OpenEngSB Manual

1.2.0.M1 "Labouring Luigi"

Table of Contents

I. Introduction 1
1. How to read the Manual 2
2. What is the Open Engineering Service Bus 3
3. When to use the OpenEngSB 4
3.1. The OpenEngSB as Base Environment 4
3.2. Reusing integration Components and Workflows 4
3.3. Management Environment 4
3.4. Simple Development and Distribution Management 4
3.5. Simple Plug-Ins and Extensions 4
II. Tutorials 5
4. HowTo - Create a connector for an already existing domain for the OpenEngSB 6
4.1. Goal 6
4.2. Time to Complete 6
4.3. Prerequisites 6
4.4. Step 1 - Use the archetype 6
4.5. Step 2 - Add the dependencies 7
4.6. Step 3 - Configure the connector 8
4.7. Step 4 - Implement the connector 8
4.8. Step 5 - Spring Setup and Internationalization 12
4.9. Step 6 - Start the OpenEngSB with your Connector 13
4.10. Step 7 - Test the new connector 13
III. OpenEngSB Framework 14
5. Quickstart 15
5.1. Writing new projects using the OpenEngSB 15
5.2. Writing Domains for the OpenEngSB 15
5.3. Writing Connectors for the OpenEngSB 15
6. Architecture of the OpenEngSB 17
6.1. OpenEngSB Enterprise Service Bus (ESB) 17
6.2. OpenEngSB Infrastructure 18
6.3. OpenEngSB Components 18
6.4. OpenEngSB Tool Domains 18
6.5. Client Tools (Service Consumer) 18
6.6. Domain Tools (Service Provider) 18
6.7. Domain- and Client Tool Connectors 19
7. Context Management 20
8. Persistence in the OpenEngSB 21
9. Security in the OpenEngSB 22
9.1. Usermanagement 22
9.2. Access control 22
9.3. Authentication 23
10. Workflows 24
10.1. Workflow service 24
10.2. Rulemanager 24
10.3. Processes 24
11. Taskbox 25

11.1. Core Functionality 25
11.2. UI Functionality 25
12. External Domains and Connectors 26
12.1. Proxying 26
12.2. Using JMS proxying 26
13. Deployer services 28
13.1. Connector configuration 28
14. OpenEngSB Platform 29
IV. OpenEngSB Available Domains & Connectors 30
15. Notification Domain 31
15.1. Description 31
15.2. Functional Interface 31
15.3. Connectors 31
16. SCM Domain 32
16.1. Description 32
16.2. Functional Interface 32
16.3. Connectors 32
17. Issue Domain 33
17.1. Description 33
17.2. Functional Interface 33
17.3. Connectors 33
18. Report Domain 34
18.1. Description 34
18.2. Functional Interface 34
18.3. Connectors 34
19. Build Domain 35
19.1. Description 35
19.2. Functional Interface 35
19.3. Connectors 35
20. Test Domain 36
20.1. Description 36
20.2. Functional Interface 36
20.3. Connectors 36
21. Deploy Domain 37
21.1. Description 37
21.2. Functional Interface 37
21.3. Connectors 37
22. Auditing Domain 38
22.1. Description 38
22.2. Functional Interface 38
22.3. Connectors 38
23. Multi-Domain Connectors 39
23.1. Connectors 39
V. OpenEngSB Commiters & Contributors 40
24. Getting Started as a Developer 41
24.1. Getting comfortable with the infrastructure 41
24.2. Prerequisites 42

24.3. Starting OpenEngSB 42
24.4. Using Eclipse 43
24.5. Using Other IDEs than Eclipse 43
24.6. Git Documentation 43
24.7. Useful Tools 45
25. How To Create an Internal Connector 48
25.1. Prerequisites 48
25.2. Creating a new connector project 48
25.3. Project Structure 50
25.4. Integrating the Connector into the OpenEngSB environment 50
26. How To Create an Internal Domain 51
26.1. Prerequisites 51
26.2. Creating a new domain project 51
26.3. Components 53
26.4. Connectors 54
27. Prepare and use Non-OSGi Artifacts 55
27.1. Create Wrapped Artifacts 55
27.2. Tips and Tricks 56
28. Release and Release Process 57
28.1. Releases and the OpenEngSB 57
28.2. Git Branches 58
28.3. Configure Maven 58
28.4. Adapt Jira 59
28.5. Perform the release 60
28.6. Spread the News 60
28.7. Prepare Changelog 61
29. Admin 62
29.1. Infrastructure 62
29.2. Logo Locations and Upgrade 63
30. Project Roles 65
30.1. Users 65
30.2. Contributors 65
30.3. Commiters 65
30.4. Project Comitee Members 65
31. Java Coding Style 66
31.1. Sun Coding Guidelines 66
31.2. General 66
31.3. Naming 69
31.4. No clutter 69
31.5. Exception Handling 69
31.6. Tests 70
31.7. XML Formatting 70
32. Writing Code 72
32.1. Maven POM files in the OpenEngSB 72
32.2. Using the same dependencies as the OPENENGSB 73
32.3. Making UI Tests Localizable 73
32.4. How to write tests 73

33. Recommended Eclipse Plug-ins for Developers 75
33.1. Properties Editor 75
33.2. Spring IDE 75
33.3. Eclipse CS 75
33.4. Drools 75
34. Writing Documentation 76
34.1. General Documentation Guidelines 76
34.2. Document a domain or connector 76
34.3. Using Docbook 77

Part I. Introduction

This parts provides general information to the project, the document, changelog and similar data which fits neither in the framework description nor in the contributor section.

The target audience of this part are developers, contributors and managers.

Chapter 1. How to read the Manual

Like any open source project we have the problem that writing documentation is a pain and nobody is paid for doing it. In combination with the rapidly changing OpenEngSB source base this will lead to a huge mess within shortest time. To avoid this problem we've introduced regular documentation reviews and, more importantly, the following rules which apply both for writing the document and for reading it.

- The manual is written as short and precise as possible (less text means lesser to read and even lesser to review)
- The manual does not describe how to use an interface but only coarse grained concepts in the OpenEngSB. Since the OpenEngSB is not an end user application, but rather a framework for developers we expect that Javadoc is no problem for them. Writing Javadoc and keep it up to date is still hard for developers, but much easier than maintaining an external document. Therefore, all concepts are explained and linked directly to the very well documented interfaces in the OpenEngSB on Github. To fully understand and use them you'll have to read this manual parallel to the interface documentation in the source code.

Chapter 2. What is the Open Engineering Service Bus

In engineering environments a lot of different tools are used. Most of these operate on the same domain, but often interoperability is the limiting factor. For each new project and team member tool integration has to be repeated again. In general, this ends up with numerous point-to-point connectors between tools which are neither stable solutions nor flexible ones.

This is where the Open (Software) Engineering Service Bus (OpenEngSB) comes into play. It simplifies design and implementation of workflows in an engineering team. The engineering team itself (or a process administrator) is able to design workflows between different tools. The entire description process happens on the layer of generic domains instead of specific tool properties. This provides an out of the box solution which allows typical engineering teams to optimize their processes and make their workflows very flexible and easy to change. Also, OpenEngSB simplifies the replacement of individual tools and allows interdepartmental tool integration.

Project management is set to a new level since its possible to clearly guard all integrated tools and workflows. This offers new ways in notifying managers at the right moment and furthermore allows a very general, distanced and objective view on a project.

Although this concept is very powerful it cannot solve every problem. The OpenEngSB is not designed as a general graphical layer over an Enterprise Service Bus (ESB) which allows you to design ALL of your processes out of the box. As long as you work in the designed domains of the OpenEngSB you have a lot of graphical support and other tools available making your work extremely easy. But when leaving the common engineering domains you also leave the core scope of the service bus. OpenEngSB still allows you to connect your own integration projects, use services and react on events, but you have to keep in mind that you're working outside the OpenEngSB and "falling back" to classical Enterprise Application Integration (EAI) patterns and tools.

However, this project does not try to reinvent the wheel. OpenEngSB will not replace the tools already used for your development process, it will integrate them. Our service bus is used to connect the different tools and design a workflow between them, but not to replace them with yet another application. For example, software engineers like us love their tools and will fight desperately if you try to take them away. We like the wheels as they are, but we do not like the way they are put together at the moment.

Chapter 3. When to use the OpenEngSB

The OpenEngSB project has several direct purposes which should be explained within this chapter to make clear in which situations the OpenEngSB can be useful for you.

3.1. The OpenEngSB as Base Environment

OSGi is a very popular integration environment. Instead of delivering one big product the products get separated into minor parts and deployed within a general envioronment. The problem with this concept is to get old, well known concepts up and running in the new environment. In addition tools such as PAX construct allow a better integration into Apache Maven, and extended OSGi runtimes, such as Karaf allow a richer and easier development. Neverthless, settting up such a system for development means a lot of hard manual work. Using the OpenEngSB such systems can be setup within minutes.

3.2. Reusing integration Components and Workflows

The OpenEngSB introduces a new level of ESB. Development with all typical ESBs mean to start from the ground and develop a complete, own environment, only using existing connectors. Using the OpenEngSB not only connectors but an entire integrated process, workflow and event environment waits for you. In addition connectors to different tools can not only be adapted to the specific needs, but also simply replaced by other connectors, using the Domain concept.

3.3. Management Environment

The OpenEngSB delivers a complete management and monitoring environment. While this environment can be added to your project standalone (similar to e.g. Tomcat management console) you also have the possibility to completely integrate the OpenEngSB management enviornment into your Apache Wicket application.

3.4. Simple Development and Distribution Management

While typical ESB have to be installed seperately from your application the OpenEngSB is delivered with your application. Develop your application in the OpenEngSB environment and scripts to embed your application into the OpenEngSB are provided. In addition easy blending allows to adapt the OpenEngSB visually to your needs and cooperate design.

3.5. Simple Plug-Ins and Extensions

The OpenEngSB provides the infrastructure for a rich Plug-In and extension system. Using maven archetypes Plug-Ins can be created, uploaded and provided to all other OpenEngSB installations or applications using the OpenEngSB.

Part II. Tutorials

This part contains tutorials for the OpenEngSB.

Chapter 4. HowTo - Create a connector for an already existing domain for the OpenEngSB

4.1. Goal

This tutorial describes examplary for all connectors the implementation of an email connector. The email connector implements the interface of the Chapter 15, *Notification Domain*, which is already implemented in the OpenEngSB. Therefore, this tutorial describes the implementation of a connector for an already present domain.

4.2. Time to Complete

If you are already familiar with the OpenEngSB about 30 minutes. If you are not familiar with the OpenEngSB please read this manual from the start or check the homepage for further information.

4.3. Prerequisites

Warning: This section is likely to change in the near future, as domains and connectors are currently separated from the rest of the OpenEngSB project. Currently connectors are developed together with the core system.

For information about how to get started as contributor to the OpenEngSB project and how to get the current OpenEngSB source please read the contributor section of the manual: Part V, “OpenEngSB Commiters & Contributors”.

4.4. Step 1 - Use the archetype

As the development of a connector is a recurring task the OpenEngSB developer team has prepared Maven archetypes and useful scripts, which provide support for the initial creation of a connector. The script for creating a new connector can be found at "/etc/scripts/gen-connector.sh".

Go into the directory "/connector" and run from there the "gen-connector.sh" script. The script generates the result in the directory from where it is started, therefore it is recommended to run it from the "/connector" directory. You can also run it from a different directory and copy the results into the "/connector" directory. Fill in the following values (if no input is provided the default value is kept):

```
Domain Name (is domainname): notification
Domain Interface (is NotificationDomain):
Connector Name: email
Version (is 1.1.0-SNAPSHOT):
Project Name (is OpenEngSB :: Connector :: Email):
```

Now the maven archetype is executed. It asks you to confirm the configuration:

```
groupId: org.openengsb.connector
artifactId: openengsb-connector-email
version: 1.1.0-SNAPSHOT
package: org.openengsb.connector.email
connectorName: Email
```

```
connectorNameLC: email
domainArtifactId: openengsb-domain-notification
domainInterface: NotificationDomain
domainPackage: org.openengsb.domain.notification
name: OpenEngSB :: Connector :: Email
 Y: : y
```

A project named "email" is created with the following structure:

```
email
|
|---pom.xml
|
|---src
     |
     |---main
          |
          |---java
          |     |
          |     |---org
          |          |
          |          |---openengsb
          |                  |
          |                  |---connector
          |                          |
          |                          |---email
          |                                |
          |                                |---EmailServiceManager.java
          |                                |
          |                                |---internal
          |                                       |
          |                                       |---EmailServiceImpl.java
          |                                       |
          |                                       |---EmailServiceInstanceFactory.java
          |
          |---resources
                  |
                  |---META-INF
                  |      |
                  |      |---spring
                  |             |
                  |             |---email-context.xml
                  |
                  |---OSGI-INF
                         |
                         |---l10n
                              |
                              |---bundle.properties
                              |
                              |---bundle_de.properties
```

All these artifacts will be covered during the implementation of the connector and explained in step 2 of this tutorial.

