Using resource bundles to store configuration information in your XPages application

Dan Soares

One of the XPages applications I had built needed to store application configuration information that an application administrator could manage without having to come to IT when configuration parameters changed. For those of us who have developed Notes client applications and have used profile documents, this is trivial. I thought about a couple of different approaches and stumbled upon this article while researching. The author recommended using resource bundles to store application level configuration information that could easily be accessed via SSJS or Java.

It was easy enough to do. Following the author's directions, I created the file resource application.properties and entered the key=value pairs. I loaded the properties file as a resource on my XPage.

```
<xp:this.resources>
<xp:bundle src="/application.properties" var="config"/>
</xp:this.resources>
```

I was then able to successfully retrieve the value of the property using Expression Language. In my case, I had a combo box (Semester) with a default value set to pull Current Semester from the properties file.

```
<xp:comboBox id="Semester" defaultValue="#{config.currentSemester}">
<xp:selectItem itemLabel="FA 2014"  itemValue="FA 2014" id="selectItem">
</xp:selectItem>
<xp:selectItem itemLabel="SP 2015"  itemValue="SP 2015">
</xp:selectItem>
<xp:selectItem itemLabel="FA 2015"  itemValue="FA 2015">
</xp:selectItem>
<xp:selectItem itemLabel="SP 2016"  itemValue="SP 2016">
</xp:selectItem>
<xp:selectItem itemLabel="FA 2016"  itemValue="FA 2016">
</xp:selectItem>
</xp:comboBox>
```

Pretty easy so far. And then, I ran into an unexpected snag. Apparently, it's not as trivial to set these property values as it is to retrieve them. Obviously, if an administrator was going to manage these configuration values, he would need a way to set them himself. And so I reached out to the very helpful XPages community and Sven Hasselbach provided me with a brilliant solution.

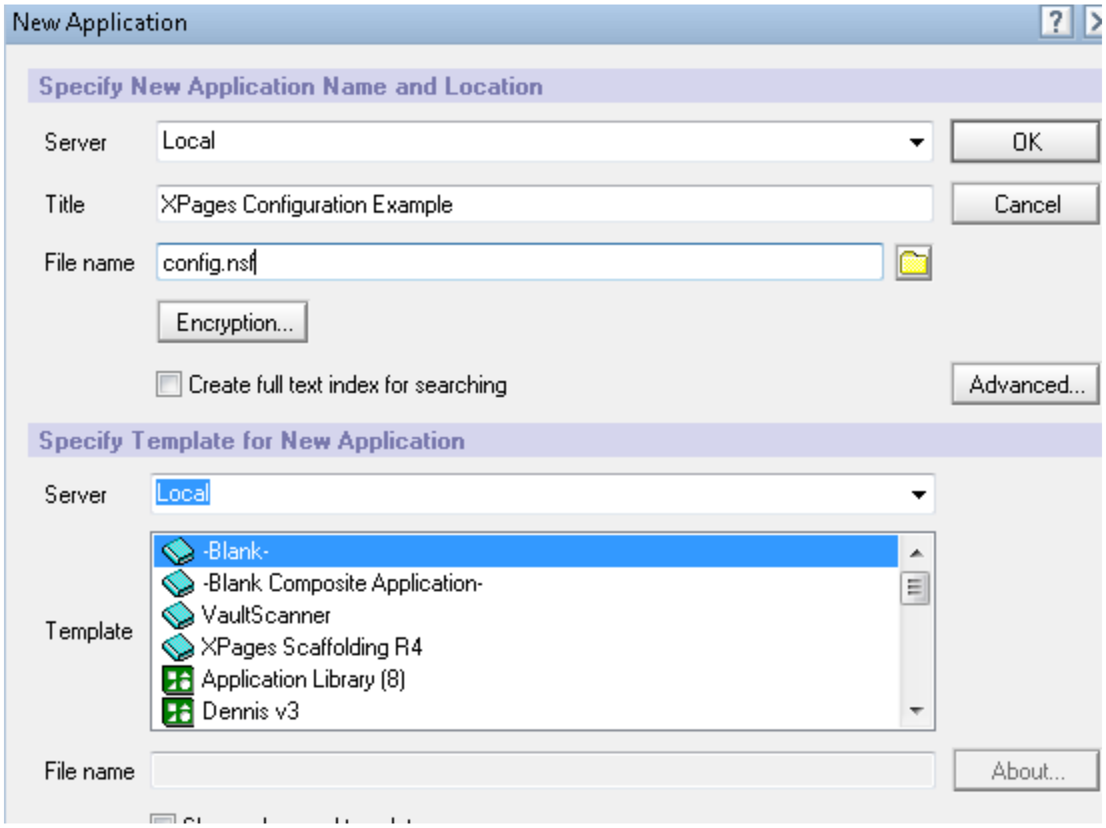
I will share how to implement this solution and have also provided a sample application that you can use to grab the code you need and follow along. The following elements are involved:

- A Java class created by Sven Hasselbach that uses the Java NAPI.
- Add the jar *lwpd.domino.napi.jar* to the build path.
- A custom control for the application configuration item that needs to be set by the application administrator.
- A custom control for the keyword field that needs to pull its default value from the configuration item.

- The file resource item application.properties with the needed key-value pair.
- And finally two XPages, one to hold the config custom control and the other to hold the form control

Let’s get started.

- Create your application (File - Application - New).



New Application

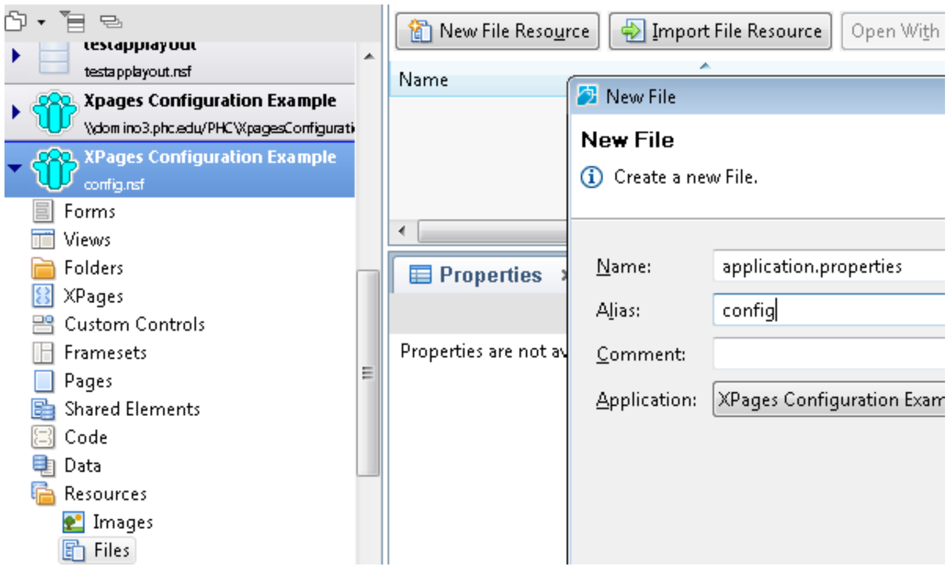
Specify New Application Name and Location

