This article shows what you can do when a particular dijit from the Dojo toolkit does not entirely address your requirements and you need to create your own custom widget. By the end, using an example with a set of requirements and an approach for how to fulfill them, you will be familiar with using a dijit and other Dojo core functionalities and how to declare your own widget.

Introduction

The Dojo Toolkit is a powerful JavaScript™ library that enables web developers to create Rich Internet Applications using object-oriented widgets with minimal development time and effort. It comes with four packages, known as Dojo (the core), Dijit (the UI framework), dojox (the dojo extension) and util. You can use the functionality provided by the toolkit as is, or you can extend them and create your own widgets. The provided functionalities include DOM manipulation, development with AJAX, events, data stores, and more.

The Dijit (dojo widget) package, dojo's own UI library, contains a collection of dojo classes that enable developers to create rich and powerful cross-platform web 2.0 interfaces with minimal effort. These Dijit widgets, or dijits, are supported by themes that are easy to manipulate. Examples of the dijits in this package are buttons, text fields, editors, progress bars, and many more.

Using these dijits, for example, you can create a submit form that includes text fields for name, email address, and phone numbers, plus date fields, checkboxes, buttons, and validation, all in a matter of minutes with minimal JavaScript knowledge.

One of the richest dijits provided is the Calendar dijit, which enables you to display a calendar in the context of a month. Users can easily navigate month by month, year by year, or jump to any month in the same year to select specific dates.

When working on the development of a Rich Internet Application (RIA), you can often use the dijits as is. However, sometimes you might require a different style (like changing color or theme), or more involved changes that could require a combination of functionality, template, and style changes. You can fulfill these requirements by either creating a new custom widget from scratch or creating a custom widget that extends an existing dijit.

This article presents an exercise in which you have a requirement to use a different variation of the Calendar widget on your website. To fulfill this requirement, you will create a new class that
will satisfy the requirements. This exercise uses Dojo version 1.7, and offers an opportunity to explore the Calendar dijit and ways to reuse an existing dijit with minimal modifications to save on development time. You'll also see a working example of a new class declared in Dojo 1.7, and explore some of the Dojo base functions, such as date manipulation, hitching, publish and subscribe, and more.

**The problem**

In this exercise, you will work on a customized version of the Calendar dijit with these requirements:

- Calendar should only display the days of the current month (hide and disable days that don’t belong to current month).
- Calendar should only display the current year (no previous or following years) at the bottom of the calendar.
- Calendar should display the current month name at the top of the calendar widget.
- Users cannot jump to any other month (disable the month drop down button at the top).
- Extract the arrows displayed at the top of the calendar for moving from month to month (backward or forward) and display arrows instead next to the calendar as dijit buttons. These two new buttons are the only way for user to change the month.
- There are minimum and maximum boundary dates, meaning that all dates outside this boundary should be disabled and inaccessible.
- Disable the appropriate month navigation button when a boundary date is reached.
- Add special styling to selected days in the calendar.
- When user selects a date, pass the date to a function that will process the new selected value.

The solution is to create a custom widget developed by editing the Calendar dijit using JavaScript and CSS. Figure 1 shows the Calendar widget before (left) and after (right) the requirements above are applied.

**Figure 1. Comparison of a standard Calendar dijit to custom Calendar widget**

To do this, you will need to create three files:

- **Dijit template**: A markup that will display the components of the custom widget.
Extending Dojo diijits to create custom widgets

- **Dijit class**: A widget class created using declare (JavaScript).
- **CSS file**: Containing with all necessary stylesheet classes.

Figure 2 shows the file structure and location of your custom widget. Your starting point is index.html, which will act as the controller for the widget in this example. The simple.css file will contain all styles.

**Figure 2. File structure**

![File structure diagram]

**Creating the widget**

**The dijit template**

Create three JavaScript div elements: one for the calendar, and two for the arrow buttons for navigating month by month backward and forward (Listing 1). The divs will use attach points (data-dojo-attach-point) for reference. Using attach points is a better choice than using ids because it will enable you to have multiple instances of the same widget on same page without having to worry about id conflicts.