4.5. Step 2 - Add the dependencies

Let's start with the dependencies. As the email connector will be based upon the javax mail libraries, we need to include dependencies for the OSGI versions of these artifacts into the pom file located at "/provision/pom.xml". So we add this dependency to the dependencies section:

```
    <dependency>
  <groupId>org.apache.servicemix.bundles</groupId>
  <artifactId>org.apache.servicemix.bundles.javax.mail</artifactId>
  <version>1.4.1_3</version>
</dependency>
```

4.6. Step 3 - Configure the connector

To configure the connector as part of the OpenEngSB two more things are necessary. At first we have to add the connector to the modules section of its parent pom if it is not already present there. If you have run the "gen-connector" script in the "connector" directory this step should have already been performed automatically for you. To check or manually add the entry, open the file "/connector/pom.xml" and add the new connector to the modules section:

```
  ...
<modules>
  <module>email</module>
...
</modules>
...
```

The second step is necessary to configure Karaf correctly. Please open the file "/assembly/pom.xml" and add the following line:

```
  ...
<profile>
  <id>release</id>
  ...
     <deployURLs>
       ...
       scan-bundle:mvn:org.openengsb.connector/openengsb-connector-email/1.2.0.M1,
       ...
     </deployURLs>
...
```

4.7. Step 4 - Implement the connector

Now you can run the following command in the root folder of the OpenEngSB to create an eclipse project for the new connector:

```
mvn eclipse:eclipse
```

Now import the connector project into Eclipse and implement the email service by implementing the classes EmailServiceImpl.java and EmailServiceInstanceFactory.java. We won't go into detail about the actual mail implementation here, so we encapsulated the mailing functionality in a mail abstraction. While the class EmailServiceImpl is responsible for the realization of the domain interface, the factory

is responsible for creating instances of the email service and for publishing the meta data necessary to configure an instance of the email service. These two classes are now explained in detail.

```
package org.openengsb.connector.email.internal;

import org.openengsb.connector.email.internal.abstraction.MailAbstraction;
import org.openengsb.connector.email.internal.abstraction.MailProperties;
import org.openengsb.core.common.util.AliveState;
import org.openengsb.domain.notification.NotificationDomain;
import org.openengsb.domain.notification.model.Notification;
import org.osgi.framework.ServiceRegistration;

public class EmailServiceImpl implements NotificationDomain {

    private final String id;

    private final MailAbstraction mailAbstraction;
    private ServiceRegistration serviceRegistration;
    private final MailProperties properties;

    public EmailServiceImpl(String id, MailAbstraction mailAbstraction) {
        this.id = id;
        this.mailAbstraction = mailAbstraction;
        properties = mailAbstraction.createMailProperties();
    }

    /**
     * Perform the given notification, which defines message, recipient, subject and
     * attachments.
     */
    @Override
    public void notify(Notification notification) {
        mailAbstraction.send(properties, notification.getSubject(), notification
                .getMessage(), notification.getRecipient());
    }

    /**
     * return the current state of the service,
     *
     * @see org.openengsb.core.common.AliveState
     */
    @Override
    public AliveState getAliveState() {
        AliveState aliveState = mailAbstraction.getAliveState();
        if (aliveState == null) {
            return AliveState.OFFLINE;
        }
        return aliveState;
    }

    public String getId() {
        return id;
    }

    public ServiceRegistration getServiceRegistration() {
        return serviceRegistration;
    }

    public void setServiceRegistration(ServiceRegistration serviceRegistration) {
        this.serviceRegistration = serviceRegistration;
    }

    public MailProperties getProperties() {
        return properties;
```

```
    }
}
```

As you can see, without the mail specific stuff the implementation is quite straight forward. Simply implement the domain interface as well as the getAliveState() method, which is used to query to current status of a tool.

```
package org.openengsb.connector.email.internal;

import java.util.HashMap;
import java.util.Map;

import org.openengsb.connector.email.internal.abstraction.MailAbstraction;
import org.openengsb.core.common.ServiceInstanceFactory;
import org.openengsb.core.common.descriptor.AttributeDefinition;
import org.openengsb.core.common.descriptor.ServiceDescriptor;
import org.openengsb.core.common.validation.MultipleAttributeValidationResult;
import org.openengsb.core.common.validation.MultipleAttributeValidationResultImpl;
import org.openengsb.domain.notification.NotificationDomain;

public class EmailServiceInstanceFactory implements
          ServiceInstanceFactory<NotificationDomain, EmailServiceImpl> {

    private final MailAbstraction mailAbstraction;

    public EmailServiceInstanceFactory(MailAbstraction mailAbstraction) {
        this.mailAbstraction = mailAbstraction;
    }

    private void setAttributesOnNotifier(Map<String, String> attributes,
            EmailServiceImpl notifier) {

        if (attributes.containsKey("user")) {
            notifier.getProperties().setUser(attributes.get("user"));
        }
        if (attributes.containsKey("password")) {
            notifier.getProperties().setPassword(attributes.get("password"));
        }
        if (attributes.containsKey("prefix")) {
            notifier.getProperties().setPrefix(attributes.get("prefix"));
        }
        if (attributes.containsKey("smtpAuth")) {
            notifier.getProperties().setSmtpAuth(Boolean.parseBoolean(attributes.
                    get("smtpAuth")));
        }
        if (attributes.containsKey("smtpSender")) {
            notifier.getProperties().setSender(attributes.get("smtpSender"));
        }
        if (attributes.containsKey("smtpHost")) {
            notifier.getProperties().setSmtpHost(attributes.get("smtpHost"));
        }
        if (attributes.containsKey("smtpPort")) {
            notifier.getProperties().setSmtpPort(attributes.get("smtpPort"));
        }
    }

    /**
     * Called when the {@link #ServiceDescriptor} for the provided service is needed.
     *
     * The {@code builder} already has the id, service type and implementation type
```

```
     * set to defaults.
     */
    @Override
    public ServiceDescriptor getDescriptor(ServiceDescriptor.Builder builder) {
        builder.name("email.name").description("email.description");

        builder
            .attribute(buildAttribute(builder, "user", "username.outputMode",
                    "username.outputMode.description"))
            .attribute(
                builder.newAttribute().id("password").name("password.outputMode")
                    .description("password.outputMode.description").defaultValue("")
                    .required().asPassword().build())
            .attribute(buildAttribute(builder, "prefix", "prefix.outputMode",
                    "prefix.outputMode.description"))
            .attribute(
                builder.newAttribute().id("smtpAuth").name("mail.smtp.auth.outputMode")
                    .description("mail.smtp.auth.outputMode.description")
                    .defaultValue("false").asBoolean().build())
            .attribute(
                buildAttribute(builder, "smtpSender", "mail.smtp.sender.outputMode",
                    "mail.smtp.sender.outputMode.description"))
            .attribute(
                buildAttribute(builder, "smtpPort", "mail.smtp.port.outputMode",
                    "mail.smtp.port.outputMode.description"))
            .attribute(
                buildAttribute(builder, "smtpHost", "mail.smtp.host.outputMode",
                    "mail.smtp.host.outputMode.description")).build();

        return builder.build();
    }

    private AttributeDefinition buildAttribute(ServiceDescriptor.Builder builder,
            String id, String nameId, String descriptionId) {
        return builder.newAttribute().id(id).name(nameId).description(descriptionId)
            .defaultValue("").required().build();

    }

    /**
     * Called by the {@link AbstractServiceManager} when updated service attributes for
     * an instance are available. The attributes may only contain changed values and
     * omit previously set attributes.
     *
     * @param instance the instance to update
     * @param attributes the new service settings
     */
    @Override
    public void updateServiceInstance(EmailServiceImpl instance, Map<String,
            String> attributes) {
        setAttributesOnNotifier(attributes, instance);
    }

    /**
     * The {@link AbstractServiceManager} calls this method each time a new service
     * instance has to be started.
     *
     * @param id the unique id this service has been assigned.
     * @param attributes the initial service settings
     */
    @Override
    public EmailServiceImpl createServiceInstance(String id,
            Map<String, String> attributes) {
        EmailServiceImpl notifier = new EmailServiceImpl(id, mailAbstraction);
        setAttributesOnNotifier(attributes, notifier);
```

```
        return notifier;
    }

    /**
     * Validates if the service is correct before updating.
     */
    @Override
    public MultipleAttributeValidationResult updateValidation(EmailServiceImpl instance,
            Map<String, String> attributes) {
        return new MultipleAttributeValidationResultImpl(true,
                new HashMap<String, String>());
    }

    /**
     * Validates if the attributes are correct before creation.
     */
    @Override
    public MultipleAttributeValidationResult createValidation(String id,
            Map<String, String> attributes) {
        return new MultipleAttributeValidationResultImpl(true,
                new HashMap<String, String>());
    }
}
```

The factory is more interesting with respect to the OpenEngSB. It is used to create and configure instances of the email service. Furthermore it is responsible for publishing which properties a mail notifier needs to be configured in a proper way. The "getDescriptor" method returns a service descriptor, which is created with the help of a builder. This service descriptor contains the properties a mail notifier needs. In this case things like user password, smtp server and so on. The "updateServiceInstance" method updates an already created instance of the mail service. Basically this means setting the properties, which are provided in the attributes map parameter (see "setAttributesOnNotifier" method). The "createServiceInstance" method is responsible for the creation of a new email service. The methods "updateValidation" and "createValidation" are used to check properties before "updateServiceInstance" or "createServiceInstance" are called. As the mail service does not want to check properties beforehand it simply returns that all values are OK.

4.8. Step 5 - Spring Setup and Internationalization

The Maven archetype already created the spring setup for the email service at src/main/resources/META-INF/spring. If properties or constructor arguments are needed for the service factory, they have to be defined in the spring setup here. In our case the mail abstraction has to be injected as constructor argument on the creation of the email service factory.

With regards to internationalization it is necessary to add a name and a description for each property used in the service descriptor (see email service factory). The properties files for English and German are also already created by the Maven archetype and can be found at "src/main/resources/OSGI-INF/l10n/". In our case the bundle.properties file contains the following entries:

```
email.name=Email Notification
email.description=This is a Email Notification Service

username.outputMode = Username
username.outputMode.description = Specifies the username of the email account
```

```
password.outputMode = Password
password.outputMode.description = Password of the specified user

prefix.outputMode = Prefix
prefix.outputMode.description = Subject prefix for all mails sent by this connector

mail.smtp.auth.outputMode = Authentification
mail.smtp.auth.outputMode.description = Specifies if the smtp authentication is on or off

mail.smtp.sender.outputMode = Sender Emailadress
mail.smtp.sender.outputMode.description = Specifies the Emailadress of the sender

mail.smtp.port.outputMode = SMTP Port
mail.smtp.port.outputMode.description = Specifies the Port for the smtp connection

mail.smtp.host.outputMode = SMTP Host
mail.smtp.host.outputMode.description = Specifies the SMTP Hostname
```

As you can see each property is defined with name and description. The same entries can be found in the German properties file (bundle_de.properties) with German names and descriptions.

4.9. Step 6 - Start the OpenEngSB with your Connector

After implementing and testing your connector locally you can try to start up the OpenEngSB with your new connector. Enter the following commands in the root directory of the OpenEngSB to build and start the OpenEngSB in development mode:

```
mvn clean install
mvn pax:provision
```

Now you can enter "list" into the karaf console to check whether your new connector was installed and started.

4.10. Step 7 - Test the new connector

Now you can use the OpenEngSB administration WebApp (available at http://localhost:8090/openengsb) to test your new connector. For more information about how to use the WebApp see the How-to section} of the the OpenEngSB homepage.

Part III. OpenEngSB Framework

This part gives an introduction into the OpenEngSB project and explains its base usage environment and the concepts, such as Domains, Connectors, Workflows and similar important ideas. Furthermore this part covers installation, configuration and usage of the administration interface to implement a tool environment according to your needs.

The target audience of this part are developers and contributors.

Chapter 5. Quickstart

As a developer you have basically two ways in which you can use the OpenEngSB. One option is to use the OpenEngSB as a runtime environment for any project. In addition you've the possibility to write Plug-Ins (Domains, Connectors, ...) for the OpenEngSB. Both cases are explained in this chapter.

5.1. Writing new projects using the OpenEngSB

TBW

5.2. Writing Domains for the OpenEngSB

To crate a new Domain run *../etc/scripts/gen-domain.sh* in the *domain* folder. The script will ask you for the name of your domain. Enter the domain name starting with a lower case letter. For the other questions valid defaults are given.