Server: Local

Title: XPages Configuration Example

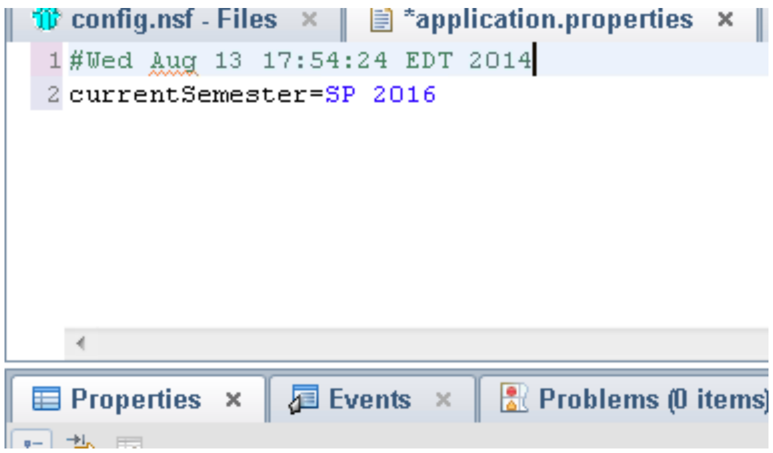
File name: config.nsf

- Create the file resource application.properties and give it the alias of config.



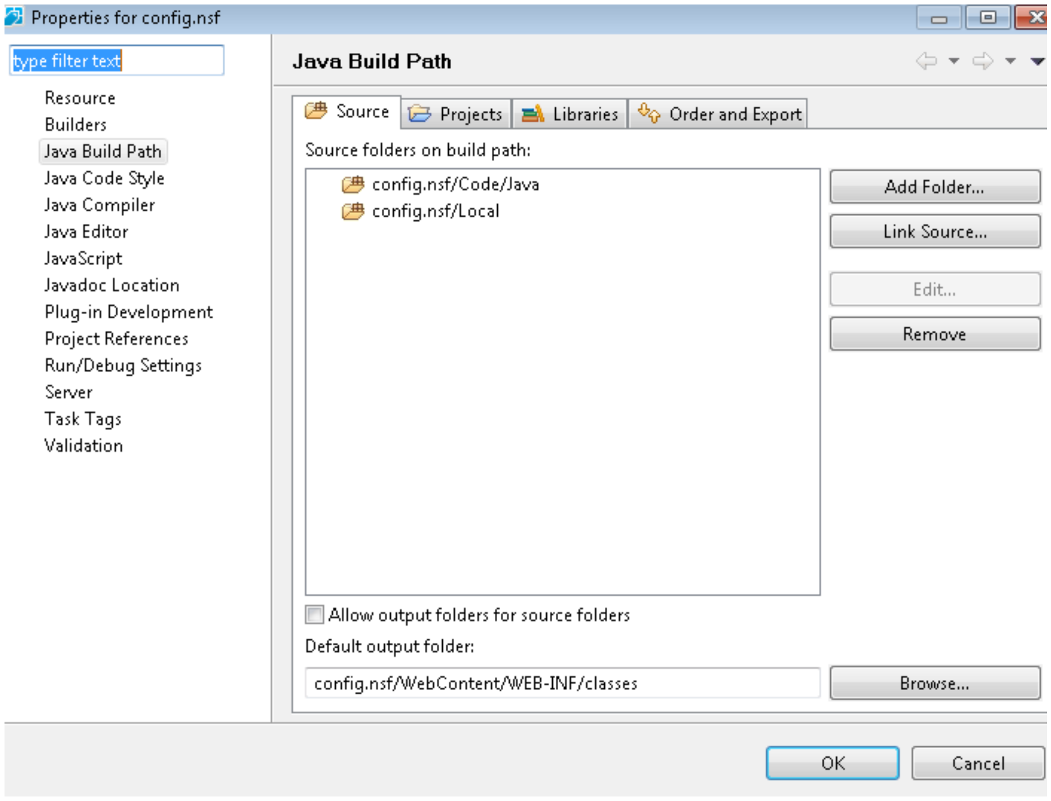
Name: application.properties
Alias: config
Comment:
Application: XPages Configuration Exam

- Add the key-value pair to the file and save it.



```
#Wed Aug 13 17:54:24 EDT 2014
currentSemester=SP 2016
```

- Switch to Package Explorer (Window – Show Eclipse Views – Package Explorer). We now need to add a folder to the build path of the project. Right click on the application name and select Properties from the context menu. Navigate to the Java build path section of the properties dialog.



