Naming and Metadata

So you want to add a new package, plugin or feature to KIELER, eh? Well, then there's some stuff you should know. First, there's some general remarks applying to all kinds of stuff. Then, there's more details specific to certain kinds of stuff.

Content

General Advice

These remarks apply to all, packages, namespaces, plug-in projects, and features.

Naming Conventions

Follow the Eclipse naming conventions; basically, projects are named after their base packages, who in turn should follow Oracle's naming conventions.

Packages and Namespaces

Naming Conventions

Packages and namespaces may use the university internal domain: http://kieler.cs.cau.de. In fact, that domain forwards to the KIELER project website, although it is not the official domain. Package names should begin with de.cau.cs.kieler, following Oracle's naming conventions.

Plug-In Projects

Naming Conventions

When a project consists of several plug-ins, all of them should share the same prefix. For example, all KIML plug-ins start with de.cau.cs.kieler.kiml. Plug-ins containing examples should begin with de.cau.cs.kieler.example. The project name, project ID and project folder should be the same.

Project Settings

- *Java Build Path*
  - On the *Libraries* tab, remove the default JRE System Library and click *Add Library*. Choose *JRE System Library* and click *Next*. Select *Execution environment*, ensure that Workspace defualt JRE is *jdk1.8.0*.
- *Checkstyle*
  - If your plug-in only contains generated code, disable Checkstyle.
  - Otherwise, activate Checkstyle.
    - In the *Exclude from checking...* list, select *all file types except*, click *Change...* and remove *properties*. Checkstyle usually gives wrong warnings for property files.
    - If there are packages containing generated code, select *files from packages* from the *Exclude from checking...* list and add the packages to it.

Version Numbering

See the Eclipse guidelines on version numbering for general advice on choosing version numbers for projects. All our projects start with version 0.1.0, which is increased after each new release where the project content or its dependencies have changes. Append the string .qualifierto all version numbers to avoid problems with the build system. (if that part is omitted, the build system won't realize that it has to build the project)

Metadata

In the plug-in manifest editor, be sure to follow these settings:

- Tab *Overview*
  - ID: start with *de.cau.cs.kieler* as prefix, except for plugins that contain third-party code
  - Version: *0.1.0.qualifier*, increase with each release according to changes in the plugin
  - Name: start with *KIELER*, e.g. *KIELER Core UI*
  - Provider: *Kiel University*
  - Check *Activate this plug-in when one of its classes is loaded* and *This plug-in is a singleton*, except if you know what you're doing
  - Execution Environment: *JavaSE-1.8.0*
  - Activator: Set the name of your Activator class (this represents the entry point of you plug-in). By default it is named "Activator.java". **Rename this class to <Name>Plugin.java** (e.g. `CorePlugin.java`).
- Tab *Dependencies*
  - For Eclipse plugins: set your current version with 0 as third digit for *Minimum Version*, e.g. *org.eclipse.core.runtime (3.5.0)*
  - For our own plugins: if you know that you really need a specific version of the plugin, do the same as for Eclipse plugins, else leave the minimum version empty
  - Don't reexport dependencies, except in one case: UI plug-ins may reexport their dependencies on plug-ins that are part of the same project. (de.cau.cs.kieler.kiml.ui could reexport a dependency on de.cau.cs.kieler.kiml)
  - Collect all KIELER internal dependencies on the bottom of the list to increase readability.
- Tab *Runtime*
  - Add all Java packages that must be visible for other plugins, e.g. if they contain classes or interfaces that must be referenced elsewhere
- Tab *Extensions*
  - Add extensions to other extension points as needed
- Tab *Build*
  - Binary Build
    - Include epl-v10.html (copy into plugin from one of the other plugins)
    - Include META-INF folder, plugin.xml and plugin.properties if present
    - Do **not** include src or bin folder
    - Include all icons, models etc. that are in your plugin
  - Source Build
    - Include epl-v10.html (copy into plugin from one of the other plugins)
    - Do **not** include src or bin folder
    - Include files that you explicitly want to be present in the source project

