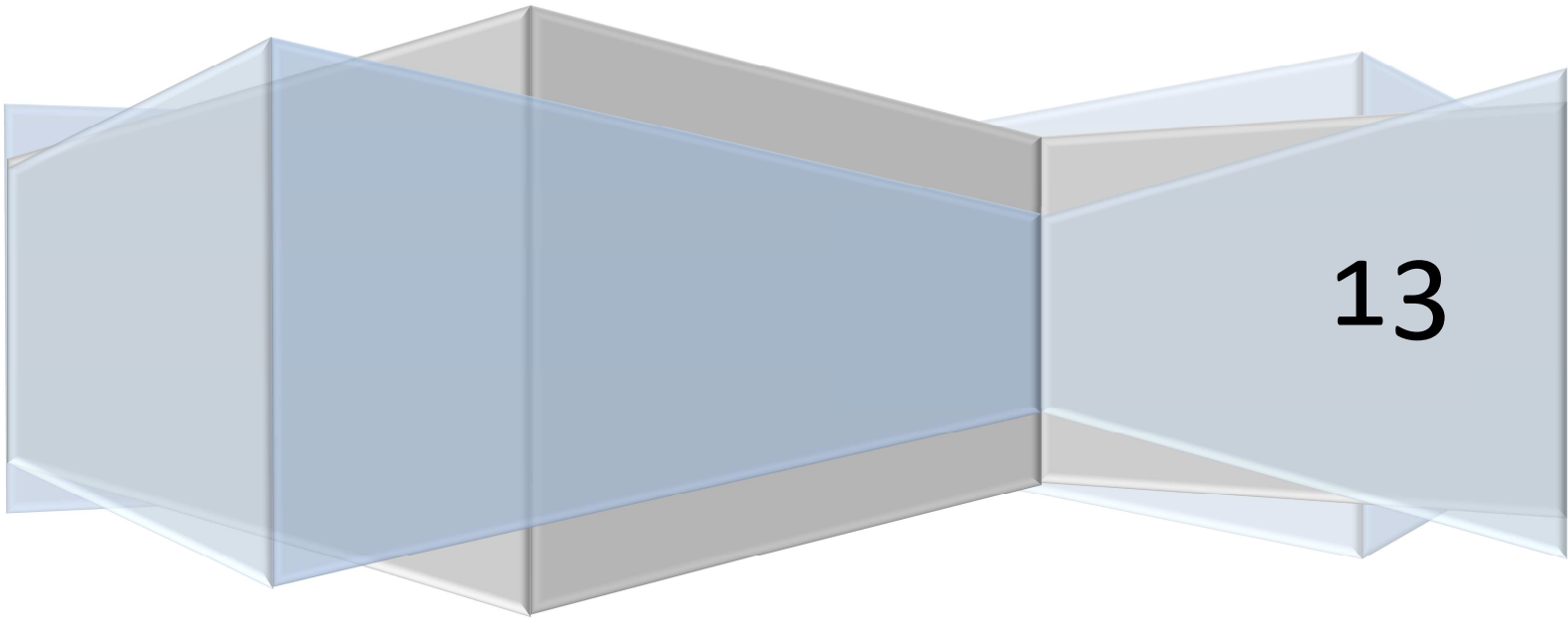
Quintessens

Using Twitter Bootstrap in XPages

The ultimate incomplete guide

Patrick Kwinten



Contents

Version

Date	Version	Author	Remarks
2013-10-15	0.1	Patrick Kwinten	Initial Draft

Terminology

The following terms are being used throughout the document. Sometimes abbreviations are used.

Term	Abbreviation	Remarks
Twitter Bootstrap	TB	Front-end framework for faster and easier web-development with mobile first in mind-set.
XPages		Rapid web and mobile application development technology.
JavaServer Faces	JSF	Java specification for building component-based user interfaces for web applications.
Dojo Toolkit	Dojo	Open source modular JavaScript library to ease the rapid development of cross-platform, JavaScript/Ajax-based applications and web sites.
IBM Notes	Notes	IBM server product that provides enterprise-grade e-mail, collaboration capabilities, and a custom application platform.
jQuery		Multi-browser (cf. cross-browser) JavaScript library designed to simplify the client-side scripting of HTML.
Responsive Web Design	RWD	Web design providing an optimal viewing experience across a wide range of devices.

Introduction

This document describes how to integrate the popular front-end framework Twitter Bootstrap with XPages. The purpose is to provide developers an introduction to use Twitter Bootstrap in XPages development projects by building an actual application.

About Twitter Bootstrap

Twitter Bootstrap describes itself as a sleek, intuitive, and powerful front-end framework for faster and easier web development.

Made for everyone

TB was made to not only look and behave great in the latest desktop browsers (as well as IE7!), but in tablet and smartphone browsers via responsive CSS as well.

Packed with features

TB is rich in features: A 12-column responsive grid, dozens of components, JavaScript plugins, typography, form controls, and even a web-based Customizer to make TB your own.

Scaffolding

- Basic typography and styles.
- Includes normalize.css to makes browsers render all elements more consistently and in line with modern standards.
- Grid system (12, 16 or 24 column grid, 980 wide container, fluid (percent instead of pixels).

About XPages

XPages is a rapid web and mobile application development technology. It allows data from IBM Notes and Relational Databases to be displayed to browser clients on all platforms.

The programming model is based on web development languages and standards including JavaScript, Ajax, Java, the Dojo Toolkit, Server-side JavaScript and JavaServer Faces.

Purpose of this document

This document has as purpose to describe how to use TB in XPages. Probably the best way is to provide some sort of hands-on examples that build an actual Notes application with a web-interface using TB components.

The provided examples are not unique. Most of them are likely taken from online resources. But bringing them together in a single document makes it hopefully more valuable.

Application architecture

The application we will build is by no way unique or complicated. Its main purpose is to demonstrate the features in TB and to create and display documents.

Phases

The development process can be ordered into several phases:

- Preparation - Building an application template.
- Application development – Implementing specific TB features for a real world application.
- …

Preparation phase

In the preparation phase we will build a template that you can re-use as a starter-kit for other application development projects.

Below is an ordered list of the activities we will perform during this phase:

1. Add TB resources to an empty Notes application.
2. Create a Bootstrap Theme.
3. Create a responsive layout.

Adding Twitter Bootstrap resources to a Notes application

TB consists of a number source files:

- CSS / JavaScript and images.

You can download the resources here: http://getbootstrap.com/. The easiest way to add all of those to your application is by adding them to the WebContent folder. This folder is treated as the "root" of the NSF when an XPage is accessed. The files and folders you import can be referenced using a URL that's relative to the root of the application/ database.

1. Open the Package Explorer and navigate to your application.
2. Right-click on the WebContent folder and choose 'Import > General > File System'.
3. Select the location where you've extracted the TB files to import them.

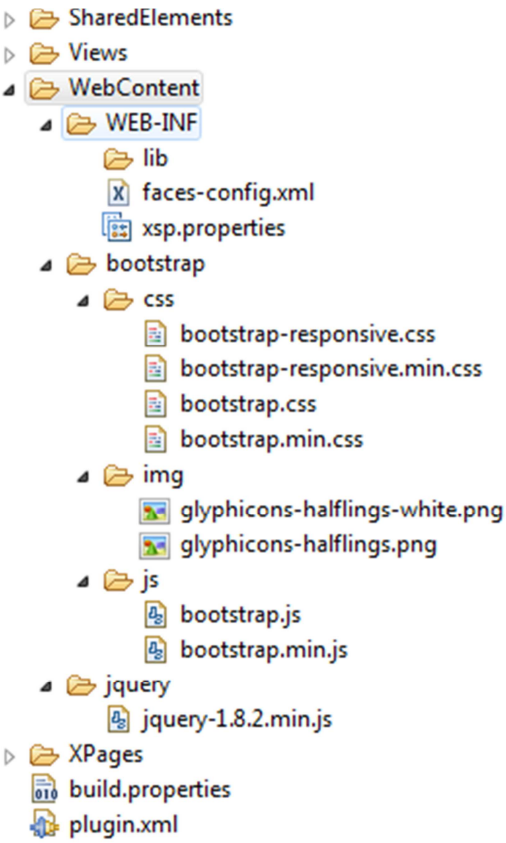
It is recommended to follow the file structure that TB already provides. Create a folder called bootstrap in the WebContent folder first.

