Using the TinyMCE Javascript Content Editor in Web Page Composer

Applies to:

SAP NetWeaver Knowledge Management SPS14.

Summary

This paper explains how to replace the standard HTML editor that is used by Web Page Composer with the popular TinyMCE editor developed by Moxiecode Systems AB. Code samples and configuration instructions are provided.

Author: Boris Magocsi

Company: SAP Labs, LLC

Created on: 11 March 2008

Author Bio

Boris Magocsi works in the Regional Implementation Group (RIG) at SAP Labs, LLC in the United States. Boris is focused mainly on working with the NetWeaver Knowledge Management API as well as development in the NetWeaver Composition Environment.

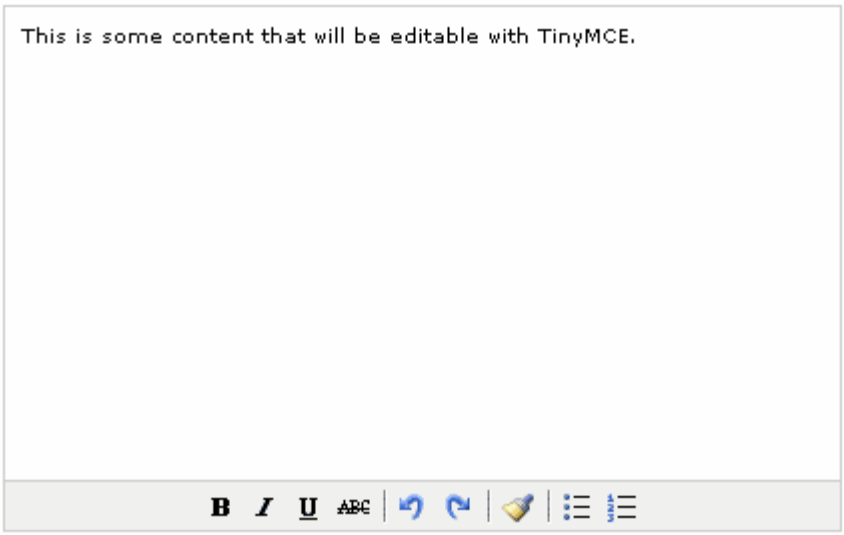
Table of Contents

Introduction

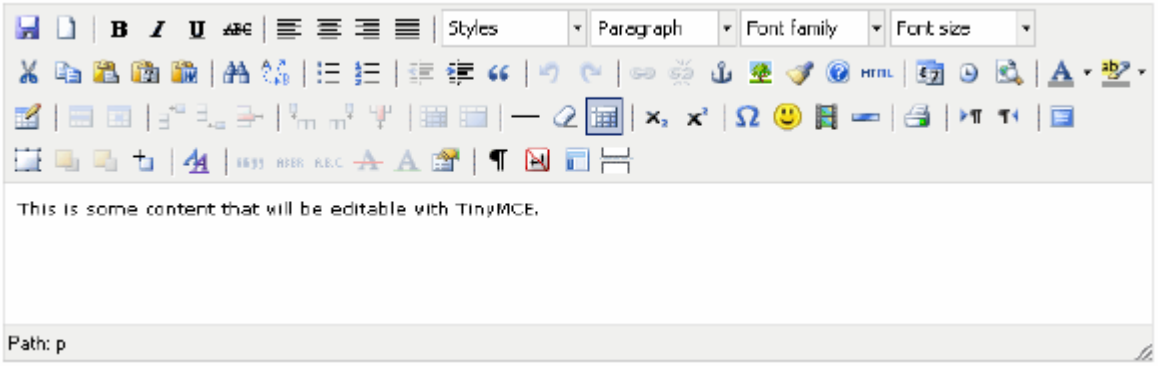
The TinyMCE HTML editor provides a feature-rich alternative to the standard editor that comes standard with Web Page Composer (WPC) in NetWeaver Portal. As of SP14, it is possible to replace the standard editor by using the WPC Editor API. This paper explains the basic steps in making that replacement, including the coding that is required, as well as the configuration changes needed to integrate the new component.