The new domain project will be added as a submodule. You eventually want to run *mvn eclipse:eclipse* and import the new project in eclipse.

Add the methods your domain supplies to the domain interface. If your domain raises any events add methods like

```
void raiseEvent(YourEvent event);
```

(your event class subtype of Event as single parameter) to the events interface.

5.3. Writing Connectors for the OpenEngSB

To crate a new Connector run *../etc/scripts/gen-connector.sh* in the *connector* folder. The script will ask you for the name of the domain you want to implement. Enter the domain name starting with a lower case letter. You may adapt the name of the implemented domain interface if you it does not match the naming convention. Supply the name of the connector staring with a lower case letter.

The new domain project will be added as a submodule. You eventually want to run *mvn eclipse:eclipse* and import the new project in eclipse.

Implement the domain interface in the supplied class (unfortunately no method stubs are generated).

Unimplemented domain methods should always throw an exception rather than return default value or do nothing. Therefore each domain method without body must throw DomainMethodNotImplementedException to indicate that requested domain functionality is not implemented.

```
@Override
public void foo() {
 throw new DomainMethodNotImplementedException();
}
```

The *ServiceFactory* has to supply a *ServiceDescriptor* that contains all attributes needed to instanciate the Connector. In the methods *createServiceInstance* and *updateServiceInstance* use the provided

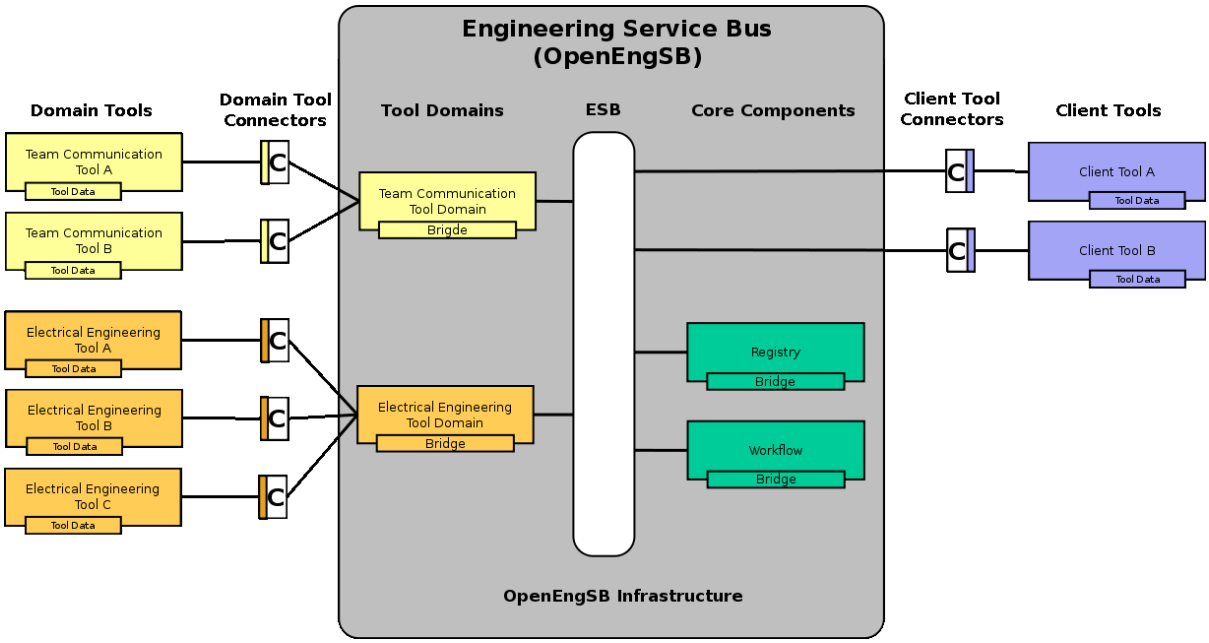
attributes to create a new new instance or update your Connector. The methods *updateValidation* and *createValidation* should do the same but try to validate the provided attributes first and return a validation result.

The generated ServiceManager usually does not have to be changed.

Chapter 6. Architecture of the OpenEngSB

This chapter tries to give a short summary of the most important concepts in the OpenEngSB architecture.

The following graphic shows the architecture of the OpenEngSB. In the center we use a bus system to integrate different modules. In this case we do not use a classical Enterprise Service Bus (ESB), but rather the OSGi service infrastructure via Spring-DM (Section 6.1, “OpenEngSB Enterprise Service Bus (ESB)”). We are using Apache Karaf as the OSGi environment. Karaf is used in this case, instead of a most basic OSGi environment, such as Apache Felix or Eclipse Equinox , because it supports us with additional features as extended console support and the feature definitions. This base infrastructure, including all modifications required for the OpenEngSB is called the Section 6.2, “OpenEngSB Infrastructure”. Within the OpenEngSB Infrastructure so called Section 6.3, “OpenEngSB Components” and Section 6.4, “OpenEngSB Tool Domains” are installed. Both types are written in a JVM compatible language, including OSGi configuration files to run in the OpenEngSB Infrastructure. They are explained later within this chapter. Different tools running outside the OpenEngSB Infrastructure are called Section 6.5, “Client Tools (Service Consumer)” or Section 6.6, “Domain Tools (Service Provider)”, depending on their usage scenario. To integrate and use them within the OpenEngSB so called Section 6.7, “Domain- and Client Tool Connectors” are used. All of these concepts are explained within the next sections.



Technical view of the OpenEngSB highlighting the most important concepts of the integration system

6.1. OpenEngSB Enterprise Service Bus (ESB)

One of the principal concepts for the OpenEngSB development is (if possible) to use already existing and proven solutions rather than inventing new ones. In this manner the OpenEngSB is an extension to the ESB concept. Typical ESBs such as Apache Servicemix or other JBI or ESB implementations always have the feeling to be huge and bloated. Complex integration patterns, messaging, huge

configuration files and similar concepts/problems lead to this feeling. And those feelings are right. They are bloated. The OpenEngsB tries a different approach. Using Karaf as its base framework the environment is VERY lightweight. Depending on your use case you can use different configurations and packages out of the box.

6.2. OpenEngSB Infrastructure

While Apache Karaf provides a rich environment and functionality we're not done with it. Via the Spring-DM extension mechanism, AOP and the OSGi listener model the OpenEngSB directly extends the environment to provide own commands for the console, fine grained security and a full grown workflow model. These extensions are optional and not required if you want to use the platform alone. Add or remove them as required for your use case.

6.3. OpenEngSB Components

These libraries are the OpenEngSB core. The core is responsible to provide the OpenEngSB infrastructure as well as general services such as persistence, security and workflows. To provide best integration most of these components are tied to the OpenEngSB ESB environment. Nevertheless, feel free to add or remove them as required for your use case.

6.4. OpenEngSB Tool Domains

Although each tool provider gives a personal touch to its product their design is driven by a specific purpose. For example, there are many different issue trackers available, each having its own advantages and disadvantages, but all of them can create issues, assign and delete them. Tool Domains are based on this idea and distill the common functionality for such a group of tools into one Tool Domain interface (and component). Tool domains could be compared best to the concept of abstract classes in in object orientated programming languages. Similar to these, they can contain code, workflows, additional logic and data, but they are useless without a concrete implementation. Together with the ESB, the OpenEngSB infrastructure and the core components the tool domains finally result in the OpenEngSB.

6.5. Client Tools (Service Consumer)

Client Tools in the OpenEngSB concept are tools which do not provide any services, but consume services provided by Tool Domains and Core Components instead. A classical example from software engineering for a client tool is the Integrated Development Environment (IDE). Developer prefer to have the entire development environment, reaching from the tickets for a project to its build results, at hand. On the other hand they do not need to provide any services.

6.6. Domain Tools (Service Provider)

Domain Tools (Service Provider) Domain Tools, compared to Client Tools, denote the other extreme of only providing services. Classically, single purpose server tools, like issue tracker or chat server, match the category of Domain Tools best. Most tools in (software+) engineering environments fit of course in both categories, but since there are significant technically differences between them they are described as two different component types.

6.7. Domain- and Client Tool Connectors

Tool Connectors connecto tools to the OpenEngSB environment. They implement the respective Tool Domain interface. As Client Tool Connectors they provide a Client Tool with an access to the OpenEngSB services. Again, Domain- and Client Tool Connectors are mostly mixed up but separated because of their technical differences. Additionally it is worth mentioning that tools can be integrated with more than one connector. This allows one tool to act in many different domains. Apache Maven is an example for such multi-purpose tools, relevant for build, as well as test and deploy of Java projects.

Chapter 7. Context Management

Each project in the OpenEngSB has its own context to store meta information necessary for running inside of the OpenEngSB. The context basically is represented as a tree structure with key-value pairs as leafs.

The context in which a workflow is executed, a rule fired or another action happens can be compared to the project in which the respective action happens. The context store therefore offers the possibility to perform project specific configurations.

The context service can be used to query the context and to insert, update or delete values. Note that under a specific name either a node or a leaf can be found, but not both. That means that the context can be compared to a file system, where context nodes are directories and context leaves files. The leaves in the context contain string key-value pairs.

The current context service extends the context service and provides additional methods for the management new root context entries (which correspond to projects).

The ContextHolder keeps track of the current threads' contexts. Invoking the set- and get-method always manipulate the context of the current Thread. When a new Thread is spawned it inherits the context from the parent thread.

Chapter 8. Persistence in the OpenEngSB

The OpenEngSB has a central persistence service, which can be used by any component within in the OpenEngSB to store data. The service is designed for flexibility and usability for the storage of relatively small amounts of data with no explicit performance requirements. If special persistence features need to be used it is recommended to use a specialized storage rather than the general storage mechanism.

The persistence service can store any Java Object, but was specifically designed for Java Beans.

The interface of the persistence service supports basic CRUD (create, update, retrieve, delete) mechanisms. Instances of the persistence service are created per bundle and have to make sure that data is stored persistently. If bundles need to share data the common persistence service cannot be used, as it does not support this feature. The persistence manager is responsible for the management of persistence service instances per bundle. On the first request from a bundle the persistence manager creates a persistence service. All later requests from a specific bundle should get the exact same instance of the persistence service.

The persistence solution of the OpenEngSB was designed to support different possible back-end database systems. So if a project has high performance or security requirements, which can not be fulfilled with the default database system used by the persistence service, it is possible to implement a different persistence back-end. To make this exchange easier a test for the expected behavior of the persistence service is provided.

Chapter 9. Security in the OpenEngSB

9.1. Usermanagement

The OpenEngSB has a central user management service, which can be used for example by an user interface. The service is designed to manage your users. You can create new user and save them to the persistence or retrieve, update and delete them.

The user management needs a back-end database, e.g. the central persistence service of the OpenEngSB.

The interface of the User manager supports basic CRUD mechanisms (create, retrieve, update, delete). The User is the used user model. It holds attributes like a password, username, if the user is enabled, or his account is expired or locked. A user is identified by his username. So the username can not be changed. Another attribute are the authorities. These are the roles granted to the user. These can be for example "ROLE_ADMIN" which defines the user as admin. Depending on the roles, a user can have different rights. For the OpenEngSB-UI a user has to have at least the role "ROLE_USER" which is the default role.

9.2. Access control

Access control is done on the service level. Core-services and connector-instances are all published as OSGi-services. Other services and components always reference these service instances. We use the approach of AOP to achieve security of these services. The openengsb.core.security-bundle publishes a service that serves as a method-interceptor. When it is attached to a service every method call on the service is preceeded with an authorization-check.

A reference to the method-interceptor can be obtained by this line in the spring-context.xml

```
<osgi:reference id="securityInterceptor" interface="org.aopalliance.intercept.MethodInterceptor" />
```

In order to attach it to an existing bean, one has to create a ProxyFactoryBean:

```
<bean id="secureServiceManager" class="org.springframework.aop.framework.ProxyFactoryBean">
  <property name="proxyInterfaces">
    <value>other.ServiceInterface</value>
  </property>
  <property name="interceptorNames">
    <list>
      <value>securityInterceptor</value>
    </list>
  </property>
  <property name="target" ref="<realBean>" />
</bean>
```

When registering a service in code rather than in a spring context.xml this can be done as seen in the AbstractServiceManager

```
import org.springframework.aop.framework.ProxyFactory;
//
// ...
//
```

```
ProxyFactory factory = new ProxyFactory(serviceObject);
factory.addAdvice(securityInterceptor);
OpenEngSBService securedService = (OpenEngSBService) factory.getProxy();
```

The decision about the allowing the user access to a service as made by looking at the services instanceId. Therefore, all services that are to be placed under this access control, must implement OpenEngSBservice, and make sure the instanceId is unique enough to ensure security. You may want to derive your service-class from AbstractOpenEngSBService.