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.min.css
├── js/
│   ├── bootstrap.js
│   ├── bootstrap.min.js
└── img/
    ├── glyphicons-halflings.png
    └── glyphicons-halflings-white.png
```

If you're using Windows you can also drag-n-drop the files right into the WebContent folder.

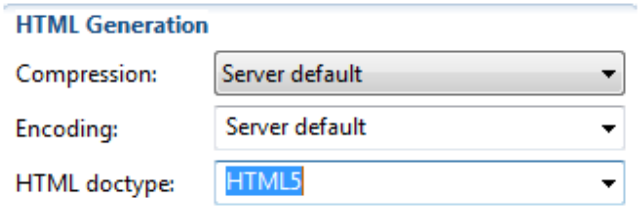
A number of components in TB rely on jQuery, therefore you must also add jQuery to your application using the same method. You can download jQuery here: http://jquery.com/download/.

The WebContent folder will eventually look similar to this:



TB requires that the HTML Doctype for your application is set to HTML5:

- Open the Application Properties.
- Open the Tab 'XPages' and navigate to section HTML Generation.
- Select for HTML Doctype: HTML5.



Finally, add the Bootstrap CSS and JavaScript to all pages, either by adding them as 'resources' to an XPage(s)/ Custom control or by creating a "theme" in your application.

Creating a Bootstrap theme

A Theme is a design element in IBM Notes, it's is a defined XML object and attribute overwrite rendering template. Themes can be used to conditionally include Lotus Domino design resources like stylesheets and script libraries or to define or extend attributes for XPage controls.

They can additionally be used to extend existing Themes, which allows application developers to apply rulesets defining when, for example, a given XPage computed field control will render in the Web browser client with a styleClass.

- Create a new Theme design element under 'Resources \ Themes'. Give it the name 'bootstrap'.

A basic bootstrap theme could look like this:

```
<theme xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="platform:/plug
in/com.ibm.designer.domino.stylekits/schema/stylekit.xsd" >

<!-- meta tags -->

<resources>

        <metaData>

            <name>viewport</name>

            <content>width=device-width, initial-scale=1.0</content>

        </metaData>

</resources>


          <!-- jquery -->

          <resource>

                    <content-type>application/x-javascript</content-type>

                    <href>jquery/jquery-1.8.2.min.js</href>

          </resource>


          <!-- bootstrap js -->

          <resource>

                    <content-type>application/x-javascript</content-type>

                    <href>bootstrap/js/bootstrap.min.js</href>

          </resource>


          <!-- bootstrap css-->

          <resource>

                    <content-type>text/css</content-type>

                    <href>bootstrap/css/bootstrap.min.css</href>

          </resource>

          <resource>

                    <content-type>text/css</content-type>

                    <href>bootstrap/css/bootstrap-responsive.min.css</href>

          </resource>

</theme>
```

Theme dissection

Mobile first

To ensure proper rendering and touch zooming the viewport meta tag will be added to the <head> section of each page:

```
<metaData>
    <name>viewport</name>
    <content>width=device-width, initial-scale=1.0</content>
</metaData>
```

jQuery

A lot of features of components in TB depend on jQuery:

```
<!-- jquery -->
<resource>
        <content-type>application/x-javascript</content-type>
        <href>jquery/jquery-1.8.2.min.js</href>
</resource>
```

Bootstrap core function

Bootstrap's core functionality is captured in the following JavaScript file, we will load the minified version):

```
<!-- bootstrap js -->
<resource>
        <content-type>application/x-javascript</content-type>
        <href>bootstrap/js/bootstrap.min.js</href>
</resource>
```

Bootstrap styling

Bootstrap's global CSS settings, fundamental HTML elements styling and it's advanced grid system is loaded in the minified version:

```
<!-- bootstrap css-->
<resource>
        <content-type>text/css</content-type>
        <href>bootstrap/css/bootstrap.min.css</href>
</resource>
```

Responsive design

Responsive behavior is supported by the following style sheet:

```
<resource>
        <content-type>text/css</content-type>
        <href>bootstrap/css/bootstrap-responsive.min.css</href>
```

```
        </resource>
```

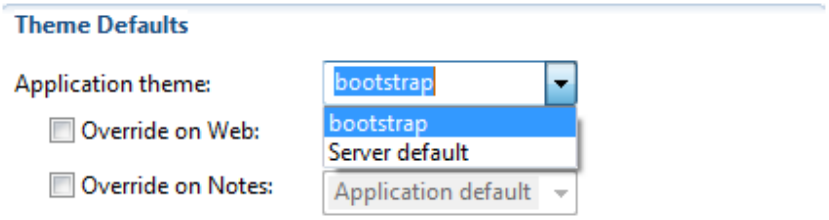
Responsive is in TB turned off by default. It's turned on by adding the viewport meta tag:

```
<metaData>
            <name>viewport</name>
            <content>width=device-width, initial-scale=1.0</content>
</metaData>
```

In case you want you can disable the responsiveness: http://getbootstrap.com/getting-started/#disable-responsive.

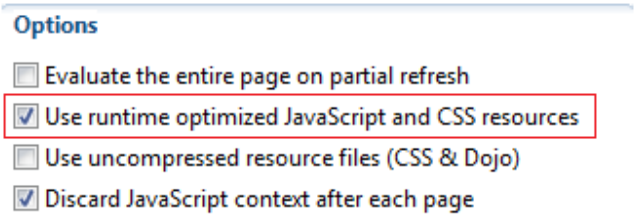
Activate theme

- Open the Application Properties .
- Open the Tab 'XPages' and navigate to section Themes Defaults.
- Select for Application theme: bootstrap.



Resource aggregation

The resource aggregation function automatically combines the CSS and JavaScript files loaded on a webpage into a single file, thus reducing the number of requests on a page. Reducing the number of requests is especially relevant when using a mobile internet connection.



If you use TB and enable the option, you'll find that it will break the links to the default Bootstrap images. The reason for this is the (virtual) location of the combined CSS file changes.

Workaround

A solution is to create a simple CSS file that overrides the image location and add that to your page/ app. The bootstrap CSS files only refer to images in 2 CSS selectors, so those are the only you need to override. In Bootstrap 2.3.1 the override CSS file looks like this:

```
/*keep showing images if css aggregation is used*/

[class^="icon-"],

[class*=" icon-"] {

  background-image: url("../../../bootstrap/img/glyphicons-halflings.png");

}

.icon-white,

.nav-pills > .active > a > [class^="icon-"],

.nav-pills > .active > a > [class*=" icon-"],

.nav-list > .active > a > [class^="icon-"],

.nav-list > .active > a > [class*=" icon-"],

.navbar-inverse .nav > .active > a > [class^="icon-"],

.navbar-inverse .nav > .active > a > [class*=" icon-"],

.dropdown-menu > li > a:hover > [class^="icon-"],

.dropdown-menu > li > a:focus > [class^="icon-"],

.dropdown-menu > li > a:hover > [class*=" icon-"],

.dropdown-menu > li > a:focus > [class*=" icon-"],

.dropdown-menu > .active > a > [class^="icon-"],

.dropdown-menu > .active > a > [class*=" icon-"],

.dropdown-submenu:hover > a > [class^="icon-"],

.dropdown-submenu:focus > a > [class^="icon-"],

.dropdown-submenu:hover > a > [class*=" icon-"],

.dropdown-submenu:focus > a > [class*=" icon-"] {

  background-image: url("../../../bootstrap/img/glyphicons-halflings-white.png");

}
```

Now here's a trick: if you only load that CSS file if resource aggregation is **enabled**, all images will still work if you disable the aggregation function:

```
<xp:this.resources>

  <xp:styleSheet href="/override.css">
```

```
   <xp:this.rendered><![CDATA[#{javascript:context.getProperty("xsp.resources.aggregate")=="true"}]]></xp:
this.rendered>

  </xp:styleSheet>

 </xp:this.resources>
```

The code above is for when you want to load the CSS file from an XPage. If you're using themes in your application (and you should), you can also conditionally load the CSS file in the theme:

```
<resource rendered="#{javascript:context.getProperty('xsp.resources.aggregate').equals('true')}">

          <content-type>text/css</content-type>

          <href>override.css</href>

</resource>
```

Creating a responsive layout

Responsive web design (RWD) is a web design approach aimed at crafting sites to provide an optimal viewing experience—easy reading and navigation with a minimum of resizing, panning, and scrolling—across a wide range of devices (from mobile phones to desktop computer monitors).

In our bootstrap theme we have already defined that we want a responsive layout:

```
<resource>
                    <content-type>text/css</content-type>
                    <href>bootstrap/css/bootstrap-responsive.min.css</href>
</resource>
```

Utility classes

You can use the following utility classes for showing and hiding content by device. Below is a table of the available classes and their effect on a given media query layout (labeled by device).

Class	Phones	Tablets	Desktops
.visible-phone	Visible	Hidden	Hidden
.visible-tablet	Hidden	Visible	Hidden
.visible-desktop	Hidden	Hidden	Visible
.hidden-phone	Hidden	Visible	Visible
.hidden-tablet	Visible	Hidden	Visible
.hidden-desktop	Visible	Visible	Hidden

Grid

TB comes with a grid system (link). This means it is organized in columns which can either contain content of any kind or nested columns.

You can decide to create a fixed column layout or a fluid layout (link). To create a responsive design you need to go for the fluid layout (link).

Example fluid:

```
<div class=”container-fluid”></div>
```

To make the grid fixed just remove the fluid class suffix:

```
<div class=”container”>
```

Reusable Custom Control for layout

From other XPage tutorials you have learned about the power of re-using custom controls, especially one for the layout of an application. In our application we’re going to do exactly the same. The custom control will have 1 Editable Area control for custom content per individual XPage.

- Create a custom control, name it e.g. ccLayout.
- Create another custom control, name it e.g. ccLayoutNavBar.

The code of the ccLayoutNavBar custom control may look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core">
    <!-- top nav menu -->
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="navbar-inner">
            <div class="container">
                <button type="button" class="btn btn-navbar" data-toggle="collapse" data-target=".nav-coll
apse">
                    <span class="icon-bar" />
                    <span class="icon-bar" />
                    <span class="icon-bar" />
                </button>
                <xp:link styleClass="brand" escape="true"
                     text="#{javascript:database.getFileName()}"
                     id="link1">
                     <xp:this.value>
                               <![CDATA[#{javascript:"/" + database.getFilePath().replace("\\", "/")}]]>
                     </xp:this.value>
                </xp:link>
                <div class="nav-collapse collapse">
                    <ul class="nav">
                        <li>
```

```
                            <xp:link escape="true" text="XPages Info" id="link4" value="http://xpages.info
">
                            </xp:link>
                        </li>
                        <li class="dropdown">
                           <a class="dropdown-toggle" data-toggle="dropdown" href="#">Links<b class="caret
" /></a>
                        <ul class="dropdown-menu">
                                <li>
                                    <xp:link escape="true"
                                          text="Official Bootstrap site"
                                          id="link2"
                                          value="http://twitter.github.io/bootstrap/"
                                          target="_blank">
                                    </xp:link>
                                </li>
                                <li>
                                <xp:link escape="true" text="Bootswatch"
                                          id="link3"
                                          value="http://bootswatch.com/"
                                          target="_blank" title="Gallery with Bootstrap themes">
                                </xp:link>
                                </li>
                            </ul>
                        </li>
                    </ul>
                </div>
                <!--/.nav-collapse -->
            </div>
        </div>
    </div>
</xp:view>
```

Don’t worry that the navigation is not application specific yet. We will fix that later.

The code for the ccLayout custom control looks much simpler. It contains the ccLayoutNavBar custom control and an Editable Area control:

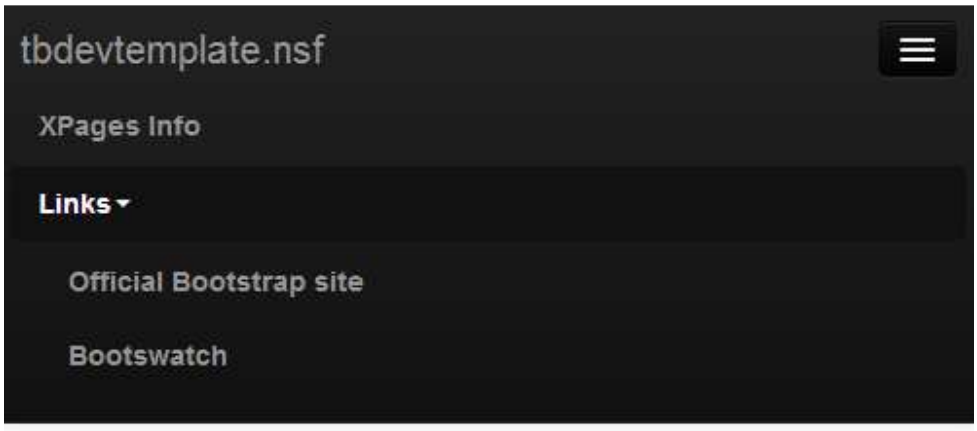
```
<?xml version="1.0" encoding="UTF-8" ?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core"
          xmlns:xe="http://www.ibm.com/xsp/coreex"
```

```
          xmlns:xc="http://www.ibm.com/xsp/custom">
          <xc:ccLayoutNavBar></xc:ccLayoutNavBar>
          <!--  main content area -->
          <div class="container">
                    <xp:callback id="callback1" />
          </div>
          <!--/ main content area -->
</xp:view>
```

You can now create an XPage and drag custom control ccLayout in and preview it in a web browser. You won't see much but small navigation bar on the top:

If you alter the browser width you see the responsive behavior. Some links will be grouped under a menu marker:

Click on it to further collapse the menu:



End of preparation phase

We have finalized the preparation phase. We now have a re-usable development template for Twitter Bootstrap projects hosted on IBM Notes using XPages.

We will refine the template later, if possible, to optimize performance and behavior.

In the next chapter we will create real Notes applications with (of course) Forms and Views.

Application development phase

The application we are going to build has as main function to demonstrate TB's capabilities and how to integrate it in XPages. The examples shown in the next chapters have therefore most demonstrative value and less application value.

Adding a simple form

The traditional Notes form acts as a data schema in XPages development, in other words via different XPages you can update (partly) a Notes document, using the Notes form as a data schema.

In traditional Notes development you mostly had just 1 form for 1 sort of document.

To explore TB' features most we will create as many different type of fields as possible.

Form structure

A form consists mostly of the following structure:

- Header or Legend.
- Field entries, mostly a combination of a label and input control.
- Actions, e.g. Save or Cancel.

Bootstrap Base provides functionality to setup forms. You can find details here: http://getbootstrap.com/css/#forms.

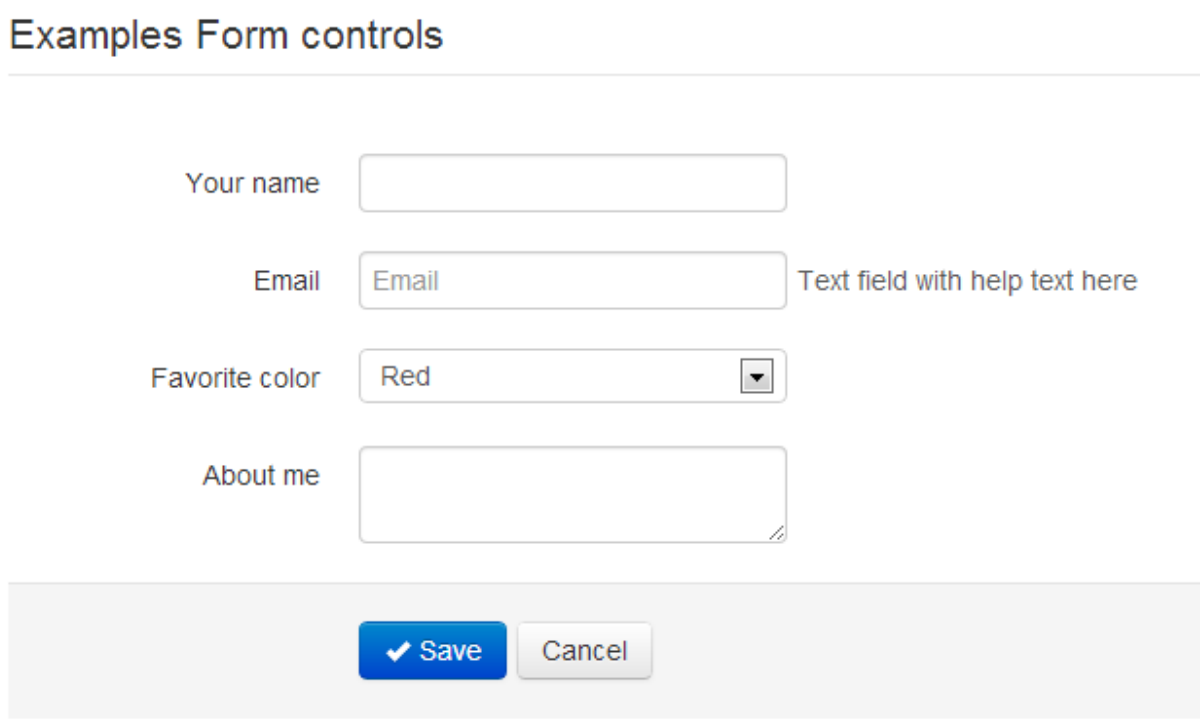
Below is the starting code for our form XPage:

```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core" xmlns:xc="http://www.ibm.com/xsp/custom">
<xp:this.data>
<xp:dominoDocument var="docDemo" formName="fDemo"></xp:dominoDocument>
</xp:this.data>
<xc:ccLayout>
<xp:panel>
<div class="form-horizontal">
<legend>Examples Form controls</legend>
<!-- standard text field -->
```

```
<div class="control-group">
<xp:label value="Your name" id="label2" for="inputName" styleClass="control-label"></xp:label>
<div class="controls">
<xp:inputText id="inputName" value="#{docDemo.name}"></xp:inputText>
</div>
</div>
<!-- standard text field with placeholder -->
<div class="control-group">
<xp:label value="Email" id="label1" for="inputEmail" styleClass="control-label"></xp:label>
<div class="controls">
<xp:inputText id="inputEmail" value="#{docDemo.email}">
<xp:this.attrs>
<xp:attr name="placeholder" value="Email"></xp:attr>
</xp:this.attrs>
</xp:inputText>
<span class="help-inline">
Text field with help text here
</span>
</div>
</div>
<!-- dropdown -->
<div class="control-group">
<xp:label value="Favorite color" id="label3" for="comboBox1" styleClass="control-label"></xp:label>
<div class="controls">
<xp:comboBox id="comboBox1" value="#{docDemo.color}">
<xp:selectItem itemLabel="Red" itemValue="red"></xp:selectItem>
<xp:selectItem itemLabel="Green" itemValue="green"></xp:selectItem>
<xp:selectItem itemLabel="Blue" itemValue="blue"></xp:selectItem>
</xp:comboBox>
</div>
</div>
<!-- textarea -->
<div class="control-group">
<xp:label value="About me" id="label4" for="inputAboutMe" styleClass="control-label"></xp:label>
<div class="controls">
<xp:inputTextarea id="inputAboutMe" value="#{docDemo.about}"></xp:inputTextarea>
</div>
</div>
```

```
<div class="form-actions">

<xp:button value="Save" id="button1" styleClass="btn btn-primary">

<i class="icon-ok icon-white" />

<xp:eventHandler event="onclick" submit="true" refreshMode="complete" immediate="false" save="true"></xp:e
ventHandler></xp:button>

<xp:button styleClass="btn" value="Cancel" id="button2"><xp:eventHandler event="onclick" submit="true" ref
reshMode="complete" immediate="true" save="false"></xp:eventHandler></xp:button>

</div>

</div>

</xp:panel>

</xc:ccLayout><xp:div style="margin-left:-24px">

</xp:div>

</xp:view>
```