The TinyMCE Javascript Editing Component by Moxicode Systems AB

The TinyMCE Wiki contains a number of examples that show the user interface of the editor. The editor has many configuration options and can be used in a number of modes, ranging from the simple:



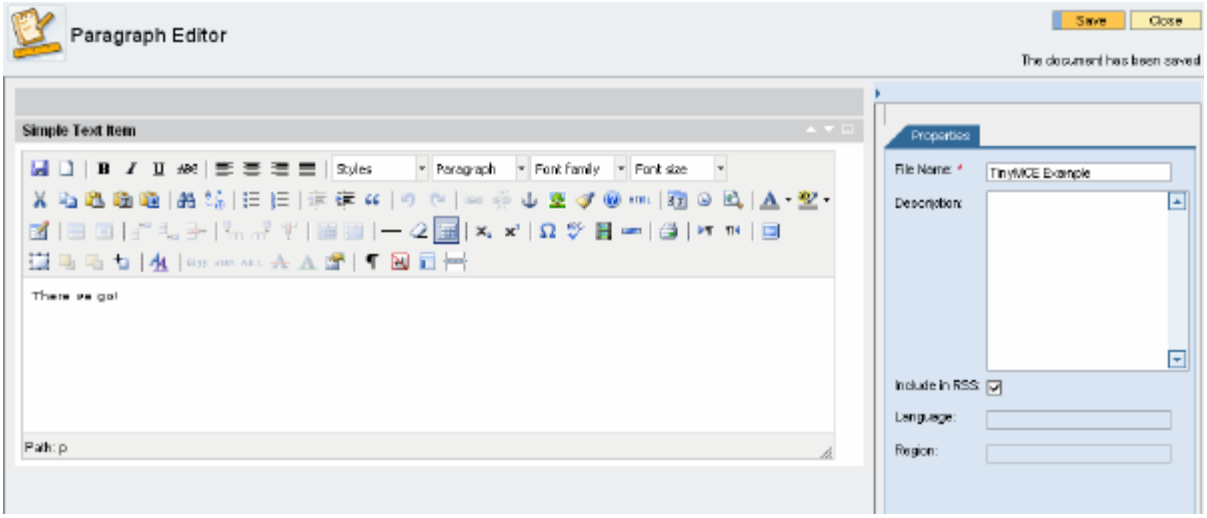
To the more complex:



Note: Not all of the features supported in TinyMCE will automatically be available to you when using it as an editor for the WPC. This paper shows the basics of integrating the editor into the WPC, enabling all of the features (such as images) is left as an exercise for the reader.

Integration into WPC

The goal of our basic integration example is to replace the standard HTML editor that is used for editing paragraphs. When we are done with coding and configuration, creating a new paragraph or editing an existing paragraph should result in the new editor being displayed:



The WPC Editor API

This section outlines the core ideas behind the editor API and gives an introduction to the basic functions of core classes and methods. This is meant to provide guidance in understanding the WPC Editor API and is not meant to be complete documentation.

IEditorComponent

Initialization

The IEditorComponent object is initialized by the framework through a call to the `init` method.

Persistence

The core interface of the WPC Editor API is IEditorComponent. Implementing this interface provides the WPC runtime with an entry point into your custom code. The implementation of this interface is responsible from converting from and to XML that can be stored in KM. The three methods that are core to this task are `initialiseFromXML`, `initializeFromPageContext` and `getXMLElement`.

The method `initialiseFromXML` is called when a new instance of the editor has been created and the content of the editor needs to be initialized from previously persisted information.

The method `initializeFromPageContext` is called when the content that is being edited already exists within the page context, i.e. the save button was pressed to persist the content and the screen is refreshed to allow further editing. This method is also called when new content (e.g. a new paragraph) is created.

Note: In the sample code, this method registers the classes delivered in the PAR file with the runtime classloader. This is to allow the use of the method call `context.getComponentForId()` which resolves the editor's class based on its id.

The third method, `getXMLElement`, is called to retrieve the content from the editor component in order to persist the content in KM. This occurs when the save button is pressed on the edit screen.

Rendering

The class that implements IEditorComponent must return an HTMLB component using the method `getHtmlbComponent` to allow the portal runtime to render the custom editor.