Properties for config.nsf

type filter text

- Resource
- Builders
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- JavaScript
- Javadoc Location
- Plug-in Development
- Project References
- Run/Debug Settings
- Server
- Task Tags
- Validation

Java Build Path

Source | Projects | Libraries | Order and Export

Source folders on build path:

- config.nsf/Code/Java
- config.nsf/Local

Add Folder... | Link Source... | Edit... | Remove

Allow output folders for source folders

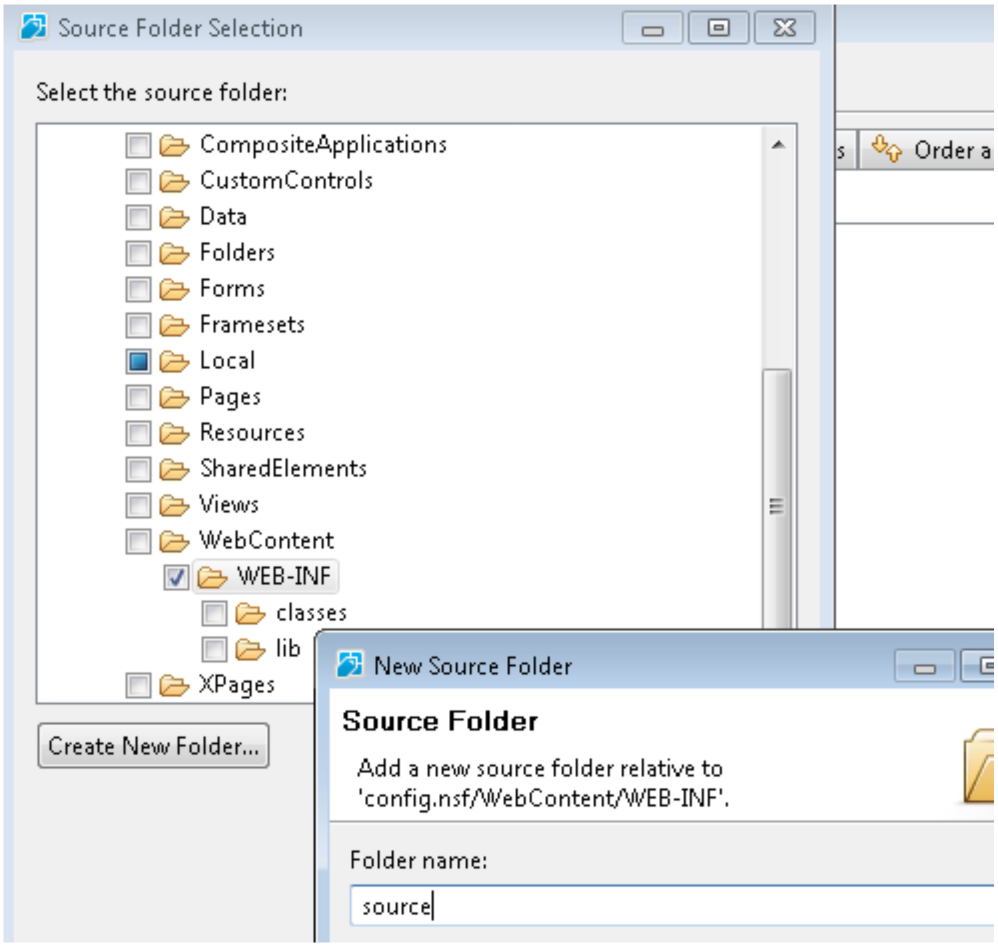
Default output folder:

config.nsf/WebContent/WEB-INF/classes

Browse...

OK | Cancel

Click on Add folder, navigate to the WEB-INF folder in the tree and click on Create New Folder. Name the folder ‘source’. Click Next and then Finish.


Source Folder Selection

Select the source folder:

- [ ] CompositeApplications
- [ ] CustomControls
- [ ] Data
- [ ] Folders
- [ ] Forms
- [ ] Framesets
- [■] Local
- [ ] Pages
- [ ] Resources
- [ ] SharedElements
- [ ] Views
- [ ] WebContent
  - [✓] WEB-INF
    - [ ] classes
    - [ ] lib
- [ ] XPages

Create New Folder...