In your browser it will look as followed:



Form walkthrough

The (traditional) Notes form that should be used as data schema:

```
<xp:this.data>

<xp:dominoDocument var="docDemo" formName="fDemo"></xp:dominoDocument>

</xp:this.data>
```

Form's in Bootstrap are by default vertical aligned, e.g. grouped labels and input fields are paired below each other. If you want to align them horizontally add:

```
<div class="form-horizontal">
```

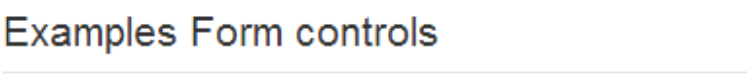
This will transform:

into:

The line:

```
<legend>Examples Form controls</legend>
```

adds a nice form header for the form:



Labels and controls are wrapped via the control-group class. To wrap a label to a control add the control-label class to the label. Any associated controls are wrapped within a control class.

```
<div class="control-group">
<xp:label value="Your name" id="label2" for="inputName" styleClass="control-label"></xp:label>
<div class="controls">
<xp:inputText id="inputName" value="#{docDemo.name}"></xp:inputText>
</div>
```

The placeholder attribute specifies a short hint that describes the expected value for an input field. You can provide help text via a span with the help-inline class.

```
<div class="control-group">
<xp:label value="Email" id="label1" for="inputEmail" styleClass="control-label"></xp:label>
<div class="controls">
<xp:inputText id="inputEmail" value="#{docDemo.email}">
<xp:this.attrs>
<xp:attr name="placeholder" value="Email"></xp:attr>
</xp:this.attrs>
</xp:inputText>
<span class="help-inline">
Text field with help text here
</span>
```

```
</div>
</div>
```

The combo box has nothing particular. In case you want to increase/decrease the width of the combo box you can use the span1 to span12 style class.

```
<div class="control-group">
<xp:label value="Favorite color" id="label3" for="comboBox1" styleClass="control-label"></xp:label>
<div class="controls">
<xp:comboBox id="comboBox1" value="#{docDemo.color}">
<xp:selectItem itemLabel="Red" itemValue="red"></xp:selectItem>
<xp:selectItem itemLabel="Green" itemValue="green"></xp:selectItem>
<xp:selectItem itemLabel="Blue" itemValue="blue"></xp:selectItem>
</xp:comboBox>
</div>
</div>
```

On the bottom of the form appear the form actions. Place them within a div with the form-actions class to visually emphasize the buttons.

Bootstrap has a rich set of different buttons: http://getbootstrap.com/2.3.2/base-css.html#buttons. The btn-primary provides extra visual weight and identifies the primary action in a set of buttons.

You can add an icon to a button: http://getbootstrap.com/2.3.2/base-css.html#icons. In case you work with a dark UI or element (the btn-primary gives a button a dark blue background) you can add color to the icon via the icon-white class.

```
<div class="form-actions">
<xp:button value="Save" id="button1" styleClass="btn btn-primary">
<i class="icon-ok icon-white" />
<xp:eventHandler event="onclick" submit="true" refreshMode="complete" immediate="false" save="true"></xp:e
ventHandler></xp:button>
<xp:button styleClass="btn" value="Cancel" id="button2"><xp:eventHandler event="onclick" submit="true" ref
reshMode="complete" immediate="true" save="false"></xp:eventHandler></xp:button>
</div>
</div>
```

That is it for a simple form. We will now add some more advanced controls.

Template update

You have noticed the Form header disappeared under the navigation bar. In order to let it appear on the right place we have to over-ride the padding properties for the Body element.

- Create a new stylesheet e.g. name it bootstrap-extended.css.
- Add the following code to it:

```
body { padding-top: 60px; padding-bottom: 40px; }

/* margin between icons and text in buttons/ links */
button > i, a > i {  margin-right: 4px; }

/* margin between 2 buttons / button links */
a + a, button + button, a + button, button + a { margin-left: 5px; }

/*forms in modals shouldn't have a bottom margin*/
.modal form {margin-bottom: 0;}

/*for this site only: style images in article*/
article img {
          margin: 10px 0;
          padding: 5px;
          border: 1px solid #c4c8cc;
          -moz-box-shadow: 4px 4px 6px #888;
          -moz-border-radius: 4px;
          -webkit-box-shadow: 4px 4px 6px #888;
          -webkit-border-radius: 4px;
}
```

The code above also overrides some other default Bootstrap behavior.

Since TB is heavily based on jQuery we add a JavaScript function that will help to select input fields in XPages. Normal jQuery selectors will not work because:

1) The server assigns a dynamic name to the field at run time.
2) The new id tag name contains colons which are special characters in jQuery selectors.

In order to get the function available do as followed:

- Create a new JavaScript code library e.g. jQuerySelectorForXPages.
- Paste in the code you find below.

```
/*
 * x$ function to be able to use the jQuery selectors with XPage id's
 * function is called using x$("#{id:inputText1}", " parameters").
 * by Mark Roden, http://openntf.org/XSnippets.nsf/snippet.xsp?id=x-jquery-selector-for-xpages
 */

function x$(idTag, param){ //Updated 18 Feb 2012
```

```
   idTag=idTag.replace(/:/gi, "\\:")+(param ? param : "");
   return($("#"+idTag));
}
```

Update your bootstrap Theme design element by adding the following code:

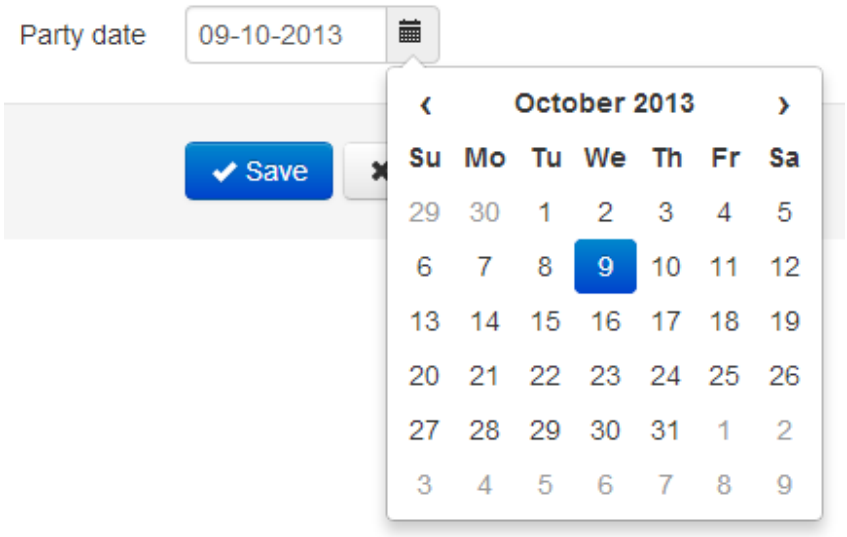
```
<!-- jquery -->
<resource>
          <content-type>application/x-javascript</content-type>
          <href>jQuerySelectorForXPages.js</href>
</resource>
```

Advanced Form controls

Bootstrap supports also more advanced controls. We will discuss them in the next chapters.