The persistence of the security-bundle manages a set of GrantedAuthorities (Roles) for each instanceId. There is one exception: Users with "ROLE_ADMIN" are always granted access.

9.3. Authentication

This chapter describes how to deal with security in internal bundles and client projects

For authentication the OpenEngSB provides an AuthenticationProvider as a service. It's obtainable via blueprint.

```
        <reference interface="org.springframework.security.authentication.AuthenticationManager" />
```

This service is able to authenticate users (org.springframework.security.authentication.UsernamePasswordAuthenticationToken) and bundles (org.openengsb.core.security.BundleAuthenticationToken). The use of the former is pretty obvious. The latter is used for authentication for internal actions, that require elevated privilages. This authentication should be used with caution, and never be exposed externally for security reasons.

Chapter 10. Workflows

The OpenEngSB supports the modeling of workflows. This could be done by two different approaches. First of all a rule-based event approach, by defining actions based on events (and their content) which were thrown in or to the bus. Events are practical for "short-time handling" since they are also easy to replace and extend. For long running business processes the secondary workflow method could be used which is based on Section 10.3, “Processes” described in Drools-Flow.

The workflow service takes "events" as input and handles them using a rulebased system (JBoss Drools). It provides methods to manage the rules.

The workflow component consists of two main parts: The RuleManager and the WorkflowService.

10.1. Workflow service

The workflow service is responsible for processing events, and makes sure the rulebase is connected to the environment (domains and connectors). When an event is fired, the workflow-service spawns a new session of the rulebase. The session gets populated with references to domain-services and other helper-objects in form of global variables. A drools-session is running in a sandbox. This means that the supplied globals are the only way of triggering actions outside the rule-session.

10.2. Rulemanager

The rule manager provides methods for modifying the rulebase. As opposed to plain drl-files, the rulemanager organized the elements of the rulebase in its own manner. Rules, Functions and flows are saved separately. All elements share a common collection of import- and global-declarations. These parts are sticked together by the rulemanager, to a consistent rulebase. So when adding a new rule or function to the rulebase, make sure that all imports are present before. Otherwise the adding of the elements will fail.

10.3. Processes

In addition to processing Events in global/context-specific rules, it is also possible to use them to control a predefined workflow. The WorkflowService provides methods for starting and controlling workflow-processes.

When the workflow service receives an event, it is inserted into the rulebase as a new fact (and rules are fired accordingly). In addition the event is "signaled" to every active workflow-process. Workflow logic may use specific rules to filter these events.

Chapter 11. Taskbox

The Taskbox enables you to combine workflows with Human Interactions.

11.1. Core Functionality

All workflows started in the OpenEngSB are supplied with the global variable ProcessBag. Inside the workflow you can populate the ProcessBag with your data. As soon as Human Interaction is needed you have to incorporate the sub-workflow "humantask", which wraps the ProcessBag into a Task. You can then query the Taskbox service for open Tasks, and manipulate the data inside of the Task (Not necessarily by Human Interaction). When you are finished, you again call the Taskbox service and supply the changed Task. The changed data gets extracted and is handed back over to the workflow.

11.2. UI Functionality

The Webtaskbox service provides additional UI Features, if you want to integrate the Taskbox-Concept into a wicket Page. You can query the Webtaskbox service for an Overview Panel that displays all open Tasks. If the default Overview Panel doesn't fit your needs exactly you can develop your own UI-Component using the (Core-)Taskboxservice. If you navigate onto a specific Task the Overview Panel displays a (default) Detail Panel populated with the values of the Task, if there is no custom Panel registered for the supplied tasktype. You can develop your own Detail-Panels and register them for a specific Tasktype via the Webtaskbox service.

Chapter 12. External Domains and Connectors

Since tools are mostly neither developed for the OpenEngSB nor written in any way that they can be directly deployed in the OpenEngSB environment a way is required to connect via different programming languages than Java and from multible protocols. This section covers the examples in different languages and protocols, how such a thing can be achived.

12.1. Proxying

The proxy mechanism allows for any method call to be intercepted.

12.1.1. Proxying internal Connector calls

The proxy mechanism allows to create proxies for any domain. To create a proxy you have to provide a port id, destination and service id to call on the remote service. A Port encapsulates the protocol that is used to call another service. There are an OutgoingPort and IncomingPort interface for respective purposes. The port id is used to load the Port via OSGI. To include a Port in OPENENGSB it just has to be exported via OSGI. The destination is a string that has to be correctly interpreted by the port to call the remote server. The service id is added as metadata do identify the service that shold get called on the remote server. It may not be needed for certain implementations.

The proxy calls the CallRouter which redirects the methodcall to the respective Port. Security is implemented in this layer.

12.2. Using JMS proxying

The current JMS Connector allows for internal method calls being redirected via JMS as well as internal services being called.

12.2.1. Proxying internal Connector calls

Whenever now a method is sent through the JMS Port the call is marshalled and sent via JMS to a queue named "receive"". The marshalling is done via JSON. The mapping has the parameters methodName, args, classes, metadata and potentially answer and callId. methodName gives the name of the method to call. Args are the serialised parameters of the method. classes are the types of the arguments. This way it is easy to unmarshall the args into the appropriate classes. metadata is a simple Map which stores key value pairs. answer can simply be yes or no and denotes if the methodcall wants an answer to the call. callId gives the return queue the caller will listen to for an answer.

An answer can have the type, arg, className and metaData properties. type can be Object, Exception or Void. arg is the serialised form of the return argument. className is the runtime class of the arg for deserialisation. metadata is a simple key value store.

12.2.2. Calling internal Services

To call an internal Service send a methodcall as described before to the "receive" queue on the on the server you want to call. The service works exactly as defined before.

By default a JMS Broker is started on port 6549.

12.2.3. Examples

12.2.3.1. Connect With Python

To test the OPENENGSB JMS implementation with Python please follow the instructions

The example can be downloaded here

12.2.3.2. Connect With CSharp

The CSharp connector is written on basis of the Apache ActiveMQ NMS connector and with help of the Spring NmsTemplate. The code is checked into the repository and could be found in `nonjava/csharp`. There an EngSB.sln file. This project file has been developed with SharpDevelop 3, but is also tested with VisualStudio 2008 CSharp Express Edition with the .Net Framework 3.5.

The example can be downloaded here

12.2.3.3. Connect With Perl

As shown in this example you can connect to the OpenEngSB in a similar way as with Python or CSharp.

The example can be downloaded here

Chapter 13. Deployer services

The OpenEngSB supports file-based configuration through its deployer services. These services are constantly checking the "config/" directory for new/changed/deleted configuration files.

If a new file is created, its configuration is loaded into the OpenEngSB. When the file changes the configuration is updated and when it is deleted the configuration is unloaded. Each deployer handles a different type of configuration file represented by different file name extensions. Details and structure of these files are covered in this section.

13.1. Connector configuration

The connector deployer service creates, updates or deletes instances of connector services.

All files in the "config/" directory with the extension ".connector" are handled by the connector deployer. The .connector files have to be simple property files containing the configuration properies of a certain connector service and their values. Additionaly the property with the key "connector" defines which type of connector shouold be created (corresponds to the "connector" property in the service definition) and the property with the key "id" defines the new service id.

Example 13.1. Example .connector configuration file for the email connector

```
connector = email
id = testServiceId
user = user
password = test
prefix = [test]
smtpAuth = true
smtpSender = test@test.com
smtpPort = 25
smtpHost = smtp.testserver.com
```

Chapter 14. OpenEngSB Platform

The aim of the OpenEngSB project, as for every open source project, is to make the life of everyone better. Or at least the life of engineers :). With that said, we want to support projects using the OpenEngSB as base environment, or providing domains and connectors. While it is easy to find a source repository and use the OpenEngSB (because of its business friendly Apache 2 license), it is not that easy to get the visibility your project earns. We want to provide you with this visibility by including your project into the OpenEngSB product family. Basically we provide you with the following infrastructure:

- Sub domain within the OpenEngSB: yourproject.openengsb.org
- Upload space for a homepage at yourproject.openengsb.org
- Two mailinglists (yourproject-dev@openengsb.org and yourproject-user@openengsb.org)
- A git repository at github.com/openengsb/yourpoject
- A place at our issue tracker
- A place at our build server

To get your project on the infrastructure you have to use the Apache 2 license for your code and use the OpenEngSB. It is not required to have any existing source base. Simply send your project proposal to the openengsb-dev mailing list and we'll discuss your project. Don't be afraid; it's not as hard as it sounds ;)

Part IV. OpenEngSB Available Domains & Connectors

This part gives an overview about the domains and their functionality the OpenEngSB supports out of the box. Furthermore each connector and necessary external tool configuration is explained.

The target audience of this part are developers and contributors.

Chapter 15. Notification Domain

The notification domain is an abstraction for basic notification services, like for example email notification.

15.1. Description

The notification domain provides the functionality for sending notifications to a specific recipient.

15.2. Functional Interface

Link to the Java Domain Interface on Github. This interface also contains information about events raised by this domain.

15.3. Connectors

15.3.1. Email Connector

The email connector is a simple notification connector based on the java mail API.

15.3.1.1. External Tool Configuration

No external tool configuration is necessary.

Chapter 16. SCM Domain

The source code management (SCM) domain is the tool domain for all SCM tools, like Git or Subversion.

16.1. Description

The SCM Domain polls external repositories for changes of content under source control and provides functionality to copy/export the repository content for further processing.

16.2. Functional Interface

Link to the Java Domain Interface on Github. This interface also contains information about events raised by this domain.

16.3. Connectors

16.3.1. Git Connector

The Git Connector is a SCM tool connector for the Git fast version control system.

16.3.1.1. External Tool Configuration

The external Git repository must be anonymously accessible with one of the following protocols:

1. git
2. http
3. ftp

No further configuration is needed.

Chapter 17. Issue Domain

The issue domain is the tool domain for all issue tracking tools, like Jira, Trac or Mantis.

17.1. Description

The issue Domain provides the possibility to create, update, delete and comment issues.

17.2. Functional Interface

Link to the Java Domain Interface on Github. This interface also contains information about events raised by this domain.

17.3. Connectors

17.3.1. Trac Connector

The Trac Connector is a issue tool connector for the Trac project management and issue tracker system.

17.3.1.1. External Tool Configuration

The external Trac tool has to be accessible via XmlRpc. For this purpose the XmlRpcPlugin has to be installed (see http://trac.edgewall.org/wiki/PluginList).

17.3.2. Jira Connector

The Jira Connector is an issue connector for the Jira issue and project tracking system.

17.3.2.1. External Tool Configuration

The Jira connector should work with a default Jira installation. However, make sure that the RPC plugin is enabled as described in the Jira XML-RPC Overview.

Chapter 18. Report Domain

The report domain is the tool domain for report generation and management tools.

18.1. Description

The report domain supports basic report generation functionality, like event logging and manual report building. Furthermore it provides basic report management features, like persistent storage of reports and a category system for report storage.

18.2. Functional Interface

Link to the Java Domain Interface on Github. This interface also contains information about events raised by this domain.

18.3. Connectors

18.3.1. Plaintext Report Connector

The plain text report tool connector is simple implementation of the report domain, which generates plain text reports.

18.3.1.1. External Tool Configuration

No external configuration is needed.

Chapter 19. Build Domain

The build domain is a domain for all build tools, like Maven or Ant.

19.1. Description

The build domain builds a specific pre-configured project or suite of projects.

19.2. Functional Interface

Link to the Java Domain Interface on Github. This interface also contains information about events raised by this domain.

19.3. Connectors

This domain is implemented by the Section 23.1.1, “Maven Connector”, which supports multiple domains.

Chapter 20. Test Domain

The test domain is a domain for all test tools, like Maven.

20.1. Description

The test domain runs all tests for a specific pre-configured project or suite of projects.

20.2. Functional Interface

Link to the Java Domain Interface on Github. This interface also contains information about events raised by this domain.

20.3. Connectors

This domain is implemented by the Section 23.1.1, “Maven Connector”, which supports multiple domains.

Chapter 21. Deploy Domain

The deploy domain is a domain for all deploy tools, like Maven.

21.1. Description

The deploy domain deploys a specific pre-configured project or suite of projects.

21.2. Functional Interface

Link to the Java Domain Interface on Github. This interface also contains information about events raised by this domain.

21.3. Connectors

This domain is implemented by the Section 23.1.1, “Maven Connector”, which supports multiple domains.

Chapter 22. Auditing Domain

The auditing domain provides various auditing services

22.1. Description

The auditing domain stores auditing logs for later retrieval

22.2. Functional Interface

Link to the Java Domain Interface on Github. This interface also contains information about events raised by this domain.