**Listing 1. The dijit template**

```html
<div class="CalendarArrow">
  <div data-dojo-attach-point="calendarPreviousMonthButtonAP"></div>
</div>

<div class="CalendarDijit">
  <span data-dojo-attach-point="calendarMonthOneAttachPoint"></span>
</div>

<div class="CalendarArrow">
  <div data-dojo-attach-point="calendarFollowingMonthButtonAP"></div>
</div>
```

**The dijit class**

As per the application requirements, you need to define these variables:

- **selectedDate**: Initial value for the calendar.
- **currentFocusDate**: The value that the calendar references to know which month to display; initially set equivalent to selectedDate.
- **calendarInstance**: Dijit calendar instance.
- **bookingWindowMaxDate**: Last allowed day in the calendar.
• **bookingWindowMinDate**: First allowed day in the calendar.
• **onValueSelectedPublishIDString**: String that represents the pub/sub channel (or topic).

### The JavaScript functions

Begin by modifying these style sheet elements:

**constructor**

Override the constructor to copy the variables passed from the controller. Use dojo/_base/ lang/mixin, which will match the variable names and copy the values into your custom widget variables (Listing 2).

**Listing 2. The constructor**

```javascript
constructor: function (args){
  if(args){
    lang.mixin(this,args);
  }
}
```

**postCreate**

All dates will be passed as a string in the en-us short format of mm/dd/yyyy. Convert all date strings to date objects using dojo/date/locale function for selectedDate, bookingWindowMaxDate, bookingWindowMinDate (Listing 3).

**Listing 3. Using dojo/date/locale**

```javascript
this.bookingWindowMinDate = locale.parse(this.bookingWindowMinDate, {formatLength: 'short', selector:'date', locale:'en-us'});
```

Create an instance of the calendar object (Listing 4). The logic for the creation is in the createCalendar function. You create an instance of a dijit calendar programmatically and attach it to a div that you will create using dojo/dom-construct (equivalent to dojo.create in an older version of dojo). This is good practice in general because it lets you destroy the calendar without losing the attach point.

**Listing 4. Returning an instance of dijit/Calendar**

```javascript
return new Calendar({
  value : selectedDate,
  currentFocus : selectedDate }, domConstruct.create("div", {},
  calendarAttachPoint));
```

Notice that you are setting the currentFocus value in the calendar dijit. The Calendar dijit will always use your local current date to display its first landing screen, so if you want the calendar to display a different screen (date), you have to set currentFocus. Therefore, for your custom widget you need to set the initial value for the calendar and the currentFocus as selectedDate (per requirements). For this example, that's a day in August 2012.

To satisfy the other requirements, you need to override these three functions from the Calendar dijit:

• **isDisabledDate**
When the calendar dijit is loading a view, it iterates through the days of the current view one by one (all 42 days) and will call the isDisabledDate and getClassForDate (covered next) functions for every day.

The isDisabledDate function is used to disable certain dates in the calendar (Listing 5). If the function returns true, the day will be disabled. Every time the calendar refreshes, this function is called and every day in the calendar is passed to it. For your custom widget, you need to:

- **Disable any day that doesn't belong to current month:** To do this, you will use the dojo/date/difference function, which compares two date objects based on an interval and return 0 if equal. You will compare the currentFocusDate variable with every day in the current view using the month interval, and return true if they are not equal to disable the day.

- **Disable days outside of the boundary dates:** Use dojo/date/difference again, but with the interval set to “day." If the return value is smaller than bookingWindowMinDate or larger than bookingWindowMaxDate, then return true to disable the date.

**Listing 5. Overriding isDisabledDate**

```javascript
isDisabledDate: function(date) {
    //disable any day that doesn't belong to current month
    if(dojoDate.difference(parent.currentFocusDate, date, "month")!==0){
        return true;
    }
    if(dojoDate.difference(parent.bookingWindowMinDate, date, "day")||
        dojoDate.difference(parent.bookingWindowMaxDate, date, "day")<0){
        return true;
    }
    else {
        return false;
    }
}
```

- **getClassForDate**

Although you disabled the days that don’t belong to the current month view with isDisabledDate, you need to hide them as well. The getClassForDate function is used to return a CSS class name to mark the day differently on the calendar. For your custom widget, you need to indicate the selectedDate by adding a blue box with a black border to that date (Listing 6). You also need to indicate the dates outside of your minimum and maximum boundaries with a gray color and hide the days that don’t belong to the current month.