New Source Folder

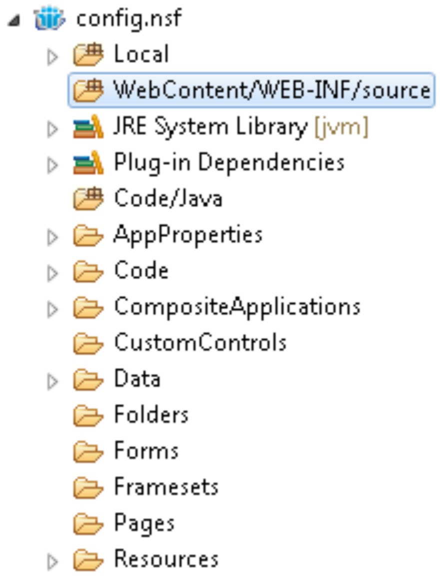
Source Folder

Add a new source folder relative to 'config.nsf/WebContent/WEB-INF'.

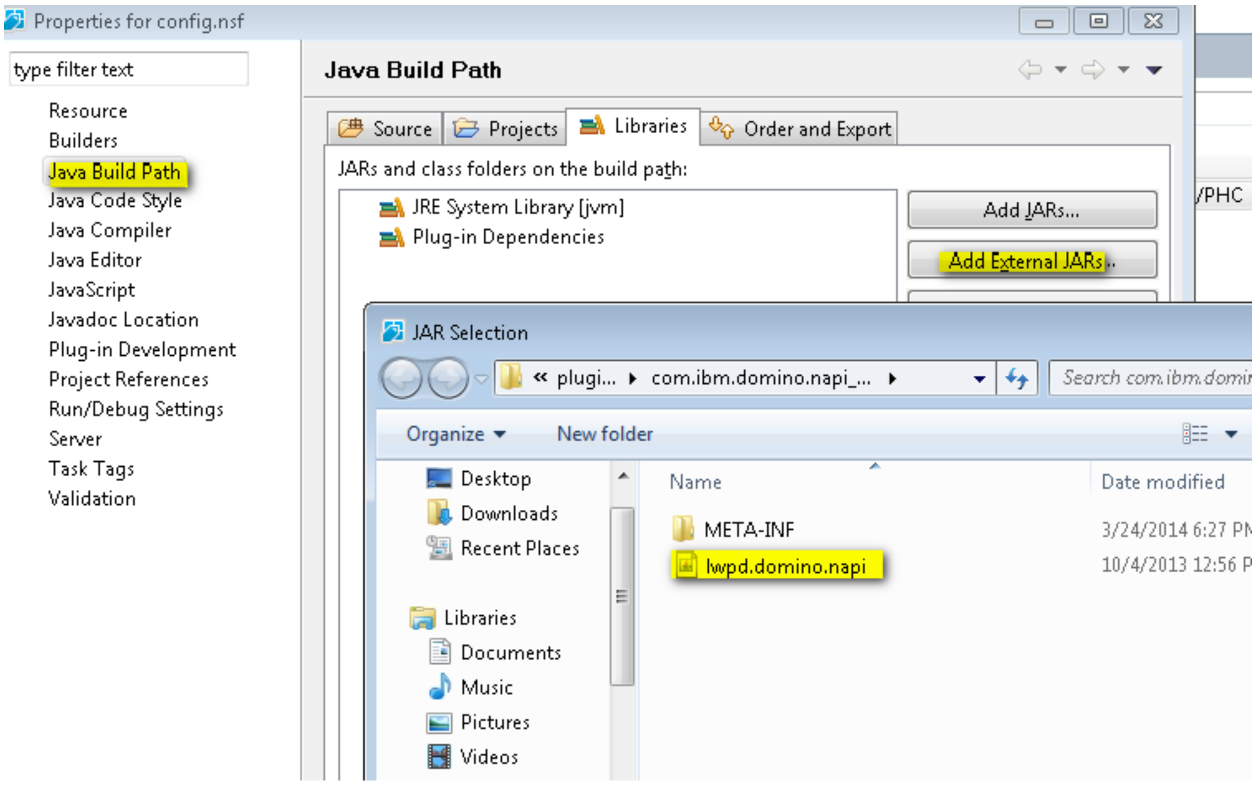
Folder name:

source

You should now see the new WEB-INF/source folder in the tree.