22.3. Connectors

22.3.1. Memory Auditing Connector

The memory auditing connector stores every audit call in memory for later retrieval.

Chapter 23. Multi-Domain Connectors

Some connectors support multiple domains. Therefore they cannot be categorized into a specific domain.

23.1. Connectors

23.1.1. Maven Connector

The Maven Connector is a build, test and deploy tool connector for Maven.

23.1.1.1. External Tool Configuration

The Maven executable has to be on the system path to make this connector work.

Part V. OpenEngSB Commiters & Contributors

This part explains how to develop additional domains, connectors and similar parts. In addition it explains the rules and infrastructure according to which the project is developed.

The target audience of this part are contributors.

Chapter 24. Getting Started as a Developer

This chapter describes the basic steps to get started as a developer for the OpenEngSB project.

24.1. Getting comfortable with the infrastructure

As any open source project the OpenEngSB development depends on a wide range of different infrastructural and communication methods to get things done. The following sub-chapters describe the different tools, their location and usage in the OpenEngSB development process.

24.1.1. Mailing Lists

The most important communication medium for the OpenEngSB development team is email. Mostly all of the vital design, architectural and infrastructural decisions are discussed on the OpenEngSB developer list. Therefore, the first step to get involved into the development of the OpenEngSB is to register to the Google Groups OpenEngSB Developer Mailing List and say hello world.

While notifications from the Hudson Build Server, about code commits and Jira issues are vital for the OpenEngSB core developer, they may not be as interesting for you. If you get annoyed by the automatically generated notification mails ignore all mails from openengsb@gmail.com and noreply@github.com to openengsb-dev@googlegroups.com. Please remember it is important to configure both, `to` and `from` in your filter. Both addresses will also send notifications directly to you which are important and should not be ignored!

24.1.2. Jira Issue Tracker

All issues are stored within a Jira instance reachable at issues.openengsb.org. Please use the issue tracker to keep track of all bugs, ideas and new features you're currently working or of which you think they might be interesting.

24.1.3. Code Repository

As for any open source project the source code is public available. We've chosen Github for this task. The project is available at github.com/openengsb/openengsb.

As explained later within this document Github is not only used to store our code, but also for collaboration, code review and patch-tracking.

24.1.4. Maven Repository

The OpenEngSB is available at Maven Central. We still have our own Maven repository at maven.openengsb.org/ and snapshots are available via the sonatype Maven repository at http://oss.sonatype.org.

24.1.5. Build Server

The master and integration branch of the OpenEngSB repository are watched and built by a Hudson build server instance available at build.openengsb.org. Notifications about failures are directly sent to the OpenEngSB developer list.

24.2. Prerequisites

First of all the latest JDK has to be installed on the system and the `JAVA_HOME` variable has to be set accordingly. All further steps are described in the subsections of this chapter.

24.2.1. Installing Git

It is assumed that Git is installed. For Linux your distribution provides already a package for git. Please use the package manager of your distribution (apt, yum, pacman, ...) to install it. For MAC binaries are available at git-scm.com. For MS users cygwin or msysgit. After installing, set at least the following variables:

```
git config --global user.name "Firstname Lastname"
git config --global user.email user@example.com
git config --global core.autcrlf input
```

24.2.2. Installing Maven

Finally download Apache Maven3 and unpack it. Add the path of the maven binary to your `PATH` variable. Furthermore you should set the `MAVEN_OPTS` environment variable to allow Maven to use more RAM. If you don't you'll get Out Of Memory errors.

```
export PATH=$PATH:/path/to/maven/bin
export MAVEN_OPTS='-Xmx1024M -XX:MaxPermSize=512m'
```

Add these commands to `~/.bashrc` to make the settings permanent.

24.3. Starting OpenEngSB

The next step is to get the OpenEngSB source by checking out the current master using git:

```
git clone git://github.com/openengsb/openengsb.git
```

Now start the OpenEngSB by executing

```
mvn clean install pax:provision
```

This command builds, tests and runs the OpenEngSB right from your command-line. Executing the following command will shutdown it again:

```
shutdown
```

24.4. Using Eclipse

Eclipse had been chosen by the OpenEngSB team as the main development environment. After checkout the code the following command creates the required Eclipse project files:

```
mvn install
mvn eclipse:eclipse
```

Start Eclipse and select any workspace. The folder `eclipse-workspace` is ignored in the OpenEngSB project structure for this purpose. But you can choose any other directory if you prefer. At the preference page go to Java/Build Path/Classpath Variables and create a new M2_REPO pointing to `~/.m2/repository`. Now use File, Import..., Existing Projects into Workspace. As the root directory select the root of the OpenEngSB source. Eclipse will list several projects and for now it's best to import them all by clicking Finish.

At `openengsb/etc/eclipse/` eclipse configuration files for formatting and Checkstyle can be found. These files should be used.

24.5. Using Other IDEs than Eclipse

Basically, the OpenEngSB is developed in plain Java, which means any other IDE than Eclipse can be used too. While there are tools for most IDEs to use Checkstyle, but non of it supports the formatting file of the OpenEngSB. Please use Checkstyle, which automatically validates the eclipse formatting rules too.

24.6. Git Documentation

24.6.1. Usage

First of all this chapter explains only the *very* basics of Git and only that parts directly relevant for the development of the OpenEngSB project, but not the entire idea and possibilities of Git. *Please* read some tutorials first to get how to work with Git and see this chapter more as an summary! You may also take a look at the Git Documentation Page and the Pro Git Book.

24.6.2. Github

OpenEngSB is developed at github.com. Please create an account there and explore its features. Specify your real name in the admin tab and add a picture. This makes it easier to associate your commits to you.

24.6.3. Starting up and configure

Before starting to work with Git some settings should be applied to Git. Therefore simply execute the following commands.

```
git config --global user.name "Firstname Lastname"
git config --global user.email user@example.com
git config --global color.ui "auto"
git config --global pack.threads "0"
git config --global diff.renamelimit "0"
git config --global core.autocrlf "input"
```

Additionally execute the special settings for github as could be found on github in the "Account Settings" tab is a point "Global git config information". Please use the two git commands described there

```
git config --global github.user username
git config --global github.token token
```

If you don't already have an SSH key you can create one by executing **ssh-keygen** Simply answer all questions from the application with "enter" without enter any values. Afterwards the content of the `id_rsa.pub` file from your `~/.ssh/` directory should be submitted to github (Account Settings/SSH public keys).

24.6.4. Contributor Workflow

Contributor are all developer who like to contribute to the OpenEngSB project, but not have commit rights to openengsb/openengsb.

Please keep in mind, that this manual is NOT a Git tutorial. Github itself, e.g. provides a great help at help.github.com. All base concepts such as forking, pull-requests, ...

Please start by choose or create a new issue. Now create a new fork of the OpenEngSB at Github (if you've not done already so; otherwise this is explained here). Clone your fork, but also add the original openengsb repository as remote repository. This is also explained here. In difference to the Github tutorial please do not commit to the master, but rather create a new branch named OPENENGSB-ISSUE_NUMBER_YOURE_WORKING_ON. Optionally append /DESCRIPTION (e.g. OPENENGSB-586/mvn-eclipse-download-fix).

```
git checkout -b OPENENGSB-ISSUE origin/BRANCH
```

BRANCH is the point where you like to start your work. If you like to contribute to the head this will be typically integration, but could also be a commit or a complete different branch. This is the OpenEngSB schema for naming branches and we'll really appreciate if you work according to it.

Now hack, commit and push as you like. If you think you're finished execute the `etc/scripts/pre-push.sh` script validating your code, tests, licenses and so on. If everything works without errors create a Github pull request on Github, between the master or integration branch (depending on where you've

created your branch on) and your branch. This process is also explained at help.github.com (here). In addition it will help if you add the link to the pull request to the issue you're working on. A commiter will tend as fast as possible to your request and give feedback or directly merge your commit into the integration/master branch.

24.6.5. Commiter Workflow

The only difference between a commiter and a contributor is that he has to watch and merge branches of contributors. If a commiter is happy with the work of a contributor. Comments and other discussions should be done on the mailing list and/or via the Github review system and pull requests.

In addition commiters typically do not create forks but rather create their branches directly in the OpenEngSB repository. This is done because the repository is covered by the OpenEngSB build server and in addition keeps everything closer together.

24.6.6. Additional Rules

1. (Contributor/Committer) All development is done in branches (also of the core developers) One exception to this rule exists: Small fixes and maintenance work which is NOT related to a new feature and does not exceed 2 commits should be cherry-picked into the master directly.

2. (Contributor/Committer) Rebase is *not* dead (although we use merges). *Never ever* commit local merges. You still should develop in local dev branches and rebasing them with the upstream branches. Only if nobody else has access to your fork you can be sure that nobody changed it!

3. (Committer) If merging branches from forked repositories ALWAYS use the `--no-ff` option for merges; this will always create a merge node (even if a fast-forward merge is possible). This is required to create a clear and consistent history!

4. Avoid backward merges from the master and keep feature branches small! This does not mean that backward merges from master are forbidden. But they should not be done too often, since they create a history not easy to read. Please use the method described on this page (with `--no-ff --no-commit`) to reduce the number of merge nodes.

5. Use *meaningful* feature branch names. Using the merge history in the master you can easily follow the development of features. But this requires (maybe long) good names! In addition, always start with OPENENGSB-NUMBER of the issue you're working on. Try to always do work based on issues. If no issue covers what you're doing create one.

24.7. Useful Tools

24.7.1. openengsb-maven-plugin

The openengsb-maven-plugin is a plugin for Apache Maven, intended to simplify various activities (creating domains or connectors, building a release artifact of the whole project etc.) when developing based on the OpenEngSB.

24.7.1.1. Configuring the openengsb-maven-plugin for your project

To use the openengsb-maven-plugin in your project add the following configuration to your project's pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>org.openengsb</groupId>
        <artifactId>openengsb-maven-plugin</artifactId>
        <version>${openengsb.maven.plugin.version}</version>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

The plugin can now be invoked via **mvn openengsb:<goal>**

24.7.1.2. Available Goals

assemble

Installs the OpenEngSB and skips tests. Furthermore a *nightly* profile is activated if available in your poms.

genConnector

Guides interactively through the creation of a connector and generates the artifact via the connector archetype. For additional info how to create a connector see Chapter 25, *How To Create an Internal Connector*

genDomain

Guides interactively through the creation of a domain and generates the artifact via the domain archetype. For additional info how to create a domain see Chapter 26, *How To Create an Internal Domain*

licenseCheck

Performs a check if appropriate license headers are available in every source file. The licenseCheck mojo wraps the com.mycila.maven-license-plugin. A large part of the default behavior of this mojo can be changed in `src/main/resources/licenseMojo/licenseCheckConfig.xml`. See this site for available configuration options. The openengsb-maven-plugin needs to be reinstalled after changing its default behavior.

licenseFormat

Adds a license header to files where the license header is missing. A large part of the default behavior of this mojo can be changed in `src/main/resources/licenseMojo/licenseFormatConfig.xml`. See this site for available configuration options. The openengsb-maven-plugin needs to be reinstalled after changing its default behavior.

pushVersion

Updates the development version.

Parameters:

- *developmentVersion*

  Default version to use for new local working copy.

provision

Equivalent to execute karaf or karaf.bat per hand after build by mvn clean install in a (typically) assembly directory.

Parameters:

- *provisionPathUnix*

  This setting should be done in the one of the assembly folders and have to point to the final directory where the karaf system, etc configs and so on consist.

- *provisionExecutionPathUnix*

  The path to the executable in the unix archive file

- *additionalRequiredExecutionPathUnix*

  Sometimes it's required that some executable files, provided in *provisionExecutionPathUnix* execute other files which have to made executable to work correctly on themselves. Those files should be specified here.

- *provisionPathWindows*

  This setting should be done in the one of the assembly folders and have to point to the final directory where the karaf system, etc configs and so on consist.

- *provisionExecutionPathWindows*

  The path to the executable in the windows archive file

- *additionalRequiredExecutionPathWindows*

  Sometimes it's required that some executable files, provided in *provisionExecutionPathWindows* execute other files which have to made executable to work correctly on themselves. Those files should be specified here.

releaseNightly

Mojo to perform nightly releases. This mojo activates the *nightly* profile in the project, where you can put your additional configuration for nightly releases.

releaseXXX

TBD

Chapter 25. How To Create an Internal Connector

This chapter describes how to implement a connector for the OpenEngSB environment. A connector is an adapter between an external tool and the OpenEngSB environment. Every connector belongs to a domain which defines the common interface of all its connectors. This means that the connector is responsible to translate all calls to the common interface to the externally provided tool.