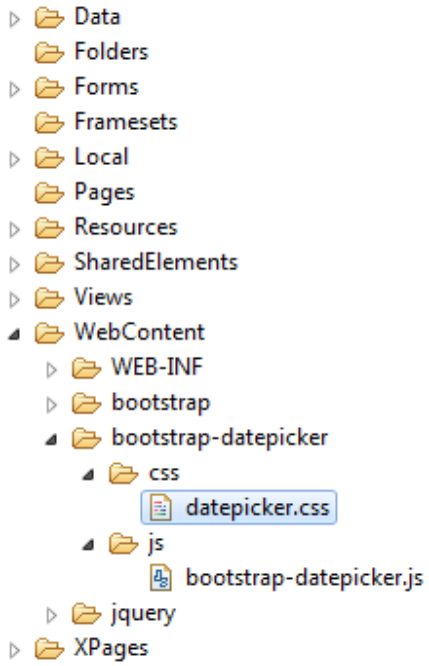
Date picker

A date picker control makes it easier for a user to enter a date. Here follows a description to add a date picker to the form.

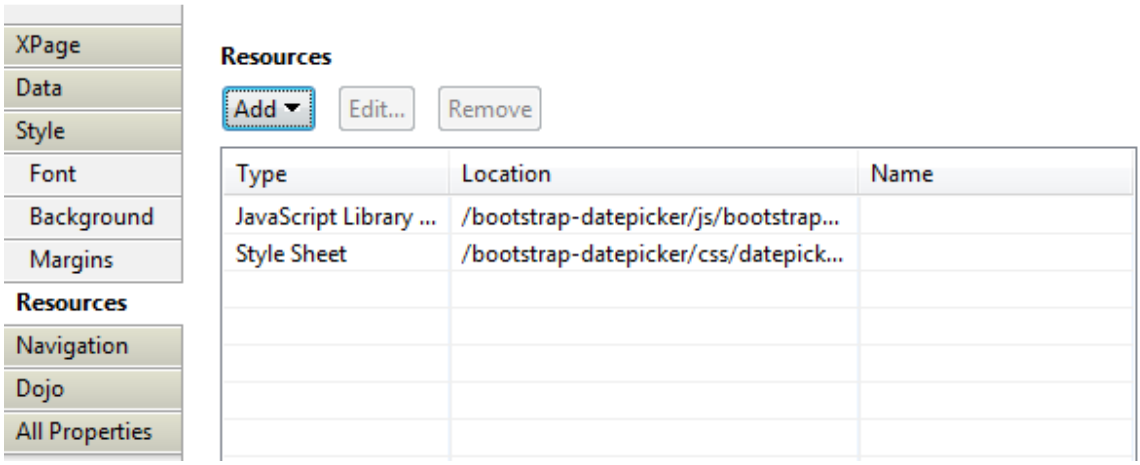


Download the required source files here: http://aymkdn.github.io/Datepicker-for-Bootstrap/.

Place the JavaScript and Stylesheet in the WebContent folder as followed:



Add the files as resources to your XPage:



Or in code:

```
<!-- js & css for date picker plugin -->
<xp:this.resources>
<xp:script src="/bootstrap-datepicker/js/bootstrap-datepicker.js" clientSide="true"></xp:script>
<xp:styleSheet href="/bootstrap-datepicker/css/datepicker.css"></xp:styleSheet>
</xp:this.resources>
```

Because it is very likely that you have date picker controls across your application we will build a custom control for it so it will be easier to re-use it. You might consider to move the resources above in this custumer control.

The custom control will take 3 properties:

- fieldDataSource.
- fieldName.
- dateFormat.

The fieldDataSource is the data source for the field, e.g. the variable name for your document data binding. The fieldName is the corresponding field on that data source. The dateFormat is the format you want to save the selected date e.g. dd-mm-yyyy.

Below is the code for this custom control:

```
<?xml version="1.0" encoding="UTF-8" ?>

<xp:view xmlns:xp="http://www.ibm.com/xsp/core">



    <!-- note: this date picker depends on a number of files:

           - Twitter Bootstrap

           - bootstrap-datepicker (http://www.eyecon.ro/bootstrap-datepicker/)

           - jQuery

           -->



    <xp:text escape="true" id="computedField1" value="#{compositeData.fieldDataSource[compositeData.fieldN
ame]}" rendered="#{javascript:!compositeData.fieldDataSource.isEditable()}">

        <xp:this.converter>

            <xp:convertDateTime type="date" dateStyle="full"></xp:convertDateTime>

        </xp:this.converter>

    </xp:text>

    <xp:div styleClass="input-append date" id="dateC" rendered="#{javascript:compositeData.fieldDataSource
.isEditable()}">

        <xp:this.attrs>

            <xp:attr name="data-date">

                <xp:this.value>

                    <![CDATA[#{javascript://default date in the picker: today or the entered value

var itDate:NotesItem = compositeData.fieldDataSource.getDocument().getFirstItem(compositeData.fieldName);

var dt:java.util.Date = @Now();

if (itDate != null) {

          if (itDate.getDateTimeValue() != null) {

                    dt = itDate.getDateTimeValue().toJavaDate();

          }

}

(new java.text.SimpleDateFormat( compositeData.dateFormat ) ).format(dt );}]]>

                </xp:this.value>
```

```
            </xp:attr>
            <xp:attr name="data-date-format" value="#{javascript:compositeData.dateFormat.toLowerCase()}">
            </xp:attr>
        </xp:this.attrs>
        <xp:inputText styleClass="input-small" size="16" id="inputDatum" value="#{compositeData.fieldDataS
ource[compositeData.fieldName]}">
            <xp:this.converter>
                <xp:convertDateTime type="date" pattern="${compositeData.dateFormat}"></xp:convertDateTime
>
            </xp:this.converter>
        </xp:inputText>
        <span class="add-on"><i class="icon-calendar"></i></span>
    </xp:div>
    <xp:scriptBlock id="scriptBlock2">
        <xp:this.value>
            <![CDATA[
$(function(){
//initialize date picker, hide it after selecting a date
x$('#{id:dateC}').datepicker()
                    .on('changeDate', function(ev){
                                                x$('#{id:dateC}').datepicker('hide');
                                        });
});
]]>
        </xp:this.value>
    </xp:scriptBlock>
</xp:view>
```

Code walkthrough

First the code for the text field that will display the selected date in read mode. It uses the fieldDataSource and fieldName properties.

Next is a div element that contains the input field and picker icon, it is rendered only in edit mode.

Then we the input field that displays the selected date. It uses the dateFormat property to display the selected date in desired format.

At last we display an icon. Clicking the icon initiates the appearance of the date picker.

```
<xp:text escape="true" id="computedField1" value="#{compositeData.fieldDataSource[compositeData.fieldName]
}" rendered="#{javascript:!compositeData.fieldDataSource.isEditable()}">
        <xp:this.converter>
            <xp:convertDateTime type="date" dateStyle="full"></xp:convertDateTime>
```

```
        </xp:this.converter>
    </xp:text>
    <xp:div styleClass="input-append date" id="dateC" rendered="#{javascript:compositeData.fieldDataSource
.isEditable()}">
        <xp:this.attrs>
            <xp:attr name="data-date">
                <xp:this.value>
                    <![CDATA[#{javascript://default date in the picker: today or the entered value
var itDate:NotesItem = compositeData.fieldDataSource.getDocument().getFirstItem(compositeData.fieldName);
var dt:java.util.Date = @Now();
if (itDate != null) {
          if (itDate.getDateTimeValue() != null) {
                    dt = itDate.getDateTimeValue().toJavaDate();
          }
}
(new java.text.SimpleDateFormat( compositeData.dateFormat ) ).format(dt );}]]>
                </xp:this.value>
            </xp:attr>
            <xp:attr name="data-date-format" value="#{javascript:compositeData.dateFormat.toLowerCase()}">
            </xp:attr>
        </xp:this.attrs>
        <xp:inputText styleClass="input-small" size="16" id="inputDatum" value="#{compositeData.fieldDataS
ource[compositeData.fieldName]}">
            <xp:this.converter>
                <xp:convertDateTime type="date" pattern="${compositeData.dateFormat}"></xp:convertDateTime
>
            </xp:this.converter>
        </xp:inputText>
        <span class="add-on"><i class="icon-calendar"></i></span>
    </xp:div>
```

Further we have a script block control. It outputs client script. It uses the jQuery selector function in the jQuerySelectorForXPages script library. Whenever the date is changed in the div element with id dateC it will hide the date picker.

```
    <xp:scriptBlock id="scriptBlock2">
        <xp:this.value>
            <![CDATA[
$(function(){
//initialize date picker, hide it after selecting a date
x$('#{id:dateC}').datepicker()
```

```
                    .on('changeDate', function(ev){
                                                x$('#{id:dateC}').datepicker('hide');
                                        });
});
]]>
        </xp:this.value>
    </xp:scriptBlock>
</xp:view>
```

Implementation

So how do you implement this custom control for a date picker? Easy: Just drop the custom control where you want the date picker to appear. Also provide the data for the custom properties.