To identify the date that needs to be styled differently, you can use dojo/date/compare, which will take two date values (date objects) and portion (string) and return 0 if equal. Here, you will pass the currentFocusDate, the day in iteration, and “date” for portion, because you are only interested in comparing the date without the timestamp. If this comparison returns 0, this function will return the class “Available,” which is defined in your CSS file (Listing 7). You will be using CSS .class selectors to target the specific elements we want to change.

**Listing 6. Overriding getClassForDate**

```javascript
getClassForDate: function(date) {
    if ( dojoDate.compare(date,selectedDate,"date") === 0) {
        return "Available";
    } // apply special style
}
```
Listing 7. CSS class to mark available days

```css
AvailabilityCalendars .Calendars .CalendarDijit .Available .dijitCalendarDateLabel {
  background-color: #bccedc !important;
  border: 1px solid #000000 !important;
}
```

You will use the same if conditions from `isDisabledDate` to identify the days out of boundaries and the days that don't belong to the current month, but return the CSS class name (Listings 8 and 9).

Listing 8. Hiding and disabling days

```javascript
if(dojoDate.difference(parent.currentFocusDate, date, "month")!==0){
  return "HiddenDay";
}
if(dojoDate.difference(parent.bookingWindowMinDate, date, "day")<0 ||
  dojoDate.difference(parent.bookingWindowMaxDate, date, "day")>0){
  return "Disabled";
}
```

Listing 9. CSS class to mark hidden and disabled days

```css
AvailabilityCalendars .Calendars .CalendarDijit .HiddenDay .dijitCalendarDateLabel {
  background-color: #ffffff !important;
  border-color: #ffffff;
  color: #ffffff;
}
AvailabilityCalendars .Calendars .CalendarDijit .Disabled .dijitCalendarDateLabel {
  background-color: #9c9c9c;
}
```

- **onChange**
  This function is invoked only when you set a new value to the calendar or when you select an enabled day on the calendar (Listing 10). This function returns a date object of the selected day. You will use that to publish the date to another method that will process the date. Call a function defined in your custom widget (onValueSelected) where you can do any needed processing before publishing to the controller (Listing 11). In this example, you will just publish the date to the controller using `dojo/_base/connect/publish`. The channel (or topic) string is stored in the variable `onValueSelectedPublishIDString`.

Listing 10. Using Calendar's onChange and hitch

```javascript
onChange : lang.hitch(this, function(date){
  this.onValueSelected(date);
})
```
Listing 11. Using publish

```javascript
onValueSelected : function (date){
    connect.publish(this.onValueSelectedPublishIDString, [date]);
}
```

Notice that you used dojo/_base/lang/hitch to give scope to call the function onValueSelected (listing 10). Your controller (in this scenario, index.html) will have a subscriber to that channel to process the date (Listing 12). In this example, you only log it. You can replace this with any other required logic.

Listing 12. Subscriber to our publish

```javascript
connect.subscribe("selectedValueID", function(date){
    //Do some processing
    console.log("New Selected Date: ", date);
});
```

The Calendar dijit ships with a monthDropDownButton at the header. This button displays a list of all months and lets the user jump to any month. To fulfill the requirements, you need to disable this button by setting monthWidget to “disabled” (Listing 13).

Listing 13. Disable drop down button in the Calendar header

```javascript
this.calendarInstance.monthWidget.set("disabled", true);
```

From a usability perspective, you also need to hide the arrow so that the user isn’t compelled to click on it. To do that, add CSS classes that will target the elements you want to manipulate (Listing 14).

Listing 14. CSS class to hide the drop down button arrow

```css
.AvailabilityCalendars .Calendars .CalendarDijit .dijitDropDownButton .dijitArrowButtonInner
{
    visibility: hidden;
}
```

Next, use CSS classes to hide the previous and following years’ digits from being displayed at the bottom (Listing 15).

Listing 15. CSS class to hide the years digits at the bottom of the Calendar

```css
.AvailabilityCalendars .Calendars .CalendarDijit .dijitCalendarPreviousYear,
.dijitCalendarNextYear
{
    padding: 1px 6px;
    visibility: hidden;
}
```

You will also hide the arrows at the top, which would otherwise enable a user to move from month to month (Listing 16).
Next, create the two new buttons to let the user navigate through the months. Use dijit/form/Button and create them programmatically. For the first button (backward), set the label as “<<” and override the onClick function (Listing 17). The logic for the onClick will be in the goToPreviousMonth function.