Metadata for Extension Points

- plugin.xml (*Extension Points*tab)
  - ID and name of schema file shall be equal, starting with a lower case character, e.g. diagramConnectors and schema /diagramConnectors.exsd
  - Name shall be equal to ID, but starting with an upper case character, e.g. DiagramConnectors
  - Do not prefix the ID with the plugin ID
- Extension point schema (*.exsdfiles)
  - *Overview*tab
    - Point ID and Name: same as in plugin.xml
    - Description: write general information on the extension point
    - Since: enter the version number of the host plugin, without .qualifier
    - Examples: write example XML code for plugin.xml that uses the extension point
    - API Information: write useful information for developers that want to use the extension point
    - Copyright:

```
Copyright 2010 by<br>
 + Christian-Albrechts-University of Kiel<br>
   + Department of Computer Science<br>
     + Real-Time and Embedded Systems Group<br>
This program and the accompanying materials are made available under the terms of the
Eclipse Public License (EPL) which accompanies this distribution, and is available at
<a href="http://www.eclipse.org/legal/epl-v10.html">http://www.eclipse.org/legal/epl-v10.
html</a>.
```

  - *Definition*tab:
    - Create new elements that can be added to the extension point and give them attributes (do not change the extension element)
    - Create a *Sequence* in the extension element and drag the top-level elements into it, set their *Max Occurrences* to *Unbounded*

Automatic Build Integration

- New plugin
  - New plugins need a `pom.xml` in the plugin's root folder

plugin pom.xml

```
 <?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>de.cau.cs.kieler</groupId>
    <artifactId>plugins</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <groupId>de.cau.cs.kieler</groupId>
  <artifactId>YOUR.NEW.PLUGIN.NAME</artifactId>
  <version>0.7.0-SNAPSHOT</version>
  <packaging>eclipse-plugin</packaging>
</project>
```

- Make sure that `<version>` is in sync with `Bundle-Version` in `META-INF/MANIFEST.MF`
- Add a `<module>` entry in the corresponding parent POM i.e. `plugins/pom.xml`
- If your plugin uses xtend, tell maven to run the xtend compiler in `pom.xml`

```
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <groupId>org.eclipse.xtend2</groupId>
        <artifactId>xtend-maven-plugin</artifactId>
        <executions>
          <execution>
            <goals>
              <goal>compile</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
```

- New JUnit test plugin
  - Basically the same steps as for plugins except `<packaging>` is `eclipse-test-plugin`
- New features
  - The same steps as for plugins except `<packaging>` is `eclipse-feature`
  - Additionally if sources are to be distributed a separate source feature with the corresponding source-bundles must be created

Features

Naming Conventions

If <name> is the name of a feature's base plug-in, then the feature project and its directory are named <name>.feature, and the feature is assigned the ID <name>.feature. For example, the plug-in de.cau.cs.kieler.core is contained in the feature de.cau.cs.kieler.core.feature, and this feature is located in the feature project de.cau.cs.kieler.core.feature in the directory features/de.cau.cs.kieler.core.feature.

Version Numbering

The version numbering conventions for plug-ins apply to features as well.

Metadata

- Project name: same as project folder, use the feature ID with .feature as suffix

- feature.xml (*Overview* tab)
  - ID: start with de.cau.cs.kieler as prefix, use .feature as suffix
  - Version: 0.1.0.qualifier, increase with each release according to changes in one of the contained plugins
  - Name: start with *KIELER*, e.g. *KIELER Core*
  - Provider: *Christian-Albrechts-Universität zu Kiel*
  - Update Site URL: http://rtsys.informatik.uni-kiel.de/~kieler/updatesite/nightly/
  - Update Site Name: *KIELER Nightly Builds*
- feature.xml (*Information* tab)
  - Feature Description: enter URL of the Wiki page of your project and write a general description as text
  - Copyright Notice:
    ```
    Copyright 2010 by Real-Time and Embedded Systems Group, Department
    of Computer Science, Christian-Albrechts-University of Kiel
    ```
  - License Agreement: enter http://www.eclipse.org/legal/epl-v10.html as URL and copy-paste the complete content of that page as text
  - Sites to Visit: add update sites of features that are not part of the official Eclipse distribution and are needed by the plugins contained in your feature
- feature.xml (*Dependencies* tab)
  - Add other KIELER features that contain plugins that are needed by your plugins, e.g. de.cau.cs.kieler.core.feature feature
  - Add single plugins if you do not want to depend on a whole feature, use *Compute* to automatically compute these plugins and remove duplicate entries
- build.properties (*Build*tab)
  - Include epl-v10.html in binary and source build (copy into plugin from one of the other plugins)
  - Include feature.xml in binary build
  - Manually add the following line to build.properties, replace <feature>by the suffix of your feature ID
    ```
    generate.feature@de.cau.cs.kieler.<feature>.source = de.cau.cs.kieler.<feature>
    ```

Source Features

This section hasn't been written yet.