In the following example we have defined a new control group and placed the data picker custom control (ccFormDatePicker) within the div element with the controls class.

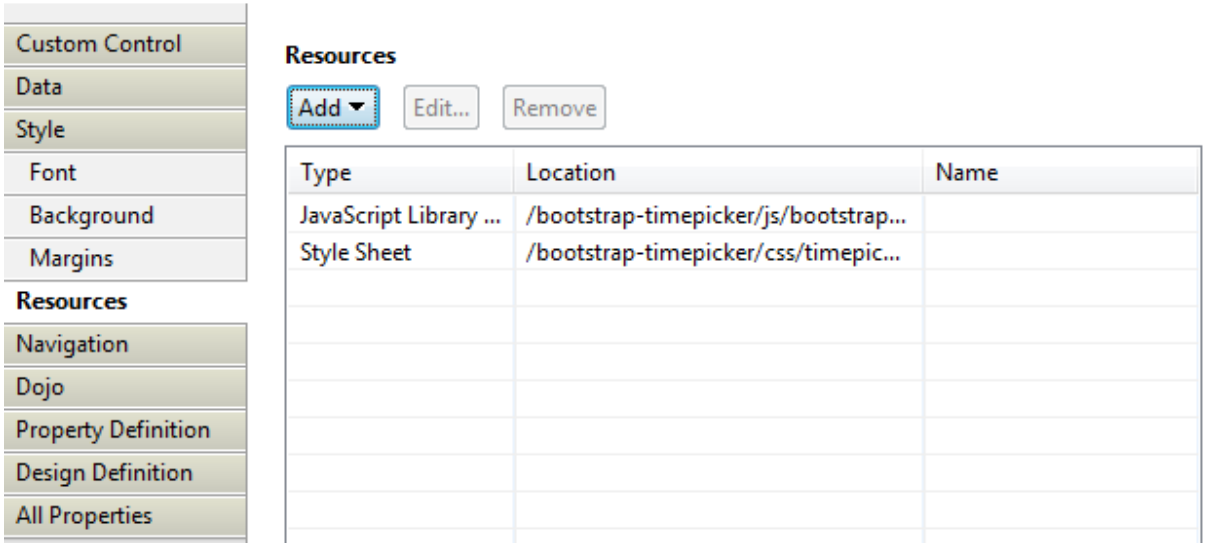
```
<div class="control-group">
          <xp:label value="Party date" id="label5"
                    for="inputDate" styleClass="control-label">
          </xp:label>
          <div class="controls">
                    <xc:ccFormDatePicker dateFormat="dd-mm-yyyy"
                              fieldDataSource="#{docDemo}" fieldName="date">
                    </xc:ccFormDatePicker>
          </div>
</div>
```

Time picker

A time picker allows a user easily select a time for a text input using your mouse or keyboards arrow keys. The user can select an hour and minutes.

You can download the required resources here: http://jdewit.github.io/bootstrap-timepicker/.

Add the files as resources to your XPage:



Or in code:

```
<!-- js & css for time picker plugin -->
<xp:this.resources>
<xp:script src="/bootstrap-timepicker/js/bootstrap-timepicker.js" clientSide="true"></xp:script>
<xp:styleSheet href="/bootstrap-timepicker/css/timepicker.css"></xp:styleSheet>
</xp:this.resources>
```

Because it is very likely that you have date picker controls across your application we will build a custom control for it so it will be easier to re-use it.

The custom control will take 6 properties:

- fieldDataSource.
- fieldName.
- requiredMessage.
- inputId.
- initialDefault.
- alternativeIdentifier.

The fieldDataSource is the data source for the field, e.g. the variable name for your document data binding. The fieldName is the corresponding field on that data source. The requiredMessage is the message you want to appear in an alert message box when no time is selected. The inputId is the id which will be dynamically assigned to the edit box to store the time in. The initialDefault is the initial date in the date picker when it's being opened for the first time.

Below is the code for this custom control:

```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core">
```

```
<!--

          note: this date picker depends on a number of files:

          - Twitter Bootstrap

          - Bootstrap-timepicker (http://jdewit.github.com/bootstrap-timepicker/)

          - jQuery

-->

          <xp:text escape="true" id="computedField1" value="#{compositeData.fieldDataSource[compositeData.
fieldName]}" rendered="#{javascript:!compositeData.fieldDataSource.isEditable()}" style="float:left;">

                    <xp:this.converter>

                              <xp:convertDateTime type="time" timeStyle="short" />

                    </xp:this.converter>

          </xp:text>

          <xp:div styleClass="input-append bootstrap-timepicker-component" style="float:left;" rendered="#
{javascript:compositeData.fieldDataSource.isEditable()}">

                    <xp:inputText id="${javascript:compositeData.inputId}" value="#{compositeData.fieldDat
aSource[compositeData.fieldName]}" required="true" style="width:43px;">

                              <xp:this.converter>

                                        <xp:convertDateTime type="time" timeStyle="short" />

                              </xp:this.converter>

                              <xp:this.validators>

                                        <xp:validateRequired message="#{javascript:compositeData.required
Message}" />

                              </xp:this.validators>



                              <xp:this.defaultValue><![CDATA[#{javascript:( compositeData.intialDefault !
= null && compositeData.intialDefault != "" ? compositeData.intialDefault : "19:00" )}]]></xp:this.default
Value>

                              <xp:this.styleClass><![CDATA[#{javascript:"timepicker-default " + composite
Data.alternativeIdentifier}]]></xp:this.styleClass>

                    </xp:inputText>

                    <span class="add-on"><i class="icon-time" /></span>

          </xp:div>

          <!--

                    the next clientside JavaScript is computed since it needs to function

                    in read and edit mode

          -->

          <xp:text escape="false" id="computedField2">

                    <xp:this.value><![CDATA[#{javascript://default date in the picker: today or the entere
d value

var itDate:NotesItem = compositeData.fieldDataSource.getDocument().getFirstItem(compositeData.fieldName);

var dt;

if (itDate != null) {
```

```
          if (itDate.getDateTimeValue() != null) {
                    dt = itDate.getDateTimeValue().toJavaDate();
                    dt = (new java.text.SimpleDateFormat( "HH:mm" ) ).format( dt );
          }
} else {
          dt = ( compositeData.intialDefault == null ? "19:00" : compositeData.intialDefault);
}
"<script type=\"text/javascript\">" +
"$(function(){" +
          "x$('" + getClientId( compositeData.inputId ) + "').timepicker( {" + "minuteStep: 10 ," + "showM
eridian: false," + "defaultTime:'" + dt + "'" + "});" + "});" +
"</script>";}]]>
                    </xp:this.value>
          </xp:text>
</xp:view
```

Code walkthrough

First the code for the text field that will display the selected date in read mode. It uses the fieldDataSource and fieldName properties.

```
<xp:text escape="true" id="computedField1" value="#{compositeData.fieldDataSource[compositeData.fieldName]
}" rendered="#{javascript:!compositeData.fieldDataSource.isEditable()}" style="float:left;">
<xp:this.converter>
<xp:convertDateTime type="time" timeStyle="short" />
</xp:this.converter>
</xp:text>
```

Next is the time picker control. It will only be shown in edit mode. It contains an Edit Box control to capture the selected time. Right beside the input field an icon is placed. When you click on it the time picker appears.

```
<xp:div styleClass="input-append bootstrap-timepicker-component" style="float:left;" rendered="#{javascrip
t:compositeData.fieldDataSource.isEditable()}">
<xp:inputText id="${javascript:compositeData.inputId}" value="#{compositeData.fieldDataSource[compositeDat
a.fieldName]}" required="true" style="width:43px;">
<xp:this.converter>
<xp:convertDateTime type="time" timeStyle="short" />
</xp:this.converter>
<xp:this.validators>
<xp:validateRequired message="#{javascript:compositeData.requiredMessage}" />
</xp:this.validators>
<xp:this.defaultValue><![CDATA[#{javascript:( compositeData.intialDefault != null && compositeData.intialD
efault != "" ? compositeData.intialDefault : "19:00" )}]]></xp:this.defaultValue>
```

```
<xp:this.styleClass><![CDATA[#{javascript:"timepicker-default " + compositeData.alternativeIdentifier}]]><
/xp:this.styleClass>

</xp:inputText>

<span class="add-on"><i class="icon-time" /></span>

</xp:div>
```

Next is a computed field which eventually will output a client-side JavaScript.