25.1. Prerequisites

In case it isn't known what a tool domain is and how it defines the interface for the tool connector then Section 6.4, “OpenEngSB Tool Domains” is a good starting point. If there's already a matching domain for this tool it is strongly recommended to use it. But if this tool requires a new domain it has to be created. This is also described in Chapter 26, *How To Create an Internal Domain*.

25.2. Creating a new connector project

To take the burden of the developer creating the initial boilerplate code and configuration, a Maven archetype is provided for creating the initial project structure. Furthermore, if the new connector is developed inside of the OpenEngSB repository, the `openengsb-maven-plugin` (see Section 24.7.1, “openengsb-maven-plugin”) (or the `etc/scripts/gen-connector.sh` script, which wraps the invocation of the maven plugin) should be used for assisted creation of a new connector project.

25.2.1. Using the Maven Archetype

It is not recommended to use the maven archetype directly, because the `genConnector` goal of the openengsb-maven-plugin executes additional tasks, like the renaming of the resulting project. Furthermore it tries to make sure that the new project is consistent with the naming conventions of the OpenEngSB project.

The following parameters have to be specified to execute the correct archetype:

- archetypeGroupId - the groupId of the OpenEngSB connector archetype.
- archetypeArtifactId - the artifactId of the OpenEngSB connector archetype.
- archetypeVersion - the current version of the OpenEngSB connector archetype.

The following parameters have to be defined for the parent of the new connector, which is not only parent of the connector, but also for the implementation of the domain and all other connectors of this domain.

- parentArtifactId - the artifactId of the project parent.

The following parameters have to be defined for the domain of the new connector.

- groupId - the groupId of the domain.
- domainArtifactId - the artifactId of the domain.

The following parameters have to be defined for the connector.

- artifactId - the connector artifact id. Has to be "openengsb-domains-<yourDomain>-<yourConnector>".
- version - the package for the source code of the domain implementation. Has to be "org.openengsb.domains.<yourDomain>".
- domainInterface - The name of the domain interface.
- parentPackage - The package in which the domain interface can be found.
- package - the package for the connector code. Usually <parentPackage>.<yourConnector> is used.
- name - the name of the implementation module. Has to be "OpenEngSB :: Domains :: <yourDomain> :: <yourConnector>"

Where <yourDomain> has to be replaced by your domain name and <yourConnector> has to be replaced by the respective connector name.

Note that the archetype will use the artifactId to name the project, but the OpenEngSB convention is to use the connector name. Therefore you will have to rename the resulting project (however if you use the `genConnector` mojo, this renaming will be performed automatically). Do not forget to check that the new connector is included in the modules section of the domain parent pom.

25.2.2. Using the `openengsb-maven-plugin:genConnector`

First make sure that the plugin is installed in your local maven repository (see Section 24.7.1, “openengsb-maven-plugin”). Then the `genConnector` mojo should be executed from the connector directory (`connector/`) (alternatively the `etc/scripts/gen-connector.sh` script can be used which invokes the openengsb-maven-plugin for you).

```
connector $ mvn org.openengsb.tooling.pluginsuite:openengsb-maven-plugin:<version>:genConnector
```

The mojo tries to guess as much as possible from your previous input. Guessed values are displayed in brackets. If the guess is what you want, simply acknowledge with `Return`. The following output has been recorded by executing the script in the `connector/` directory:

```
Domain Name [domain]: notification <Enter>
Domain Interface [NotificationDomain]: <Enter>
Connector Name [myconnector]: twitter <Enter>
Version [1.0.0-SNAPSHOT]: <Enter>
Project Name [OpenEngSB :: Domains :: Notification :: Twitter]: <Enter>
```

Only the domain and connector name was set, everything else has been guessed correctly. After this inputs the Maven Archetype gets called and may ask you for further inputs. You can simply hit `Return` each time, because the values have been already set by the mojo. If it finishes successfully the new connector project has been created and you may start implementing.

25.3. Project Structure

The newly created connector project should have the exact same structure as the following listing:

```
-- pom.xml
-- src
   -- main
      -- java
      |  -- org
      |     -- openengsb
      |        -- domains
      |           -- notification
      |              -- twitter
      |                 -- internal
      |                 |  -- MyServiceImpl.java
      |                 |  -- MyServiceInstanceFactory.java
      |                 -- MyServiceManager.java
      -- resources
         -- META-INF
         |  -- spring
         |     -- connector-context.xml
         -- OSGI-INF
            -- l10n
               -- bundle_de.properties
               -- bundle.properties
```

The `MyServiceImpl` class implements the interface of the domain and thus is the communication link between the OpenEngSB and the connected tool. To give the OpenEngSB (and in the long run the end user) enough information on how to configure a connector, the `MyServiceInstanceFactory` class provides the OpenEngSB with meta information for configuring and functionality for creating and updating a connector instances. The `MyServiceManager` class connects connector instances with the underlying OSGi engine and OpenEngSB infrastructure. It exports instances as OSGi services and adds necessary meta information to each instance. Since the basic functionality is mostly similar for all service managers, the `MyServiceManager` class extends a common base class `AbstractServiceManager`. In addition the `AbstractServiceManager` also persists the configuration of each connector, so that the connector instances can be restored after a system restart.

The spring setup in the resources folder contains the setup of the service manager. Here additional bean setup and dependency injection can be configured.

The OpenEngSB has been built with localization in mind. The Maven Archetype already generates two `bundle*.properties` files, one for English (bundle.properties) and one for the German (bundle_de.properties) language. Each connector has to provide localization through the properties files for service and attributes text values. This includes localization for names, descriptions, attribute validators, option values and more. For convenience the `BundleStrings` class is provided on all method calls where text is needed for user representation for a specific locale.

25.4. Integrating the Connector into the OpenEngSB environment

The service manager is responsible for the integration of the connector into the OpenEngSB infrastructure. The correct definition of this service is critical.

Chapter 26. How To Create an Internal Domain

This chapter describes how to implement a domain for the OpenEngSB environment. A domain provides a common interface and common events and thereby defines how to interact with connectors for this domain. For a better description of what a domain exactly consists of, take a look at the architecture guide Chapter 6, *Architecture of the OpenEngSB*.

26.1. Prerequisites

In case it isn't known what a domain is and how it defines the interface and events for connectors, then Section 6.4, “OpenEngSB Tool Domains” is a good starting point.

26.2. Creating a new domain project

To get developers started creating a new domain a Maven archetype is provided for creating the initial project structure. Furthermore, if the new domain is developed in the OpenEngSB repository, the `openengsb-maven-plugin` (see Section 24.7.1, “openengsb-maven-plugin”) or the `etc/scripts/gen-domain.sh` script (which only wraps the invocation of the plugin) should be used for further convenience.

26.2.1. Using the Maven Archetype

It is not recommended to use the maven archetype directly, because the *genDomain* goal of the openengsb-maven-plugin executes additional tasks, like the renaming of the resulting project. Furthermore it tries to make sure that the new project is consistent with the naming conventions of the OpenEngSB project.

The following parameters have to be specified to execute the correct archetype:

- archetypeGroupId - the groupId of the OpenEngSB domain archetype.
- archetypeArtifactId - the artifactId of the OpenEngSB domain archetype.
- archetypeVersion - the current version of the OpenEngSB domain archetype.

The following parameters have to be defined for the parent of the new domain, which is not only parent of the domain implementation, but also for all connectors of this domain.

- groupId - the groupId of the project parent. Has to be "org.openengsb.domains.<yourDomain>".
- artifactId - the artifactId of the project parent. Has to be "openengsb-domains-<yourDomain>-parent".
- version - the version of the domain parent, which is usually equal to the current archetype version.
- name - the name of the parent module. Has to be "OpenEngSB :: Domains :: <yourDomain> :: Parent"

The following parameters have to be defined for the implementation of the new domain.

- implementationArtifactId - the implementation artifact id. Has to be "openengsb-domains-<yourDomain>-implementation".
- package - the package for the source code of the domain implementation. Has to be "org.openengsb.domains.<yourDomain>".
- implementationName - the name of the implementation module. Has to be "OpenEngSB :: Domains :: <yourDomain> :: Implementation"

Where <yourDomain> has to be replaced by your domain name, which is usually written in lower case, like e.g. report for the report domain.

Note that the archetype will use the artifactId to name the project, but the OpenEngSB convention is to use the domain name. Therefore you will have to rename the resulting project. Do not forget to check that the new domain is included in the modules section of the domains pom.

26.2.2. Using `openengsb-maven-plugin:genDomain`

First make sure that the plugin is installed in your local maven repository (see Section 24.7.1, “openengsb-maven-plugin”). Then the `genDomain` mojo should be executed from the domains directory in your OpenEngSB repository (alternatively the `etc/scripts/gen-domain.sh` script can be used which invokes the openengsb-maven-plugin for you).

```
domains $ mvn org.openengsb.tooling.pluginsuite:openengsb-maven-plugin:<version>:genDomain
```

You'll be asked to fill in a few variables the script needs to create the initial project structure. Based on your input, the mojo tries to guess further values. Guessed values are displayed in brackets. If the guess is correct, simply acknowledge with `Return`. As example, the following output has been recorded while creating the Test domain:

```
Domain Name [mydomain]: test <Enter>
Version [1.0.0-SNAPSHOT]: <Enter>
Prefix for project names [OpenEngSB :: Domains :: Test]: <Enter>
```

Only the domain name has been filled in, while the rest has been correctly guessed. After giving the inputs, the Maven archetype gets executed and may ask for further inputs. You can simply hit `Return`, as the values have been already correctly set. If the mojo finishes successfully two new Maven projects, the domain parent and domain implementation project, have been created and setup with a sample implementation for a domain.

26.2.3. Project structure

The newly created domain should have the exact same structure as the following listing:

```
-- implementation
|  -- pom.xml
|  -- src
|     -- main
|     |  -- java
```

```
|     |  -- org
|     |     -- openengsb
|     |        -- domains
|     |           -- mydomain
|     |              -- MyDomain.java
|     |              -- MyDomainEvents.java
|     |              -- MyDomainProvider.java
|     -- resources
|        -- META-INF
|        |   -- spring
|        |      -- mydomain-context.xml
|        -- OSGI-INF
|           -- l10n
|              -- bundle_de.properties
|              -- bundle.properties
-- pom.xml
```

The project contains besides simple stubs for the domain interface, the domain events interface and the domain provider also a resources folder, which contains the spring setup and property files for internationalization.

Although the generated domain does in effect nothing, you can already start the OpenEngSB for testing with `mvn clean install pax:provision` and the domain will be automatically be picked up and started.

The spring setup in the resources folder already contains the necessary setup for this domain to work in the OpenEngSB environment. Furthermore the default implementation proxies for the domain interface, which forwards all service calls to the default connector for the domain and the default implementation of the domain event interface, which forwards all events to the workflow service of the OpenEngSB are configured.

Each OpenEngSB bundle (core, domain, connector) has been designed with localization in mind. E.g. the Maven Archetype already creates to `bundle*.properties` files, one for English (bundle.properties) and one for the German (bundle_de.properties) language. Each connector has to provide localization through the properties files. For domains, this only means localization for a name and description of the domain itself.

26.3. Components

1. Domain interface - This is the interface that connectors of that domain must implement. Operations that connectors should provide, are specified here. Events that are raised by this Domain in unexpected fashion (e.g new commit in scm system) are specified on the Interface. The Raise Annotation and the array of Event classes it takes as an argument are used. If the Raise annotation is put on a method the events that are specified through the annotation are raised in sequence upon a call.

2. Domain event interface - This is the interface that the domain provides for its connectors to send events into the OpenEngSB. The event interface contains a `raiseEvent(SomeEvent event)` method for each supported event type.

3. Domain Provider - The domain provider is a service that provides information about the domain itself. It is used to determine which domains are currently registered in the environment. There is an abstract class, that takes over most of the setup.

4. Spring context - There are three services, that must be registered with the OSGi service-environment. First there is the domainprovider of course. Moreover the domain must provide a kind of connector itself, since it must be able to handle service calls and redirect it to the default-connector specified in the current context. And finally the domain provides an event interface for its connectors, which can be used by them to send events into the OpenEngSB. The default implementation of this event interface simply forwards all events sent through the domain to the workflow service. But domains can also provide their own implementation of their event interface and add data to events or perform other tasks. There is a beanfactory that creates a Java-Proxy that can be used as ForwardService both for the forwarding of service calls from domain to connector and for the forwarding of events to the workflow service. The service call ForwardService looks up the default-connector for the specified domain in the current context and forwards the method-call right to it. The event forward service simply forwards all events to the workflow service of the OpenEngSB.

26.4. Connectors

For information regarding the implementation of connectors for the newly created domain see Chapter 25, *How To Create an Internal Connector*.

