().getWindowInstance();
  window.destroyInstance();
}
```

```
private void showWindow(String windowName)
{
  IWDWindowInfo windowInfo =
  (IWDWindowInfo)wdThis.wdGetAPI().getComponent().getComponentInfo().findIn
    Windows(windowName);
  IWDWindow window =
```

```
  wdThis.wdGetAPI().getComponent().getWindowManager().createModalWindow(win
  dowInfo);
  wdContext.currentPopUpElement().setWindowInstance(window);
  window.setWindowPosition(300, 150);
  window.show();
}
```

2. Write the following code in the *onActionEditEntry()* action:

```
public void onActionEditEntry
(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent,
com.sap.tc.webdynpro.progmodel.api.IWDNodeElement nodeElement )
{
  //@@begin onActionEditEntry(ServerEvent)
showWindow("EditEntryWindow");
  wdContext.currentEditEntryElement().setAttributeValue("additionalText",
    nodeElement.getAttributeValue("additionalText"));
  wdContext.currentEditEntryElement().setAttributeValue("subject",
    nodeElement.getAttributeValue("subject"));
  wdContext.currentEditEntryElement().setAttributeValue("end",
    nodeElement.getAttributeValue("end"));
  wdContext.currentEditEntryElement().setAttributeValue("start",
    nodeElement.getAttributeValue("start"));
  wdContext.currentEditEntryElement().setAttributeValue("semantics",
    nodeElement.getAttributeValue("semantics"));
  wdContext.currentContextElement().setIndexEntry(nodeElement.index());
  //@@end
}
```

3. Write the following code in the *onActionCreateEntry()* action:

```
public void onActionCreateEntry
(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
  {
  //@@begin onActionCreateEntry(ServerEvent)
  showWindow("CreateEntryWindow");
  wdContext.currentEditEntryElement().setAdditionalText("");
  wdContext.currentEditEntryElement().setSemantics(WDTableCellDesign.STANDA
   RD);
  wdContext.currentEditEntryElement().setSubject("");
  CctCode timeZoneCode = new CctCode("CET", null,null,null,null,null);
  Timestamp timeStamp = Timestamp.valueOf("2007-01-01 08:00:00.000");
  CctDateTime dateTime = new CctDateTime(timeStamp, timeZoneCode, false);
  wdContext.currentEditEntryElement().setEnd(dateTime);
  wdContext.currentEditEntryElement().setStart(dateTime);
  //@@end
}
```

1. Write the following code in the *deleteEntry()* method:

```
public void deleteEntry(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent
wdEvent )
  {
  //@@begin deleteEntry(ServerEvent)
  destroyWindow();
  wdContext.nodeTeam().nodeEntries().removeElement(wdContext.nodeEntries().
     getElementAt(wdContext.currentContextElement().getIndexEntry()));
  //@@end
}
```

2. Write the following code in the *createEntry()* method:

```
public void createEntry(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent
wdEvent )
  {
  //@@begin createEntry(ServerEvent)
  destroyWindow();
  createEntry(wdContext.currentEditEntryElement().getStart(),
    wdContext.currentEditEntryElement().getEnd(),
    wdContext.currentEditEntryElement().getSubject(),
    wdContext.currentEditEntryElement().getAdditionalText(),
    wdContext.currentEditEntryElement().getSemantics(),
    wdContext.nodeTeam().getLeadSelection());
  //@@end
}
```

3. Write the following code in the *editEntry()* method:

```
public void editEntry(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent
wdEvent){
  //@@begin editEntry(ServerEvent)
destroyWindow();
  int index = wdContext.currentContextElement().getIndexEntry();
  IEntriesElement entry =
   (IEntriesElement)wdContext.nodeTeam().nodeEntries().getElementAt(index);
  IEditEntryElement editEntry =
   (IEditEntryElement)wdContext.currentEditEntryElement();
  entry.setAdditionalText(editEntry.getAdditionalText());
  entry.setSubject(editEntry.getSubject());
  entry.setSemantics(editEntry.getSemantics());
  entry.setStart(editEntry.getStart());
  entry.setEnd(editEntry.getEnd());
  //@@end
}
```

Write the following code in the *cancel()* method:

```
destroyWindow();
```

Firing the Events in Component Controller:

1. Switch to the Java Editor of the *CalendarComp*
1. Write into the *editEntry* method:

   ```
   wdThis.wdFireEventEditEntry();
   ```

2. Write into the *deleteEntry* method:

   ```
   wdThis.wdFireEventDeleteEntry();
   ```

3. Write into the *cancel* method:

   ```
   wdThis.wdFireEventCancel();
   ```

4. Write into the *createEntry* method:

   ```
   wdThis.wdFireEventCreateEntry();
   ```

Calling the Component Controller methods from the *CreateEntryView*:

5. Switch to the Java Editor of the *CreateEntryView*
2. Write into the *onActionSave* method:

   ```
   wdThis.wdGetCalendarCompController().createEntry();
   ```

6. Write into the *onActionSelectMember* method:

```
wdContext.nodeTeam().setLeadSelection(index);
```

7. Write into the *onActionCancel* method:

```
wdThis.wdGetCalendarCompController().cancel();
```

Calling the Component Controller methods from the *EditEntryView*:

The last steps have to be performed analogical to the *EditEntryView*:

3. Write into the *onActionSave* method:

```
wdThis.wdGetCalendarCompController().editEntry();
```

8. Write into the *onActionDelete* method:

```
wdThis.wdGetCalendarCompController().deleteEntry();
```

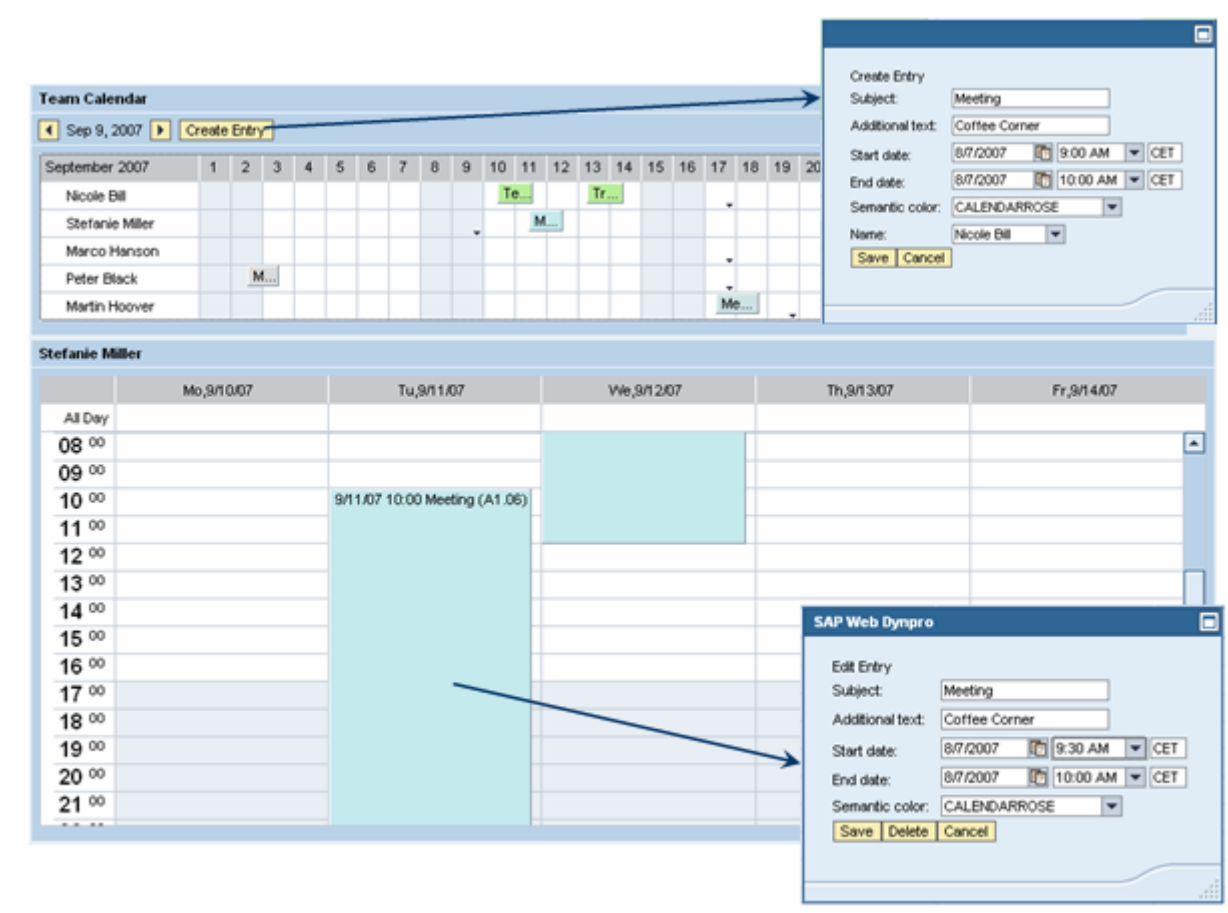
9. Write into the *onActionCancel* method:

```
wdThis.wdGetCalendarCompController().cancel();
```

Building, Deploying, and Running the Calendar Tutorial

You have now successfully completed the tutorial *Implementing a Calendar in Web Dynpro Java* - **Congratulations**!

1. Open the context menu for the *CalendarApp* and choose *Deploy New Archive and Run*.



The team calendar displays the various team members. By clicking on the day in the upper calendar, you can display the selected team member and the corresponding time in more detail in the corresponding row in the lower weekly calendar.

Depending on your selection, this therefore allows you to switch the member, the date, or both.

Below the title of the team calendar, you find the toolbar with the *CalendarPaginator* that allows navigation in the team calendar. Using this element, you can jump either a month forward or a month back in the team calendar. You can also use it to create a new entry by selecting an entry in the lower week calendar. You can enter these modifications in popups.

If you select and click an entry in the weekly calendar, you can change or delete it in a popup.

Related Links

Parameter Mapping

Core Data Types

Calendar

Copyright