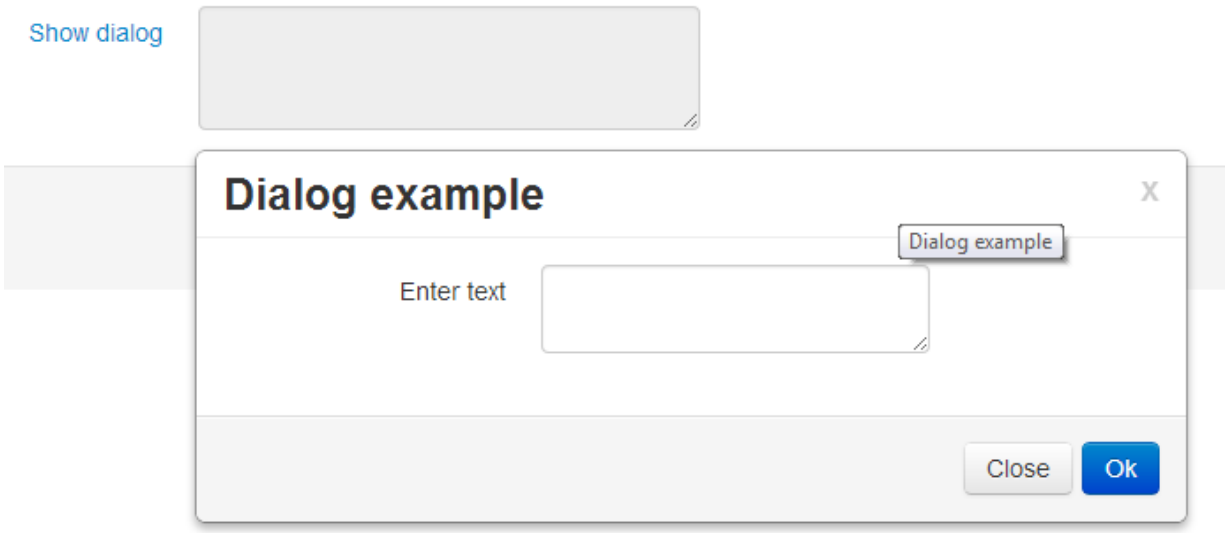
```
<xp:text escape="false" id="computedField2">

<xp:this.value><![CDATA[#{javascript://default date in the picker: today or the entered value

var itDate:NotesItem = compositeData.fieldDataSource.getDocument().getFirstItem(compositeData.fieldName);

var dt;

if (itDate != null) {

if (itDate.getDateTimeValue() != null) {

dt = itDate.getDateTimeValue().toJavaDate();

dt = (new java.text.SimpleDateFormat( "HH:mm" ) ).format( dt );

}

} else {

dt = ( compositeData.intialDefault == null ? "19:00" : compositeData.intialDefault);

}

"<script type=\"text/javascript\">" +

"$(function(){" +

"x$('" + getClientId( compositeData.inputId ) + "').timepicker( {" + "minuteStep: 10 ," + "showMeridian: f
alse," + "defaultTime:'" + dt + "'" + "});" + "});" +

"</script>";}]]>

</xp:this.value>

</xp:text>
```

Dialog

A dialog box should improve the dialog with the user, e.g. by cutting the portion of content in smaller bits.

The Dialog control in the Extension Library looks by default not like the *Modal* component in TB. However by changing some properties and the help of jQuery the Dialog control can look the same.

In our next example we will build a standard Dialog component and configure it to reflect a Bootstrap styled dialog:



Our example contains:

- A Link control that is presented as a label. The link will show the dialog.
- A Multi line Edit Box (text area) that will display the text entered in the dialog.
- A Dialog control which contains a text area and buttons to hide the dialog and copy the entered text in the other text area.

```
<div class="control-group">
          <!-- link control to show dialog -->
          <xp:link escape="true" text="Show dialog" id="link1" styleClass="control-label">
                    <xp:eventHandler event="onclick" submit="true" refreshMode="partial" refreshId="dialog
1">
                              <xp:this.action><![CDATA[#{javascript:getComponent("dialog1").show();}]]></
xp:this.action>
                    </xp:eventHandler>
          </xp:link>
          <div class="controls">
          <!-- multiline edit box control to display text entered in dialog -->
          <xp:inputTextarea id="inputTextarea2" value="#{docDemo.dialog}" styleClass="input-xlarge unedita
ble-input" rows="3" style="min-height:60px;" disabled="true">
          </xp:inputTextarea>
          <!-- dialog control -->
          <xe:dialog title="#{javascript:compositeData.dialogTitle}"
                              styleClass="modal" style="margin-left: inherit" id="dialog1">
                              <xe:this.onShow><![CDATA[var titleBar = $(".modal .dijitDialogTitleBar").ad
dClass("modal-header");
          //replace titlenode
```

```
          var titleNode = $(".dijitDialogTitle", titleBar);
          var title = titleNode.text();
          titleNode.remove();
          var close = $(".dijitDialogCloseIcon", titleBar).removeClass("dijitDialogCloseIcon").addClass("c
lose");
          titleBar.append("<h3>" + title + "</h3>");
          ]]></xe:this.onShow>
                              <xp:panel styleClass="modal-body">
                                        <div class="form-horizontal">
                                                  <div class="control-group">
                                                            <xp:label value="Enter text" id="label6" for
="inputTextArea" styleClass="control-label" />
                                                            <div class="controls">
                                                                      <xp:inputTextarea id="inputTextare
a" value="#{viewScope.textDialog}" />
                                                            </div>
                                                  </div>
                                        </div>
                              </xp:panel>
                              <xe:dialogButtonBar id="dialogButtonBar2" styleClass="modal-footer">
                                        <xp:button value="Close" id="button1" styleClass="btn">
                                                  <xp:eventHandler event="onclick" submit="true" refresh
Mode="partial" refreshId="dialog1">
                                                            <xp:this.action><![CDATA[#{javascript:getCom
ponent("dialog1").hide()}]]></xp:this.action>
                                                  </xp:eventHandler>
                                        </xp:button>
                                        <xp:button value="Ok" id="button2" styleClass="btn btn-primary">
                                                  <xp:eventHandler event="onclick" submit="true" refresh
Mode="partial" refreshId="ccLayout">
                                                            <xp:this.action><![CDATA[#{javascript:var te
xt = getComponent("inputTextarea").getValue();
docDemo.replaceItemValue("dialog",text);
getComponent("dialog1").hide();}]]></xp:this.action>
                                                  </xp:eventHandler>
                                        </xp:button>
                              </xe:dialogButtonBar>
                    </xe:dialog>
          </div>
</div>
```

Code walkthrough

We start with a Link control. When clicked a show method on the Dialog control is performed:

```
<!-- link control to show dialog -->

<xp:link escape="true" text="Show dialog" id="link1" styleClass="control-label">

          <xp:eventHandler event="onclick" submit="true" refreshMode="partial" refreshId="dialog1">

                    <xp:this.action><![CDATA[#{javascript:getComponent("dialog1").show();}]]></xp:this.act
ion>

          </xp:eventHandler>

</xp:link>
```

The following text area will get the text entered in the text area on the Dialog control. We disable the input option and provide a min-weight otherwise it will not look like a text area but a regular input box.

```
<xp:inputTextarea id="inputTextarea2" value="#{docDemo.dialog}" styleClass="input-xlarge uneditable-input"
rows="3" style="min-height:60px;" disabled="true"></xp:inputTextarea>
```

The Dialog control will give the corresponding style for the Modal component in TB.

```
<xe:dialog title="#{javascript:compositeData.dialogTitle}" styleClass="modal" style="margin-left: inherit"
id="dialog1">
```

When the Dialog control is showed the title bar of the Dojo Dijit is altered by adding the class model-header. The title bar gets the text from the Dojo Dijit title, which will be removed also.

The close button in the Dojo Dijit loses its dijit style class and a TB modal class will be added to make it appear properly.

