eHealth Framework

How to set up a new eHF module

Imprint

InterComponentWare
Altrottstraße 31
69190 Walldorf
Tel.: +49 (0) 6227 385 0
Fax.: +49 (0) 6227 385 199

Document version: preliminary
Document Language: en (US)
Product Name: eHealth Framework
Product Version: 2.10.3
Last Change: 23.03.2010
Editorial Staff: BAS Technology

Notice

Note:

The current version of this document has a draft status and various chapters are still in review.

The document is collaboratively built with the use of the Darwin-Information-Typing-Architecture (DITA) and has therefore a draft status concerning styles and layout. The necessary adaptations are currently also in a developmental stage.

Contents

1 Overview

The ICW eHealthFramework and the modelling tool Topcased need to be installed. The eHF is composed of several modules (user management, authorization, code system, commons, etc). Almost all eHF base modules and custom modules have in common, that they are based on the same generic architecture. A model driven generator generates a new custom eHF module with all the appropriate aspects of this architecture. Before starting the development of the new application module, the project structure for this application module needs to be setup. To do this, the Maven genapp plugin is used, in conjunction with a custom designed eHF module template.

2 Tasks

2.1 Generate a basic module project

1. Choose a module project type.
   There are several module templates available that can be used to create different module types:
   - ehf-module-template : Used to create an eHF application module.
   - ehf-repository-module-template : Used to create an eHF module contain a Codesystem repository.

   The ehf module template is used in conjunction with the Maven genapp command to create a new module.

2. Create a genapp maven project.
   With your favorite text editor of choice create the following `project.xml` file, saving it directly in your Eclipse <<*workspace*>> folder.

```
<?xml version="1.0"?>
<project>
    <pomVersion>3</pomVersion>
    <currentVersion>SNAPSHOT</currentVersion>
    <dependencies>
        <dependency>
            <groupId>ehf</groupId>
            <artifactId>ehf-genapp-maven-plugin</artifactId>
            <version>X.X.X</version>
            <type>plugin</type>
        </dependency>
    </dependencies>
</project>
```

3. In the `project.xml` just created, update the correct versions in the genapp maven project.
   In the version tag, replace *X.X.X* with the version of the eHF that you are currently working with i.e. 2.9.5.

4. Create the module template using the Maven genapp command and the ehf-project-templates contained in the `ehf-genapp-maven-plugin`.
   Open a console window, and navigate to your Eclipse <<*workspace*>> folder. Run the `maven genapp` plug-in in conjunction with the ehf module template desired e.g.

```
maven genapp -Dmaven.genapp.template="ehf-module-template"
```

   The *ehf-module-template* template can be replaced with other ehf module templates if desired.

5. Answer the various prompts of the Maven genapp plugin.
   - *project root directory* is the root directory where the new module will be located. This can either be a relative or absolute path. Maven will create this directory if it doesn't exist.
   - *application ID* is used by Maven to define the artifactId tag in the *project.xml* file of the module. This is the name that will be used by Maven when building the project artifacts (i.e. *.jar* files etc.).
   - *application group ID* is used by Maven to define the groupId tag in the *project.xml* file of the module. All artifacts with the same group id are stored in the same top level folder within the maven repository.

- *application name* is a short descriptive name. It is used by Maven to define the name tag in the *project.xml* file of the module.
- *application package* is the top level java package that will be used throughout the code.
- *module name* is the name of the module to be used throughout the various configuration files when referring to the module, and also for the naming of fragment files that will be integrated into parent files (e.g. Spring configuration files or the schema name to be used in the database). The name of the module should not contain any prefixes, and if the module is to ultimately be part of the eHF, it should not contain the ehf- prefix.
- *ehf-version* is the version of the ehf. It will be used for the ehf dependencies of the project.

Once completed, Maven has generated a basic eHF module project in the Eclipse workspace.

2.2 Import the project in Eclipse

1. Select the menu *File* | *New* | *Project...* from within Eclipse.
   The *New Project* window appears.
2. Expand the *General* folder, select *Project* and click *Next* .
   The *Wizard* window appears.
3. Enter the project name and click *Finish* .

The project will now be listed in the package explorer of Eclipse.

2.3 Update the project dependencies

1. Open the *project.xml* file in the root folder of the project from within Eclipse.
2. Click on the file within the editor window and hit *Ctrl-F* to open the *Find/Replace* dialog window in Eclipse.
   Make sure the source tab is selected in the lower left hand corner of the editor.
3. Enter *"<version>SNAPSHOT"* in the *Find* textbox, and *"<version>x.x.x"* (where *x.x.x* is a placeholder for the locally used version of the eHF)in the *Replace With* textbox.
   By default the dependencies for eHF modules, defined in the *project.xml* file is *SNAPSHOT*, which is the current development version of the eHF.
4. Click *Replace All* to correct all the eHF dependencies to have the correct version.
   There will be one *"SNAPSHOT"* left in the *project.xml* file between the *<currentVersion>* tag. This tag defines the version of the new module, and as it is still in development it is okay to name it *SNAPSHOT*.

Having now changed all the versions of the eHF dependencies in the module, Maven will use the correct version while building the module.

2.4 Generate the project folder structure

1. Open a console window and navigate to the project folder, which was created by the Maven genapp plugin.
2. Run the Eclipse plugin by entering `maven eclipse:eclipse` .
   The console will then show the Maven output.

3. Return to Eclipse. Select the project in Eclipse's Package Explorer and press *F5* to refresh the project.

Now Maven will generate the project folder structure and the *.classpath* and *.project* files in the Eclipse workspace.