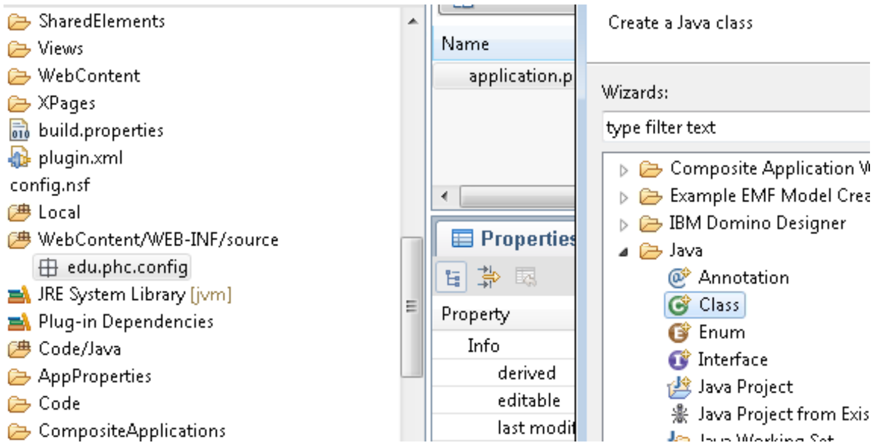

- config.nsf
  - Local
  - WebContent/WEB-INF/source
  - JRE System Library [jvm]
  - Plug-in Dependencies
  - Code/Java
  - AppProperties
  - Code
  - CompositeApplications
  - CustomControls
  - Data
  - Folders
  - Forms
  - Framesets
  - Pages
  - Resources

- We will now add the *lwpd.domino.napi.jar* file to the build path. Open up the properties dialog again - Java Build Path - Libraries and click on Add External Jars. Navigate to Domino\osgi\shared\eclipse\plugins\com.ibm.domino.napi_9.0.1.20131004-1200 and choose the .jar file.



- We're now ready to bring in Sven's Java class that utilizes the Java NAPI to modify the properties file (resource bundle).

  Right click on the WEB-INF/source folder you created and choose New - Other - Package. I named mine edu.phc.config. Right click on the package and choose New - Other - Class. Name it Toolbox. Click Finish.