The returned component can render the editor itself by implementing the `renderAsString` method, or it can hand rendering off to specialized `HtmlbRenderer` objects by not overwriting the `renderAsString` method. If the latter is done, be sure to register the `HtmlbRenderer` implementation class with the `RenderManager`:

```
RendererManager.addRenderer(
                TinyMCEComponent.UI_ID,
                BrowserType.DEFAULT,
                TinyMCERenderer.class);
```

This is best done in the `afterInit` method of the portal service that is contained in the PAR (see code example). The benefit of using this approach is that it allows the use of different renderers depending on the client platform (Netscape-type browsers, Internet Explorer, OS platform, etc).

Specifics of using TinyMCE

In the example code, the TinyMCE editor is tied into the scenario within the TinyMCERenderer class, which is a `HtmlbRenderer` object. The Javascript files required for TinyMCE to function are contained within the deployed PAR and are referenced in the generated page by adding a head resource:

```
renderContext.getDocument().addHeadResource(
                ResourceType.JAVASCRIPT,
                "text/javascript",
                jspath);
```

The TinyMCE editor has a mode in which it takes over any HTML TEXTAREA objects on the web page. Since there are more than one TEXTAREA objects on the WPC editing page, the editor_selector option is used to point the editor framework to a specific textarea containing only the content to be edited. If this selector were not used, then other TEXTAREAs would also turn into TinyMCE-edited objects when rendered on the page.

```
edit.setInitParameter("mode", "textareas");
edit.setInitParameter("theme", "advanced");
edit.setInitParameter("editor_selector", "mceEditor");
```

To demonstrate the use of the TinyMCE framework, a number of specific options have been set using the `setInitParameter` method.

Finally, in order to initialize the TinyMCE editing object, a call needs to be made that will result in the generation of initialization Javascript:

```
renderContext.getDocument().setHeadRawText( edit.getInitString().toString() );
```

Bootstrap and portalapp.xml

The sample code also contains an interface/class pair that extends and implements the portal PRT IService interface. The `Bootstrap` class is referenced in the portalapp.xml file:

```
<service name="Bootstrap">
      <service-config>
        <property name="className"
                  value="com.sap.nwrig.example.wpc_editor_api.Bootstrap"/>
        <property name="startup" value="true"/>
      </service-config>
    </service>
```

The startup property causes the PRT to call the service's `init` method upon portal startup or deployment. Within the `init` method, the PAR classloader is registered with the runtime, which makes the runtime aware of the new deployable unit and its contents.

The `afterInit` method of the Bootstrap class is used to register the specialized `HtmlbRenderer` with the `RendererManager`.

Summary of Classes and Interfaces in Example Code

`EditorComponent`: This class implements the `IEditorComponent` interface of the WPC editor API. It therefore handles the steps required to persist and load the edited content from KM (it does not persist or load from KM itself, the framework handles this). It also handles instantiation of the `TinyMCEComponent` editor object.

`TinyMCEComponent`: This class extends the HTMLB Component class and represents the editor object. In the example code, it does not handle rendering via the `renderAsString` method (not overridden).

`TinyMCERenderer`: This class extends the HTMLB `HtmlbRenderer` class and handles all rendering of the editor object for all platforms. In order to handle multiple platforms differently, separate classes analogous to this class should be implemented and registered using the `RendererManager`.

`IBootstrap`: This interface implements the PRT `IService` interface and defines the key that uniquely identifies the PAR deployable unit on all environments on which the PAR is to be deployed. In other words, make sure that no other PAR uses the same key or you will end up overwriting deployable units with one another.

`Bootstrap`: This is the service implementation that is referenced in the portalapp.xml file. The init method is called on portal startup and on deploy. Pending a successful `init` call, the `afterInit` method is called, where the `TinyMCERenderer` is registered with the `RendererManager`.

Configuration Information

The new editor is integrated into the WPC environment by performing configuration steps. The first step is automatically completed during deployment provided the src.config tree within the development project contains the relevant information. Beyond that, an XML file needs to be updated and a call needs to be made to trigger reloading of the editor configuration.