You want the calendar to move backward one month every time a user clicks on the button. In goToPreviousMonth, you need to first change the currentFocusDate to currentFocusDate - 1 month and then refresh the view of the calendar. Finally, you must check if this is the last month to display and if so, disable the button.

Use dojo/date/add function, which takes a date object, interval (String) and amount (integer). For your situation, the date will be the currentFocusDate object, interval is “month,” and amount is -1 (Listing 18).

Set the new view for the calendar by setting the currentFocus with the new date value. (This will automatically refresh the calendar and display the new view).

Finally, you check if this will be the last month view by comparing the currentFocusDate with the minimum boundary; if it is, then disable the backward button. Also, check if you should enable the forward button (in case it was disabled and now you are moving away from the maximum boundary) (Listing 19).

The second button works the same way. The label will be “>>>” and onClick calls goToNextMonth, which uses same function except that you add one month (Listing 20).
Listing 20. Function that controls the button to move to the following month

```javascript
function goToNextMonth (calendarInstance) {
  this.currentFocusDate = dojoDate.add(this.currentFocusDate, "month", 1);
  calendarInstance.set("currentFocus", this.currentFocusDate);
  if (this.isLastCalendarMonth(this.bookingWindowMaxDate, this.currentFocusDate)) {
    this.calendarFollowingMonthButton.set("disabled", true);
  } else if (!this.isLastCalendarMonth(this.bookingWindowMinDate, this.currentFocusDate)) {
    this.calendarPreviousMonthButton.set("disabled", false);
  }
}
```

Finally, Listing 21 shows an example of what would be in your controller class, which will make the call to instantiate your new customCalendar.

Listing 21. Snapshot of how to create an instance of the new custom calendar widget

```javascript
require(["myUtil/customCalendar", "dojo/_base/connect"], function(myCalendar, connect) {
  var params = {
    "bookingWindowMinDate": "10/9/2011",
    "bookingWindowMaxDate": "10/9/2012",
    "selectedDate": "8/15/2012",
    "onValueSelectedPublishIDString": "selectedValueID"
  }; var myTest = myCalendar(params);
  myTest.placeAt("nodeId", "last");

  connect.subscribe("selectedValueID", function(date) {
    // Do some processing
    console.log("I got: ", date);
  });
});
```

As you can see, you are creating an object params with the values needed to pass to your custom calendar widget and subscribing to the channel.

**More functions**

There are a few other functions and properties that you might find useful in this scenario:

- When you pass a date object and locale to dojo/date/locale/isWeekend, it returns true if the day is a weekend day (Saturday and Sunday for en-us locale). This can be used to disable or style weekend days differently, if needed.

- The Calendar dijit also contains a dayWidth property that takes a string as a value. By default it is set to “narrow,” which shortens the displayed calendar day, for example, to “M” rather than Monday. Other values are “wide” for full day name display, and “abbr” for an abbreviation (such as “Mon”).

A variation of these custom widget requirements might have asked for the widget to display multiple calendars, and then require both calendars to advance when the user clicks to view the next month (Figure 3). This can be easily achieved by changing the widget variables to support an array rather than a single value variable.
Conclusion

Through a combination of JavaScript and CSS modifications, you can easily create a custom widget to better satisfy your project requirements. This article demonstrated this practice using Dojo 1.7 to declare a class that extended the Calendar dijit, and explored some of the dojo functionalities like date manipulation, hitching, publish and subscribe, and other base dojo functions. Hopefully, you will be able to apply these steps to extend a Dojo dijit and create new widgets of your own.
# Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample application</td>
<td>customCalendar17.zip</td>
<td>4 KB</td>
</tr>
</tbody>
</table>
Related topics

- Dojo Toolkit 1.8
- Knowledge path: Get Started with Dojo Development
- Dijit Overview
- Dijit Theme Tester
- CSS class selector
- Dojo Toolkit
- Dojo 1.7 Documentation

© Copyright IBM Corporation 2012
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)