It’s now time to copy and paste Sven’s Java class overwriting the Toolbox.java class you just created.

```
package edu.phc.config;

import java.io.ByteArrayOutputStream;

import java.io.InputStream;

import java.util.Properties;


import com.ibm.designer.domino.napi.NotesAPIException;

import com.ibm.designer.domino.napi.NotesDatabase;

import com.ibm.designer.domino.napi.NotesNote;

import com.ibm.designer.domino.napi.NotesSession;

import com.ibm.designer.domino.napi.design.FileAccess;

public class Toolbox {

          /**

    * loads the properties from a file

    *

    * @param dbPath full path of the database

    * @param fileName name of the file to load

    * @return the properties object

    */

   public Properties loadProperties(final String dbPath, final String fileName) {

      try {

         // load the file

         InputStream inStream = getFile( dbPath, fileName );


         // if file exists, init a properties object

         if (inStream != null) {

            Properties props = new Properties();

            props.load( inStream );

            return props;
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * saves a property file to a database
     *
     * @author Sven Hasselbach
     *
     * @param dbPath full path of the database
     * @param fileName name of the file to load
     * @param props the properties object
     */
    public void saveProperties(final String dbPath, final String fileName, final Properties
props) {
        try {
            // init Notes objects
            NotesSession nSession = new NotesSession();
            NotesDatabase nDB = nSession.getDatabaseByPath(dbPath);
            nDB.open();

            // store properties in byte array
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            props.store(bos, "My XSP Properties");

            // save the property file
            NotesNote nFile = FileAccess.getFileByPath(nDB, fileName);
        FileAccess.saveData(nFile, fileName, bos.toByteArray() );

        // recycle the objects
        nFile.recycle();
        nDB.recycle();
        nSession.recycle();

    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * loads a property file from a database
 *
 * @author Sven Hasselbach
 * @param dbPath full path of the database
 * @param fileName name of the file to load
 * @return InputStream content of the file
 */
private InputStream getFile(final String dbPath, final String fileName) {
    try {
        // init Notes objects
        NotesSession nSession = new NotesSession();
        NotesDatabase nDB = nSession.getDatabaseByPath(dbPath);
        nDB.open();

        // get the file
        NotesNote nNote = FileAccess.getFileByPath(nDB, fileName);
        InputStream inStream = FileAccess.readFileContentAsInputStream(nNote);
```

```
                // recycle the objects
                nNote.recycle();
                nDB.recycle();
                nSession.recycle();

                return inStream;
            } catch (NotesAPIException apiEx) {
                apiEx.printStackTrace();
            }
            return null;
        }
    }
```

- Let’s now create our custom controls. The first one is named cc_configuration. This will be embedded in the XPage that will be used by the application administrator to set the properties key-value pair.

Here’s the source code for it:

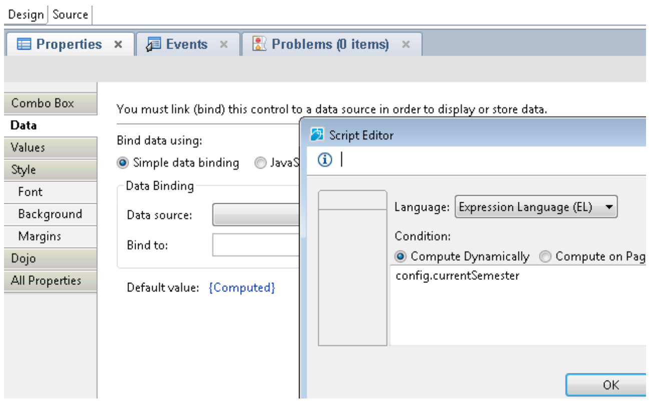
```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core">
	<xp:this.data>
		<xp:dominoDocument var="document1" formName="config">

		</xp:dominoDocument>
	</xp:this.data>

	<xp:this.resources>
		<xp:bundle src="/application.properties" var="config"></
xp:bundle>
	</xp:this.resources>
	<xp:table>
		<xp:tr>
			<xp:td>
				<xp:label value=" Set current semester:"
					id="ChoosecurrentSemester_Label1"
for="currentSemester1">
				</xp:label>
			</xp:td>
			<xp:td>

			<xp:comboBox id="Semester"
defaultValue="#{javascript:config.currentSemester}">
```

```
                        <xp:selectItem itemLabel="FA 2014"
                              itemValue="FA 2014">
                        </xp:selectItem>
                        <xp:selectItem itemLabel="SP 2015"
                              itemValue="SP 2015">
                        </xp:selectItem>
                        <xp:selectItem itemLabel="FA 2015"
                              itemValue="FA 2015">
                        </xp:selectItem>
                        <xp:selectItem itemLabel="SP 2016" itemValue="SP
2016"></xp:selectItem>
                        <xp:selectItem itemLabel="FA 2016" itemValue="FA
2016"></xp:selectItem>
                  </xp:comboBox></xp:td>
            </xp:tr>
            <xp:tr>
                  <xp:td></xp:td>
                  <xp:td></xp:td>
            </xp:tr>
            <xp:tr>
                  <xp:td>
                        <xp:button value="Save" id="button1">
                              <xp:eventHandler event="onclick" submit="true"
                                    refreshMode="complete" immediate="false"
save="true"
                                    id="eventHandler1">
                                    <xp:this.action><![CDATA[#{javascript:var
currSemester = getComponent("Semester").getValue();

importPackage(edu.phc.config);
var toolbox:edu.phc.config.Toolbox = new Toolbox();
var props:java.util.Properties = toolbox.loadProperties(
database.getFilePath(), "/application.properties" );
props.put( "currentSemester", currSemester );
toolbox.saveProperties( database.getFilePath(),"/application.properties",
props );}]]></xp:this.action>
                              </xp:eventHandler>
                        </xp:button>
                  </xp:td>
                  <xp:td>

                  </xp:td>
            </xp:tr>
      </xp:table>
      </xp:view>
```