Step 1: Register New Editor Component

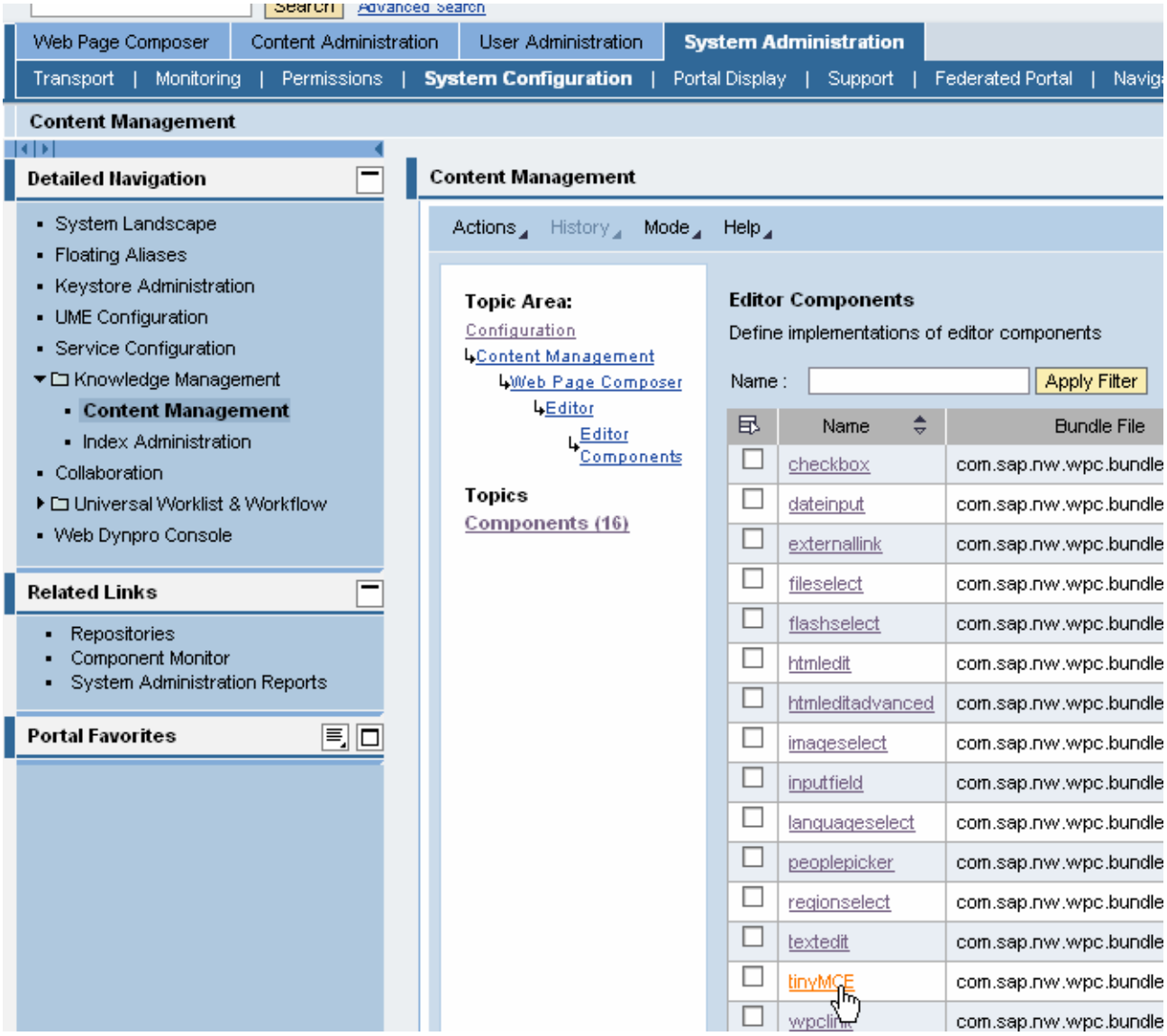
The sample project contains an XML file located under

```
src.config\install\data\cm\wpc\editor\components\tinyMCE.co.xml
```