At last the title is added in the Modal title bar.

```
<xe:this.onShow><![CDATA[var titleBar = $(".modal .dijitDialogTitleBar").addClass("modal-header");

//replace titlenode

var titleNode = $(".dijitDialogTitle", titleBar);

var title = titleNode.text();

titleNode.remove();

var close = $(".dijitDialogCloseIcon", titleBar).removeClass("dijitDialogCloseIcon").addClass("close");

titleBar.append("<h3>" + title + "</h3>");

]]></xe:this.onShow>
```

On the bottom of the dialog we have 2 buttons:

- A Close button to hide the Dialog control.
- A OK button that hides the Dialog control and copies the value from the text area in the Dialog control to the one on our form.

```
          <xe:dialogButtonBar id="dialogButtonBar2" styleClass="modal-footer">

                    <xp:button value="Close" id="button1" styleClass="btn">

                              <xp:eventHandler event="onclick" submit="true" refreshMode="partial" refres
hId="dialog1">
```

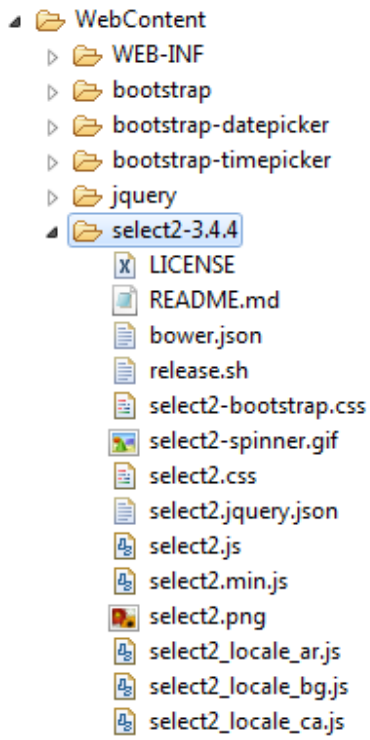
```
                                        <xp:this.action><![CDATA[#{javascript:getComponent("dialog1").hid
e()}]]></xp:this.action>
                                </xp:eventHandler>
                        </xp:button>
                        <xp:button value="Ok" id="button2" styleClass="btn btn-primary">
                                <xp:eventHandler event="onclick" submit="true" refreshMode="partial" refres
hId="ccLayout">
                                        <xp:this.action><![CDATA[#{javascript:var text = getComponent("in
putTextarea").getValue();
docDemo.replaceItemValue("dialog",text);
getComponent("dialog1").hide();}]]></xp:this.action>
                                </xp:eventHandler>
                        </xp:button>
```

Select box

Select boxes (Combobox control) are plain boring in XPages. Select2 is a jQuery based replacement for select boxes. It supports searching, remote data sets, and infinite scrolling of results.

Perform the following steps to enable Select2:

- Download the resources from GitHub.
- Add the Select2 library to the WebContent directory in your application and keep the same folder structure.



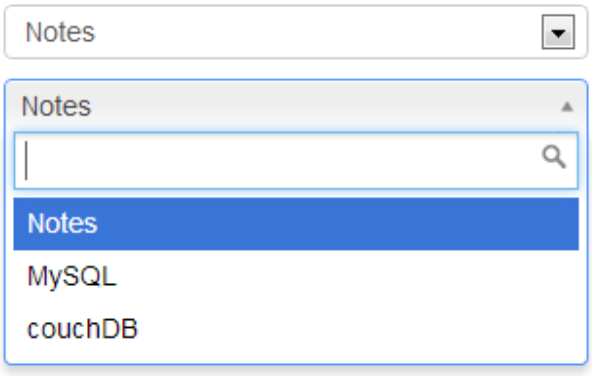
- Include select2.min.js and select2.css on your XPages:

```
<xp:this.resources>
          <xp:script src="/select2-3.4.4/select2.min.js" clientSide="true">
          </xp:script>
          <xp:styleSheet href="/select2-3.4.4/select2.css"></xp:styleSheet>
</xp:this.resources>
```

- Convert a regular (XPage) combobox control into a Select2 variant by adding a few lines of JavaScript.

Example Basic select box

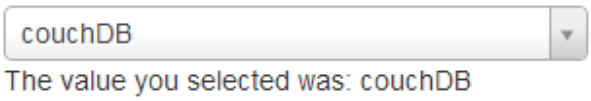
The following code will transform a regular Combobox control into a select box that allows you to search on any string contained in any option:




```
<xp:scriptBlock id="scriptBlock2">
  <xp:this.value><![CDATA[
  $(document).ready(
    function() {
      x$( "#{id:comboBox2}" ).select2();
    }
  );
  ]]></xp:this.value>
</xp:scriptBlock>
<xp:comboBox id="comboBox2" style="min-width:300px;">
          <xp:selectItem itemLabel="Notes" itemValue="Notes"></xp:selectItem>
          <xp:selectItem itemLabel="MySQL" itemValue="MySQL"></xp:selectItem>
          <xp:selectItem itemLabel="couchDB" itemValue="couchDB"></xp:selectItem>
</xp:comboBox>
```

On change event

You can alter the script block so it will refresh another element that displays the selected value:




```
<xp:scriptBlock id="scriptBlock2">
          <xp:this.value><![CDATA[$(document).ready(
          function() {
                    x$( "#{id:comboBox2}" ).select2().on("change", function(e) {
          XSP.partialRefreshPost( "#{id:refreshThis}" );
          } )
          }
);]]></xp:this.value>
</xp:scriptBlock>
```

Here is the code for the Computed Text control with ID refreshThis:

```
<div>
          You selected: 
          <xp:text escape="true" id="refreshThis">
                    <xp:this.value><![CDATA[#{javascript:getComponent("comboBox2").getValue()}]]></xp:this
.value>
          </xp:text>
</div>
```

Example Multi-selects

The following code will transform a List Box control into a multi-select box that allows easy picking from the list. The selected values can easily be removed via the close button in the selected value:


```
<xp:scriptBlock id="scriptBlock1">
<xp:this.value>
<![CDATA[
  $(document).ready(
    function() { x$( "#{id:listBox1}" ).select2({
      placeholder : "Click to select 1 or more databases..." });
    }
  );
```

```
]]>
</xp:this.value>
</xp:scriptBlock>
<xp:listBox id="listBox1" style="width:300.0px" multiple="true">
          <xp:selectItem itemLabel="Notes" itemValue="Notes"></xp:selectItem>
          <xp:selectItem itemLabel="MySQL" itemValue="MySQL"></xp:selectItem>
          <xp:selectItem itemLabel="couchDB" itemValue="couchDB"></xp:selectItem>
</xp:listBox>
```

Buttons

You can apply button styles by using the btn class. You should these clases to only <a> and <button> elements for the best rendering.

Button		class=""	Description
Default	Default	btn	Standard gray button with gradient
Primary	Primary	btn btn-primary	Provides extra visual weight and identifies the primary action in a set of buttons
Info	Info	btn btn-info	Used as an alternative to the default styles
Success	Success	btn btn-success	Indicates a successful or positive action
Warning	Warning	btn btn-warning	Indicates caution should be taken with this action
Danger	Danger	btn btn-danger	Indicates a dangerous or potentially negative action
Inverse	Inverse	btn btn-inverse	Alternate dark gray button, not tied to a semantic action or use
Link	Link	btn btn-link	Deemphasize a button by making it look like a link while maintaining button behavior

In XPages you apply the class to the styleClass property:

```
<xp:button value="Standard button" styleClass="btn" id="button1"></xp:button>
```

Button sizes

In case you like larger or smaller buttons you can add .btn-large, .btn-small, or .btn-mini for additional sizes.

```
<xp:button value="Small button" styleClass="btn btn-mini" id="button8"></xp:button>
```

Icons

To make your buttons more outstanding you can also apply icons to them.

```
<xp:button value="Home" id="button4" styleClass="btn"><i class="icon-home" /></xp:button>

<xp:button value="Where's the fire?" id="button7" styleClass="btn btn-danger"><i class="icon-fire icon-whi
te" /></xp:button>
```

Style sheet override.css applies the correct relative location of the images:

```
/*change the (relative) location of images if resource aggregation is used*/

[class^="icon-"],

[class*=" icon-"] {

  background-image: url("../../../bootstrap/img/glyphicons-halflings.png");

}
```

An overview of applicable icons you can find here:

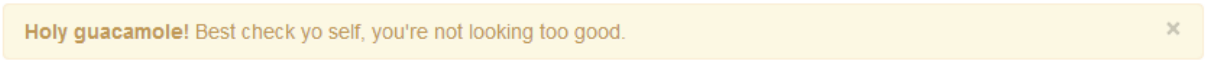
http://getbootstrap.com/2.3.2/base-css.html#icons

Alert/Messages

In XPages you have the Display Error(s) controls to display messages.

- The Display Error control displays error messages for one control.
- The Display Errors control displays error messages for multiple controls.

The equivalent in TB is called Alert message.



The following code is for a button that adds a message to the FacesMessage.

```
<xp:button value="Add error message" id="button2" styleClass="btn btn-danger">

<i class="icon-exclamation-sign icon-white" />

<xp:eventHandler event="onclick" submit="true"

          refreshMode="partial" refreshId="messagePanel">
```

```
          <xp:this.action><![CDATA[#{javascript:var msg = "<strong>Holy guacamole!</strong>Best check yo s
elf, you're not looking too good.";

facesContext.addMessage(null, new javax.faces.application.FacesMessage( javax.faces.application.FacesMessa
ge.SEVERITY_ERROR, msg, msg) );}]]></xp:this.action>

</xp:eventHandler>

</xp:button>
```

It refreshes a control.

Usefull resources

Resource	Remarks	Address
Bootstrap4XPage	Site focuses on using TB with XPages. You can find here snippets, demo's, themes and more.	http://www.bootstrap4xpages.com