- The code behind the Save button above is the second part of Sven’s solution that calls on the class we created earlier to set values in the properties file. Also note, that we have loaded the properties file as a resource in this custom control
- It’s now time to create the custom control that will be used in the application. We have a field on that custom control to pull the current semester from the properties file and use it as the default value. Here’s the source code for the custom control:

```
<?xml version="1.0" encoding="UTF-8"?>
<xp:view xmlns:xp="http://www.ibm.com/xsp/core">
      <xp:this.resources>
            <xp:bundle src="/application.properties" var="config"></
xp:bundle>
      </xp:this.resources>
      <xp:panel>
            <xp:table>
                  <xp:tr>
                        <xp:td>
                              Semester: &#160;
                              <xp:comboBox id="Semester"
                                    defaultValue="#{config.currentSemester}">
                                    <xp:selectItem itemLabel="FA 2014"
                                          itemValue="FA 2014"
id="selectItem12">
                                    </xp:selectItem>
                                    <xp:selectItem itemLabel="SP 2015"
                                          itemValue="SP 2015">
                                    </xp:selectItem>
                                    <xp:selectItem itemLabel="FA 2015"
                                          itemValue="FA 2015">
                                    </xp:selectItem>
                                    <xp:selectItem itemLabel="SP 2016"
                                          itemValue="SP 2016">
                                    </xp:selectItem>
                                    <xp:selectItem itemLabel="FA 2016"
                                          itemValue="FA 2016">
                                    </xp:selectItem>
                              </xp:comboBox>
                        </xp:td>
                  </xp:tr>
            </xp:table>
      </xp:panel>
</xp:view>
```

- Note how we use EL to pull the value from the application.properties file using the alias config. Also, once again, we have loaded the application.properties file as a resource for this custom control.



You must link (bind) this control to a data source in order to display or store data.

Bind data using: Simple data binding [✓] JavaS

Data Binding

Data source:

Bind to:

Default value: {Computed}

Language: Expression Language (EL)

Condition: Compute Dynamically [✓] Compute on Pag

config.currentSemester

- We’re almost done. We now need two XPages, one to hold each of the controls. Name the first XPage config.xsp (holds the cc_configuration control) and the second one, example.xsp (holds the cc_details custom control). Add a button at the top of the config.xsp page that lets you open the example.xsp XPage.

- To test, open the config.xsp page and set a semester value to something different than the initial value we gave the key in the application.properties file. To confirm that it has been set, open up the properties file and check.

  After you confirm it has been changed, click on the button to open the example.xsp XPage. The default value for semester should be the one you just set in the properties file from the config.xsp XPage.

And that’s it. HUGE thanks to Sven Hasselbach for the sample Java class and SSJS he provided and to Jakob Nielsen who got me started on resource bundles.

Sample database is posted along with the article.

Reading resources:

http://lpar.ath0.com/2013/10/07/how-to-store-application-level-configuration-information-in-xpages-applications/

http://stackoverflow.com/questions/23482973/creating-a-new-database-and-want-to-set-the-xsp-properties-in-it-via-js