This XML configurable object definition registers the newly developed editor component with the WPC framework. Examining the contents of this file reveals that it contains a name/class mapping that registers the `IEditorComponent` implementation class under the name “tinyMCE”.

```
<Configurable configclass="component" owner="com.sap.nw.wpc.config">
   <property name="bundlefile" value="com.sap.nw.wpc.bundles.Commands"/>
   <property name="implclass"
         value="com.sap.nwrig.example.wpc_editor_api.element.EditorComponent"/>
   <property name="description" value="tinyMCE edit component"/>
   <property name="name" value="tinyMCE"/>
</Configurable>
```

The same configuration can be carried out in the portal UI using the KM Configuration iView. After deployment, the new editor component configuration should be visible there:



Step 2: Update Paragraph Text / Editor Mapping

The topic of this paper is replacing the standard HTML editor with the TinyMCE editor, and in order to do this the developer needs to edit the existing XM rather than creating new definitions.

The new editor component has already been registered with the system in the first step. Now the framework needs to be told that text elements are to be edited using the new tinyMCE component, rather than the htmledit component.

Note: It's also possible to switch to a more advanced editor delivered by SAP by using the htmleditadvanced component in place of the htmledit component.

In order to carry out the mapping, the file /etc/wpceditor/types/wpc_simple_item.xml needs to be edited from the following:

```
<elements>
      <element id="text" description="xml.xlbl.text" type="htmledit"
            default="true" singleinstance="true" nodelete="true"/>
</elements>
```

And changed to:

```
<elements>
      <element id="text" description="xml.xlbl.text" type="tinyMCE"
            default="true" singleinstance="true" nodelete="true"/>
</elements>
```

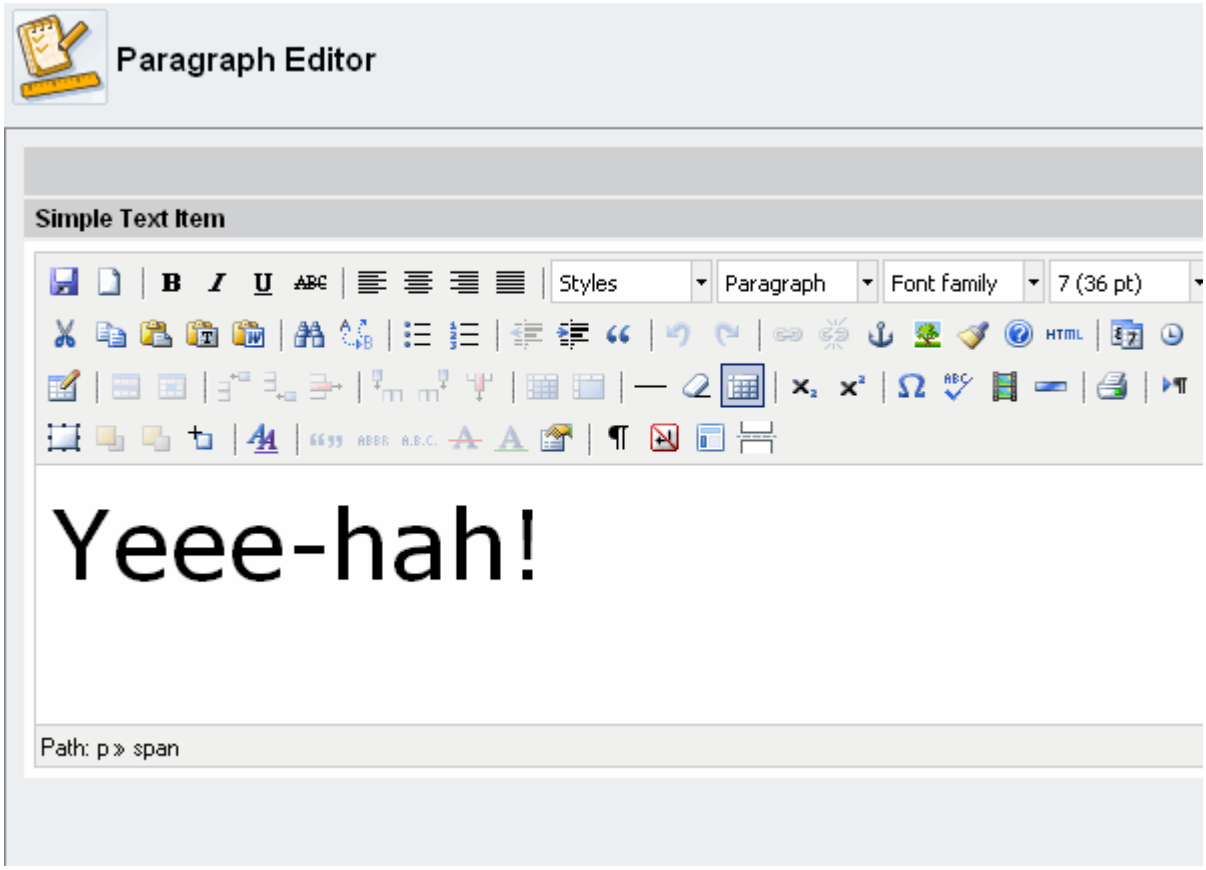
Changing this single attribute from "htmledit" to "tinyMCE" changes the editor used for editng objects of type "text" (given by the "id" attribute).