Chapter 27. Prepare and use Non-OSGi Artifacts

Basically, wrapped JARs do not differ in any way from basic jars, besides that they are deployable in OSGi environments. They are used as regular jar files in the OpenEngSB. Nevertheless, the wrapping itself is not as painless. This chapter tries to explain the process in detail.

27.1. Create Wrapped Artifacts

This chapter is a step by step guide on how to create a wrapped JAR.

1. In case that no osginized library is available in the public repositories a package has to be created. Because of the simplicity of the process it should be done by hand. First of all create a folder with the name of the project you like to wrap within openengsb/wrapped. Typically the groupId of the bundle to wrap is sufficient. For example, for a project wrapping all Wicket bundles the folder org.apache.wicket is created.

2. As a next step add the newly created folder as a module to the openengsb/wrapped/pom.xml file in the module section. For the formerly created Wicket project org.apache.wicket should be added to the module section.

3. Now create a pom.xml file and a osgi.bnd file in the newly created project folder.

4. The pom.xml contains the basic project information. As parent for the project the wrapped/pom.xml should be used. Basically for every wrapped jar the project has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
OPENENGSB LICENSE
-->
<project>

  <parent>
    <groupId>org.openengsb.wrapped</groupId>
    <artifactId>openengsb-wrapped</artifactId>
    <version>1</version>
  </parent>

  <properties>
    <bundle.symbolicName>wrapped_jar_group_id</bundle.symbolicName>
    <wrapped.groupId>wrapped_jar_group_id</wrapped.groupId>
    <wrapped.artifactId>wrapped_jar_artifact_id</wrapped.artifactId>
    <wrapped.version>wrapped_jar_version</wrapped.version>
    <bundle.namespace>${wrapped.groupId}</bundle.namespace>
  </properties>

  <modelVersion>4.0.0</modelVersion>
  <groupId>${wrapped.groupId}</groupId>
  <artifactId>org.openengsb.docs.${wrapped.groupId}</artifactId>
  <version>${wrapped.version}</version>

  <name>${bundle.symbolicName}</name>

  <packaging>bundle</packaging>

  <dependencies>
    <all_jars_which_should_be_embedded />
  </dependencies>
```

```
</project>
```

5. The osgi.bnd file contains the OSGi specific statements for the maven-bundle-plugin. While the default export and import are already handled in the root pom project specific settings have to be configured here. For example all packages within the bundle-namespace are always exported. This is for most scenarios sufficient. In addition all dependencies found are automatically imported as required. This is generally not desired. Instead the parts of the library which have to be imported should be defined separately. The following listing gives a short example how such a osgi.bnl file can look like. For a full list of possible commands see the maven-bundle-plugin documentation.

```
#
# OPENENGSB LICENSE
#
Embed-Dependency: *;scope=compile|runtime;type=!pom;inline=true

Import-Package: sun.misc;resolution:=optional,\
javax.servlet;version="[2.5.0, 3.0.0)",\
*;resolution:=optional
```

27.2. Tips and Tricks

Although the description above sounds quite simple (and wrapping bundles is simple mostly) still some nasty problems can occur. This section summarizes good tips and ideas how to wrap bundles within the OpenEngSB.

- The best workflow to wrap a bundle is according to our experiences, to execute the previously described steps and simply start the OpenEngSB (pax:provision). Either it works or creates a huge stack of exceptions with missing import statements. Simply try to fulfill one problem, than start again till all references are resolved.
- Embedding artifacts is nothing bad. Although it is recommended to use all references artifacts of a bundle directly as OSGi components it can be such a pain sometimes. Some references are simply not required by any other bundle or are too hard to port. In such cases feel free to directly embed the dependencies in the wrapped jar.

Chapter 28. Release and Release Process

This section provides a step by step description to execute a release of the OpenEngSB. It is relevant for everyone marked in the OpenEngSB Team List as release manager because only they have the required rights to execute the following steps.

28.1. Releases and the OpenEngSB

Every release of the OpenEngSB consists of the following parts: RELEASE.MAJOR.MINOR.TYPE. Every release of this type is available at Maven Central. Optionally SNAPSHOT is appended. Snapshot releases are available from the Sonatype Snapshot repository. This section explains what each modifier means and how it is used within the OpenEngSB.

SNAPSHOTS: Snapshots are always available from the latest build of the OpenEngSB. They are taken from the master branch automatically at each commit.

TYPE: Type could be MX, RCX or RELEASE, where X is a number. While RELEASE marks a final release, ready for use in your production environment, M and RC are typically not ready for production. M stands for Milestone release and is cut every two weeks to present the current state of the OpenEngSB and allow a coarse grained planning and roadmap for the OpenEngSB team. RC, release candidates, are handled differently. After everything is finished and the OpenEngSB teams think that the current work is ready for a release, we provide a release candidate and invite everyone to test the release. If there are any issues with the release we fix them and provide another release candidate. During this process no new features, but only bug fixes are handled. We continue this process as long as there are no new bug reports for a RC for two weeks. Then we re-release the latest release candidate as final release. This process only applies for RELEASE and MAJOR. MINOR is handled differently, as explained later on.

RELEASE is a increasing number used for mayor changes within the OpenEngSB architecture. In addition all methods and interfaces marked as deprecated are removed during such a release. It is also possible that a RELEASE does not enhance any mayor architectural concept but is only used to get rid of all the deprecated methods, generated during MAJOR releases.

MAJOR is the main feature development number of the OpenEngSB. Each release containing new features will be a MAJOR release. Nevertheless, between MAJOR releases architectural concepts are not removed but only set to deprecated. This means they only enhance functionality but try to not break with former releases.

MINOR releases are bug-fix releases. They do not include any new features but only fix bugs within the OpenEngSB. They have no release plan, but are simply cut after each bug-fix.

To visualize the explained process the following example. Assume we have released openengsb-1.0.0.RELEASE. Now we're working on openengsb-1.1.0.RELEASE. Therefore we start developing openengsb-1.1.0.M1 which will be released in two weeks. During the development of 1.1.0.M1 a bug occurs at openengsb-1.0.0.RELEASE. During the development the bug is fixed and openengsb-1.0.1.RELEASE is released. After 1.1.0.M1 we require three additional milestone releases to get feature releases. Six weeks after 1.1.0.M1 we'll release 1.1.0.RC1. From now on we continue to develop 1.2.0.M1 (or 2.0.0.M1, depending on the gravity of the changes) and wait for feedback on 1.1.0.RC1. Now a bug-report occurs for 1.0.1.RELEASE. We fix the bug, release 1.0.2.RELEASE

with the fix. If it also affects 1.1.0.RC1, we fix the bug there too and release 1.1.0.RC2 (still working on 1.2.0.M1(!)). Now assume that some other bug reports are received for 1.0.0.RC2. We fix them and release 1.1.0.RC3. In the meantime we finished 1.2.0.M1 and start work on 1.2.0.M2. Now two weeks after the release of 1.1.0.RC3 without any new bug-reports we re-release 1.1.0.RC3 to 1.1.0.RELEASE (starting the game again from the beginning).

28.2. Git Branches

For the best cooperation between Git and Maven the OpenEngSB team has developed its own workflow with branches during releases. For different project phases (milestone, RC, final, support) different workflows apply.

28.2.1. New Feature Workflow

For new features the already described workflow apply. This means create a feature branch based on the integration branch, add your commits and create a pull request if you're finished. Your changes will be merged (after review) to the integration branch. From time to time the integration branch ins merged into the master, which is pushed as snapshots to sonatype.

28.2.2. Milestone Releases

For milestone releases about one day before a planned release a openensb-1.X.0-release branch is created. This branch can be forward merged to integration as often as liked (no backward merges are allowed). If all final bugs and changes are done the MX version is released on this branch and the branch is merged into integration and deleted again. During this process any number of new features are merged into integration, without affecting the release any longer.

28.2.3. Release Candidates

RCs are the pre-level for final releases. This means, after the openengsb team decides a release is ready to go, two new branch are created from the latest commit AFTER the milestone release (where the mvn versions are set back to the snapshot version): openengsb-1.X.x-dev and openengsb-1.X.x-release. openengsb-1.X.x-dev is used for bug-fixes. Every fix which should also be merged into the integration branch/master should be branched off openengsb-1.X.x-dev and afterewards merged into integration and openengsb-1.X.x-dev. If a release is ready openengsb-1.X.x-dev is merged into openengsb-1.X.x-release, where the release takes place. BUT no merge from openengsb-1.X.x-release to openengsb-1.X.x-release is allowed!

28.2.4. Final and Support Releases

All support and final releases are handled exactly as the RC releases between the openengsb-1.X.x-dev and openengsb-1.X.x-release branch.

28.3. Configure Maven

For the right rights to deploy to maven central and upload maven site to openengsb.org the following entries are required in your ~/.m2/settings.xml file:

```
<settings>
```

```
  <server>
    <id>sonatype-nexus-snapshots</id>
    <username>SONATYPE_USERNAME</username>
    <password>SONATYPE_PASSWORD</password>
  </server>
  <server>
    <id>sonatype-nexus-staging</id>
    <username>SONATYPE_USERNAME</username>
    <password>SONATYPE_PASSWORD</password>
  </server>
  <server>
    <id>OpenengsbWebServer</id>
    <username>OPENENGSB_SERVER_USERNAME</username>
    <password>OPENEGNSB_SERVER_PASSWORD</password>
  </server>
  <profiles>
    <profile>
      <id>milestone</id>
      <properties>
        <gpg.passphrase>GPG_PASSPHRASE</gpg.passphrase>
      </properties>
    </profile>
    <profile>
      <id>release</id>
      <properties>
        <gpg.passphrase>GPG_PASSPHRASE</gpg.passphrase>
      </properties>
    </profile>
    <profile>
      <id>final</id>
      <properties>
        <gpg.passphrase>GPG_PASSPHRASE</gpg.passphrase>
      </properties>
    </profile>
  </profiles>
<settings>
```

All the usernames and passwords can be retrieved from someone marked as administrator in the OpenEngSB Team List.

In addition you have to have a GPG key for your mail address (the same you're using to commit to the OpenEngSB source repository which is uploaded to the MIT Key Server.

28.4. Adapt Jira

A word in front, how Jira is used for the OpenEngSB. Jira is used for bug tracking and release planning. ONLY each Milestone release has its own target. Release candidates and final releases are handled differently. Since we release RC and MINOR releases quite often its much too much administration work to keep JIRA up to date.

Ok, knowing that the release process is simple:

- If you release a milestone release close the release target (e.g. 1.0.0.M1)
- If you release a release candidate create a VERSION.RCX release target and close the old one.
- If you release a final release (MAJOR RELEASE) create a new release target 1.0.X.RELEASE.
- If you release a minor release close the 1.0.X.RELEASE target and create 1.0.(X+1).RELEASE.

28.5. Perform the release

Performing a release is quite simple, because of the maven release plugin and some scripts. Simple follow these steps:

- 1) Execute ./etc/scripts/release-[final|milestone].sh with the path to your repository (e.g. ~/openengsb
- Now that the artifacts are available for sync to maven central you have to push them from the staging to the final repository. Therefore follow the steps as explained here
- If everything works fine execute git push;git push --tags

28.6. Spread the News

Post a message to the OpenEngSB twitter account with the following content:

```
openengsb-VERSION "NAME" released, closing XX issues (JIRA_RELEASE_REPORT_SHORT_URL).
  Try the new features now: http://openengsb.org
```

Mails in this case are not only used for notification but also to get the developers and users to try a new release and report issues and problems. Therefore, we use different templates for different types of releases of the project.

The following template shows a copy and paste template for mails send for a release candidate. This mail should only be sent to the developer mailing list:

```
Hey guys,

I've just uploaded openengsb-1.0.0.RC4 to maven central (Should be available
within the next hour).

Sources can be downloaded here:

https://github.com/openengsb/openengsb/zipball/openengsb-1.0.0.RC4

The binary release can be downloaded here:

http://repo1.maven.org/maven2/org/openengsb/openengsb/1.0.0.RC4/openengsb-1.0.0.RC4.zip

Between openengsb-1.0.0.RC3 and openegnsb-1.0.0.RC4 we've fixed the following
issues:

** Bug
  * [OPENENGSB-548] - jetty7 - felix problems
  * [OPENENGSB-605] - Use png as favicon for openengsb war file and script

** Improvement
  * [OPENENGSB-603] - Context has to be stored persistently and
  * restored on system startup
  * [OPENENGSB-610] - Maven connector has to support the execution of a configurable command

** Task
  * [OPENENGSB-606] - update docs new jira release

** TBD
  * [OPENENGSB-589] - document release process for stable branches
```

```
Please give it a try and report all problems you encounter here:

http://issues.openengsb.org/jira/browse/OPENENGSB/fixforversion/10142

If there are no new issues reported within the next 72 hours I set RC4 as the
final release 1.0.0.RELEASE.

