Combining Different Types of Plugins

There are several types of plugins, each with their own distinct features. The "B" in WCB (plugin) stands for "Bundle", which means that the different features can be bundled together in one plugin. This page explains how to combine different types.

Make sure your plugins are created from an archetype. To learn how to do this, take a look at the plugin development - Quickstart topic. By starting your plugins based on archetypes, you always have the most clean example code available. Merging plugins basically means merging the "Activator.java" files and the directory structures of all your plugins.

The steps that need to taken to complete the merge are described below.

1. Examine your current situation: what's the current status of the plugins you're about to merge? If you're still in development for example, there's no real content for this plugin, the strategy below could be followed. When you do have real content, another strategy is needed because content that is written to the JCR by a plugin is linked to its plugin ID. Changing the plugin ID this plugin would lose all it's JCR content. If the plugin is already in use, go to step **2b. Existing plugin**.

2a. New plugin

Create a "generic" plugin with a common package and bundle name, so that the functionality you're about to merge fits this name. For example: when you have a BookReviewMediaItem and a BookReviewElement, the common package name "Books" could be plausible. Keep in mind that a new plugin id is created now.

Go to step **3**.

2b. Existing plugin

When you have an existing plugin (with real content), where functionality needs to be added to, you have another situation. Use the original root package /domain, to ensure backwards compatibility.

3. Now add the source files of the plugin's step by step, starting with the following:

- Component specific folders from Java packages in /src/main/java (exclude directories/files described in step 4.)
- JSPFs in editpresentation & JSPFs plus XMLs (descriptor files) in showpresentation / presentationtype (check for identical names!)
- Static files in /static/backend

4. Now the "difficult" part needs to be done: merging files. The following files need to be merged:

- Activator.java -> read below **1. Activator.java**
- Messages in /src/main/resources/messages
- WCBConstants.java in /src/main/java/.../api
- Online Help files in /src/main/resources/help.*
- POM files (dependencies have to be added). Remember to re-run the Maven eclipse:eclipse command to let Eclipse know something changed.

5. In Eclipse you will still notice some errors, because certain classes cannot be found anymore. Therefore packages have to be renamed, so that they will fit in the new package.
Example: the new "generic" package is named: com.gxwebmanager.solutions.books
The "old" plugin package name was: com.gxwebmanager.solutions.bookreviewmediaitem

The latter has to refactored through Eclipse, to the following:
com.gxwebmanager.solutions.books.bookreviewmediaitem

6. The last thing you need to do, is the use the "organize import" functionality in Eclipse, to update references. O, and don't forget to manually update the implementation class, Eclipse might not fix that for you.

Example: `@Interfaces("com.gxwebmanager.solutions.bookreviewmediaitem.api.ReviewMediaItemMediaItemVersion")`

Has to be changed to:

```
@Interfaces("com.gxwebmanager.solutions.books.api.ReviewMediaItemMediaItemVersion")
```

1. Activator.java
First: all your plugins have a file named "Activator.java". This file is responsible for setting up the plugin and allows WebManager to 'know' what is inside the plugin and how to interact with it. Choose one plugin that you want to merge the others into and examine its "Activator.java" file. In particular, examine the "getBundleDefinition()" method as it sums up the component definitions.

```
34 /**
35   * Creates and returns the bundle definition of the WebManager component bundle
36   * @return the bundle definition of the WebManager component bundle
37   */
38 @Override
39 protected ComponentBundleDefinition getBundleDefinition() {
40   ComponentBundleDefinitionImpl componentBundleDefinition = new ComponentBundleDefinitionImpl();
41   componentBundleDefinition.setId(BUNDLE_DEFINITION_ID);
42   componentBundleDefinition.setName("Presentation " + plugin_ID);
43   componentBundleDefinition.setDescription("plugin containing presentation " + plugin_ID);
44   ComponentDefinition[] componentDefinitions = new ComponentDefinition[]
{getPresentationComponentDefinition()};
45   componentBundleDefinition.setComponentDefinitions(componentDefinitions);
46   return componentBundleDefinition;
47 }
48 /**
49   * Returns the component definition for the presentation component
50   * @return The component definition for the presentation component
51   */
52 protected ComponentDefinition getPresentationComponentDefinition() {
53   PresentationComponentDefinitionImpl definition = new PresentationComponentDefinitionImpl(false);
54   definition.setId(COMPONENT_DEFINITION_ID);
55   definition.setName(plugin_ID + " presentation");
56   definition.setDescription("Implements the " + plugin_ID + " presentation");
57   definition.setTypeId(PresentationComponentType.class.getName());
58   definition.setProperties(new Hashtable<String, String>());
59   definition.setImplementationClassName(SimplePresentationComponent.class.getName());
60   definition.setInterfaceClassNames(new String[]{PresentationComponent.class.getName()});
61   definition.setDependencies(new ComponentDependency[]{});
62   definition.setWrapperClassNames(new String[]{});
63   return definition;
64 }
```

On line 44, all component definitions are summed up in the componentDefinitions array. If you want to bundle more plugins, you have to expand that array.

Line 48 - 64 contain the component definition of one plugin, in this case a presentation plugin. Copy the definition methods from the other "Activator.java" files that you want to merge after this method "getPresentationComponentDefinition()".

Next, expand the "componentDefinitions" array on line 44 by using all your definition methods, e.g. like below:

```
44 ComponentDefinition[] componentDefinitions = new ComponentDefinition[] {
45   getPresentationComponentDefinition(),
46   getElementComponentDefinition(),
47   getServiceComponentDefinition()
48 };
```