Step 3: Trigger Reload of Configuration

In order to trigger a re-read and reload of the editor configuration, load the following page in your browser:

/irj/servlet/prt/portal/prtroot/com.sap.nw.wpc.designtime.ReloadEditorConfigComponent

Once these three steps have been carried out, create a new paragraph in WPC and you should see the TinyMCE editor in place of the standard HTML editor:



Appendix

A Note about Libraries

You will need to gather the WPC and KM libraries from your server installation, since they are not included in NetWeaver Developer Studio (or not always up to date depending on your patch level).

The easiest way to do this is to search for a set of files on the server and copy them to your local PC. Then, you can set up a classpath variable to point to that location.

To get the libraries, go to the path

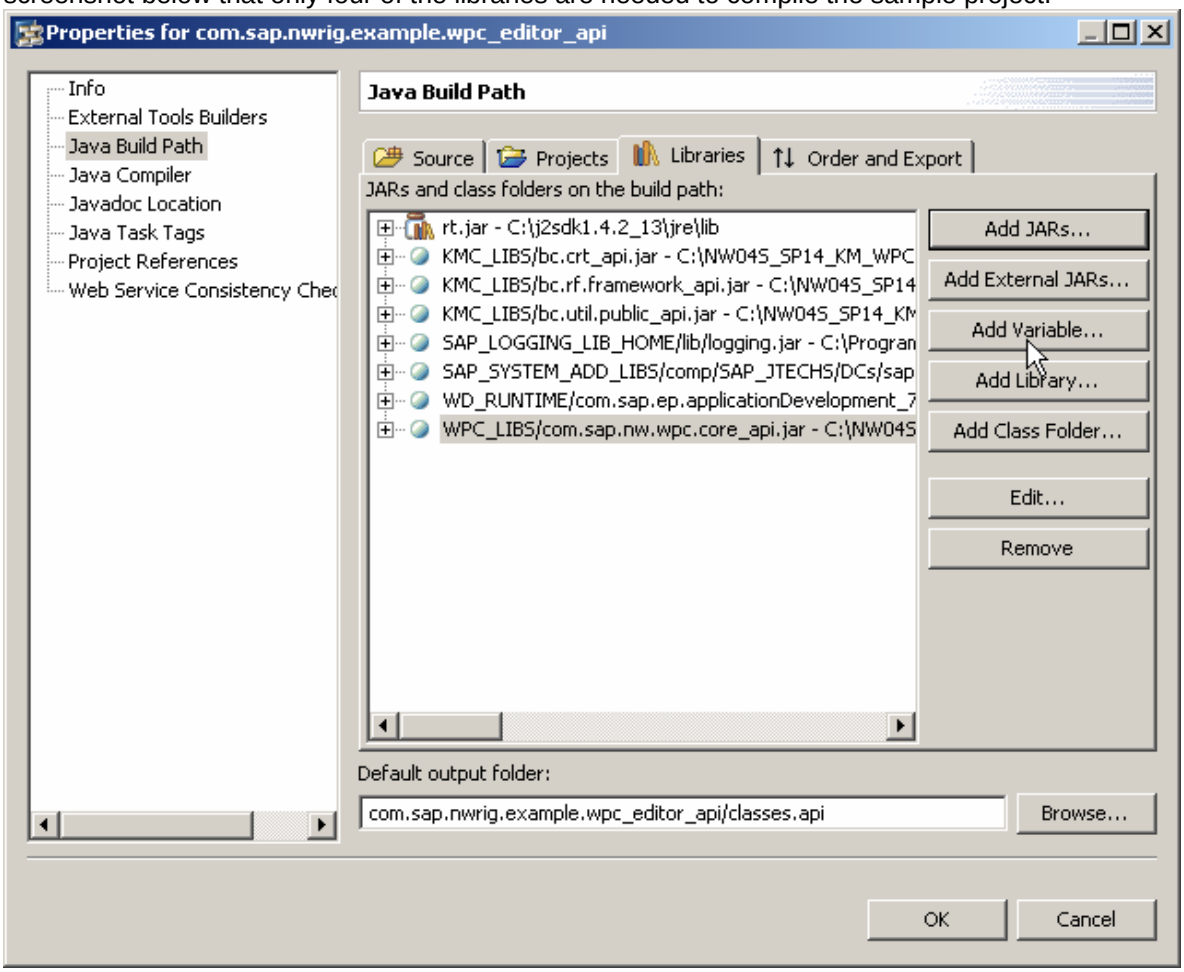
\usr\sap\<SID>\JCXX\j2ee\cluster\server0\apps\sap.com\irj\servlet_jsp\irj\root\WEB-INF\portal

on your server and search for the following patterns there:

km*_api.jar
bc*_api.jar
coll*_api.jar
wpc*_api.jar

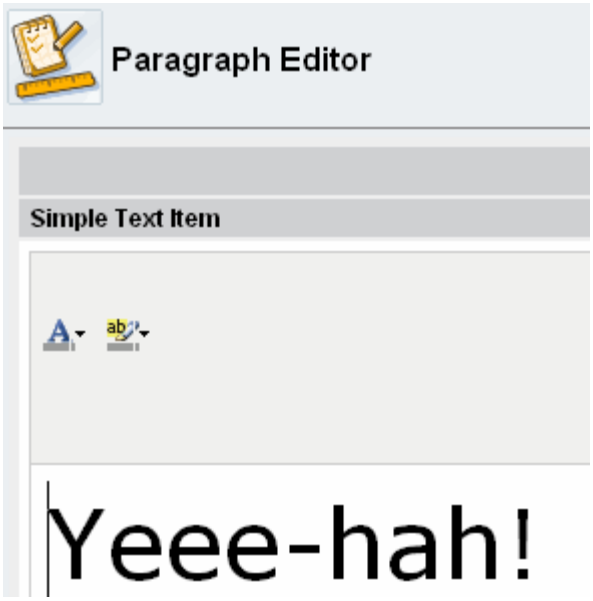
If you look in the .classpath file of the sample project, you will see that you really only need a few of these to make this program work. However, you may be modifying the code to include more functionality or might want to do more WPC / KM development in the future, so copying all the files that match those patterns is not such a bad idea.

Next, place the files in a directory on your PC. Then, set up two classpath variables in NetWeaver Developer Studio that points to that directory. Do this by selecting “Add Variable” in the project properties pane, then follow the steps to set up “KMC_LIBS” and “WPC_LIBS” to point to your new directory. You can see from the screenshot below that only four of the libraries are needed to compile the sample project.



TinyMCE UI Initialization

In some cases the TinyMCE UI does not seem to initialize properly when the edit dialog is first opened. This results in a nearly blank button pane.



Choosing the font color drop-down and selecting “More colors” will generally trigger the rest of the UI to load. If you can figure out why this is happening, please let the author of this document know.

Getting the Sample Code

The sample code that shows TinyMCE integration into the Web Page Composer is available from SDN for download. Note that it would be a bad idea to use this in a production environment, since it’s sample code provided as-is, without any warranty of any kind.

Related Content

TinyMCE Editor

Copyright