Kind regards,
andreas
```

28.7. Prepare Changelog

The changelog is a file to inform users about the changes in the version they are using. This file should only contain the releases which are done in one branch. E.g. the master will never contain changelog about minor releases; because of the way we handle Jira those changes are captured and included anyhow.

Now the CHANGELOG.md file has to be updated. Therefore the following template with the correct version have to be copied in the current changelog file (the latest version always has the most "on-top" position in the text file):

```
openengsb-VERSION
------------------

Add a General Description

### Highlights
  * [e.g.] org.openengsb.domain.scm.doSomething() is removed

### Details

Copy JIRA issues here
```

The following sections explain shortly what changes belong to which part of the changelog.

28.7.1. General Description

The general description summarizes the most important changes in this release. This is a short and verbal description of the changes.

28.7.2. Highlights

The highlight section could be a little bit more detailed than the general description. Things which should be changed by developers could be explained here and other important points could be lined up here.

28.7.3. Details

The details section contains a copy of the release notes generated by Jira if a developer wants to take a detailed look at the changes included in this release.

Chapter 29. Admin

This section is relevant for everyone marked in the OpenEngSB Team List as administrator. If you require anything of the following points to be done please write to the openengsb-dev mailing list or send a mail directly to one of the administrators.

29.1. Infrastructure

This section describes the OpenEngSB infrastructure and the relevant parts to manage it.

29.1.1. OpenEngSB Infrastructure Server

The main server hosting our selfmaintained infrastructure runs Ubuntu Linux and is hosted under the domain "openengsb.org". The server is mainained remotely via SSH [pw:server].

An apache2 server processes all requests and forwards it to the corresponding service. The config-file that connects the subdomains to the corresponding services is located in /etc/apache2/sites-enabled/000-default.

This forwards point to a directory in /var/www that redirects the browser to the correct page (like build.openengsb.org -> build.openengsb.org/hudson) The tomcat-server for the homepage is located in /var/opt/tomcat. JIRA is located in /var/opt/atlassian-jira-enterprise-4.1.2/ Further all passwd-files to control http-access are located in /etc/apache2

29.1.2. OpenEngSB Build

Hudson is accessible at http://build.openengsb.org. To become an admin create account and write mail to one of the current admins.

29.1.3. OpenEngSB Issuetracker

JIRA is accessible at http://issues.openengsb.org. To become an admin create account and write mail to one of the current admins.

29.1.4. OpenEngSB git

The github is located at http://git.openengsb.org. To become an admin create a github-account (if you don't have one) and write mail to one of the current admins.

29.1.5. OpenEngSB Maven

29.1.5.1. internal

The internal maven-repo is accessible at http://maven.openengsb.org. Use [pw:nexus] to login.

29.1.5.2. external

The external maven-repo hosting released artifacts is located at http://oss.sonatype.org. Use [pw:maven] to login.

29.1.6. OpenEngSB Mailinglist

To obtain admin-access for the mailing lists register google-account (if you don't have one), join mailinglists and write mail to one of the current admins

29.2. Logo Locations and Upgrade

This section describes the locations of the logo and what have to be upgraded to the latest logo. The following items are used in this section and are (should be) all available within openengsb/etc/branding.

- openengsb.png: The full logo of the OpenEngSB in png format. The size is not too important. At every location used it is resized according to the requirements automatically.
- openengsb_small.png: A reduced version of the OpenEngSB logo. The most important thing with this logo is that it have to be rectangular, since some cases require this.
- openengsb.ico: This is the openengsb_small.png logo convert to an ico file. Threfore scale the openengsb_small.png. On unix install imagemagic and png2ico and follow the following steps. Before you start upate openengsb_smal.png in `etc/branding`

```
convert -resize 64x64 openengsb_small.png openengsb64x64.png
convert -resize 32x32 openengsb_small.png openengsb32x32.png
convert -resize 16x16 openengsb_small.png openengsb16x16.png
png2ico openengsb.ico openengsb16x16.png openengsb32x32.png openengsb32x32.png
```

29.2.1. External Infrastructure

This section describes which tools have to be upgraded and how this is done.

- Jira: Use openengsb_small.png as project logo.
- Twitter: Use openengsb.png as background and openengsb_small.png as logo.
- Github: Upgrade gravatar with openengsb_icon.png to upgrade openengsb@gmail.com.
- Facebook: Use openengsb.png for the group logo.
- Google Groups: Use openengsb_small.png for the group logos (in all three lists).

29.2.2. Internal Management Application

This section covers how to upgrade the logos in the internal management application located within openengsb/ui/web.

- src/main/resources/openengsb.png (openengsb.png)
- src/main/resources/openengsb.ico (openengsb.ico)

29.2.3. Documentation

Manual, Maven Site and all additional presentations of the OpenEngSB are covered within this section describing how and where to upgrade a logo.

- docs/homepage/src/site/resources/images/openengsb.png uses openengsb.png to present a banner on the homepage.
- docs/skin/src/main/resources/images/openengsb.ico contains openengsb.ico which is presented as favicon on openengsb.org
- docs/manual/src/main/docbx/resources/images/openengsb.png contains openengsb.png which should be presented on the html and pdf documentation of the OpenEngSB.

Chapter 30. Project Roles

This section describes the how the roles in the OpenEngSB Project are defined.

Basically the OpenEngSB is, from it's structure exactly as any Apache Software Foundation (ASF) project. We split the different roles in User, Contributor, Commiters and Project Comitee Members. In addition the OpenEngSB is developed in a metocracy, similar to ASF the persons doing the most affect the project most. Basically the only reason we're not an ASF project is that we prefer using GIT :)

30.1. Users

Users are persons using the OpenEngSB.

This group does not contribute in the OpenEngSB project in any way. They download the OpenEngSB, use it and may ask questions in the IRC channel or on the mailing lists.

30.2. Contributors

Contributors are users who contribute ideas, issues or pull requests.

Basically the only difference between users and contributors are that they actively contribute to the OpenEngSB in one or another way. Those users have full rights on the issue tracker after they've created an account but are not allowed to access the OpenEngSB core repos with write karma.

30.3. Commiters

Commiters have the same rights as contributors with the difference that they have wirte access to the OpenEngSB Github repositories. They are allowed to directly push changes, but also should review pull-requests.

To become a commiter a person have to be active on different parts of the OpenEngSB. Provide patches, write documentation, answer on the user mailing list and the IRC channel. If a contributor is active for an undefined time the project comitee members may vote to add a contributor to a commiter.

30.4. Project Comitee Members

Project comitee members have the same rights as commiters, with the difference that they are responsible for the project. PMCs relese the OpenEngSB, vote contributors to commiters and commiters to PMCs.

Every commiter can become a PMC through active contribution to the OpenEngSB.

Chapter 31. Java Coding Style

31.1. Sun Coding Guidelines

The OpenEngSB Coding Guidelines are based upon the Code Conventions for the Java Programming Language. There are some additions and deviations for this project.

31.1.1. Line length

A line length of 80 was standard 10 years ago, but with increasing screen size and resolution a length of 120 is more reasonable.

31.1.2. Wrapping

Use the auto-formatter of your IDE. Import the Eclipse Formatter file.

31.1.3. Number of declarations per line

Only one declaration per line is allowed.

31.1.4. Declaration placement

Declare variables where they are needed. It's easier to read and restricts the scope of variables. Don't overshadow variables.

31.1.5. Blank lines

The body of a method should not start with a blank line.

31.2. General

31.2.1. File format

Every Java file has to be UTF-8 encoded and has to use UNIX line endings. Indentations consist of four spaces, tab-stops are not allowed.

31.2.2. Header

Every source file has to start with this header:

```
/**

   Copyright 2010 OpenEngSB Division, Vienna University of Technology

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0
```

```
   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.

 */
```

31.2.3. Duplication

Code duplication has to be avoided at all costs.

31.2.4. Use guards

Guards are a possibility to reduce the amount of nesting. Heavily nested code is much harder to read. Bad:

```
public void foo() {
    if (conditionA) {
        if (conditionB) {
            if (conditionC) {
                // do some work
            }
        } else {
            throw new MyException();
        }
    }
}
```

Good:

```
public void foo() {
    if (!conditionA) {
        return;
    }

    if (!conditionB) {
        throw new MyException();
    }

    if (!conditionC) {
        return;
    }

    // do some work
}
```

31.2.5. Keep methods short

Methods longer than 40 lines are candidates for refactoring. A method should only do one thing and has to be easily understandable. The number of arguments should be minimized. A method should only be at a single level of abstraction.

31.2.6. Use enums

Prefer typesafe enumerations over integer constants.

31.2.7. Avoid use of static members

Static members are a sign of a design error because they are like global variables. It's fine if you declare a constant as final abstract of course.

31.2.8. Use fully qualified imports

Don't import org.example.package.*, instead import the needed classes.

31.2.9. Never declare implementation types

Use interfaces or the abstract base class instead of concrete implementation classes where possible. Don't write:

```
ArrayList<String> names = new ArrayList<String>();
```

Instead use the interface name:

```
List<String> names = new ArrayList<String>();
```

This is especially important in method signatures.

31.2.10. SerialVersionUID

Don't declare serialVersionUID just because your IDE tells you. Have a good reason why you need it. This can cause bugs that are hard to detect.

31.2.11. Restrict scope of suppressed warnings

If you have to suppress a warning make sure you give it the smallest possible scope. This means you should never annotate a whole class with @SuppressWarnings. A method may be acceptable but you should try to annotate the problematic statements instead.

31.2.12. Use String.format()

Use String.format() instead of long concatenation chains which are hard to read.

31.2.13. Array declaration style

Always use

```
 Type[] arrayName;
```

instead of the C-like

```
Type arrayName[];
```

31.2.14. Comments

Don't make funny comments, be professional. All comments have to be in English. Comment what methods do, not how they do it. Do not comment what is already stated in code.

31.3. Naming

31.3.1. Interfaces

Interfaces are not marked by starting their names with I. This exposes more information than necessary and is not Java-like.

31.3.2. Don't abbreviate

Do not use abbreviations if it's not a project wide standard. Long method names are preferable to inconsistency. With automatic code completion this isn't a problem anyway.

31.4. No clutter

- Exception/Log Messages have to be concise. Don't end messages with "...".
- Don't overuse FINAL, use it where you have a good reason something has to be final. Although it doesn't hurt to declare everything as final it clutters the code.
- Don't use history tables in source files. Use the SCM system if you are interested in the changes of a file.
- Don't use the JavaDoc author tag. Also use the SCM system.
- Don't declare unnecessary constructors, especially the empty default constructor.
- Don't make implicit calls explicitly, i.e. calling super(); in every constructor.
- Don't specify modifiers that are implicit, i.e. don't make methods in interfaces `public abstract`.
- Don't initialize fields with null, they are automatically initialized with null.
- Don't use banners in comments.
- Don't use closing brace comments, i.e. } // end if, they are a sign of too long methods.
- Don't comment out code and commit it. This confuses programmers why it is there. Simply delete it, it's still present in the SCM history.

31.5. Exception Handling

- Don't log and throw. Either a exception should be logged or thrown to be processed at a more appropriate place.
- Don't swallow exceptions silently. If you have to do it, you have to make a comment stating the reason.
- Use runtime exceptions where possible.
- Wrap exceptions in a RuntimeException if you don't want to specify the Exception in your method signature and you can't handle it.

- Write meaningful exception message.

31.6. Tests

31.6.1. General

- Make use of JUnit 4 features, e.g. @Test(expected = SomeException.class)
- Tests should not output anything. They have to be automatically verified.
- Don't catch exceptions just to fail manually. Declare the method to throw the exception.
- Install a shutdown hook for test data files. This assures that they will be deleted and the project remains in a clean state.
- Use Mockito for mocking.
- Tests should have descriptive method names. It should be deducible what will be tested. Bad: `testError()`. Good: `invalidInMessage_ShouldReturnErrorResponse()`.

31.6.2. Naming Scheme

The Maven profiles for running the tests are configured to filter based on the naming of the test class. The package layout is just a further convenience for the developer for running the tests manually.

- Unit Tests test one class/method/feature in isolation from their dependencies by using test doubles as replacement. They should be fast and need no special environment setup for execution.
- Filenames end with Test.java
- Located in the normal package structure, i.e. `outer.project.package.inner.project.package`
- Integration Tests combine individual software modules to test their interaction with each other. They do not need a special environment setup for execution.
- Filenames end with IT.java
- Located in `outer.project.package.it.inner.project.package`
- User Tests need a special execution environment and thus are not run automatically during any maven phase.
- Filenames end with UT.java
- Located in `outer.project.package.ut.inner.project.package`

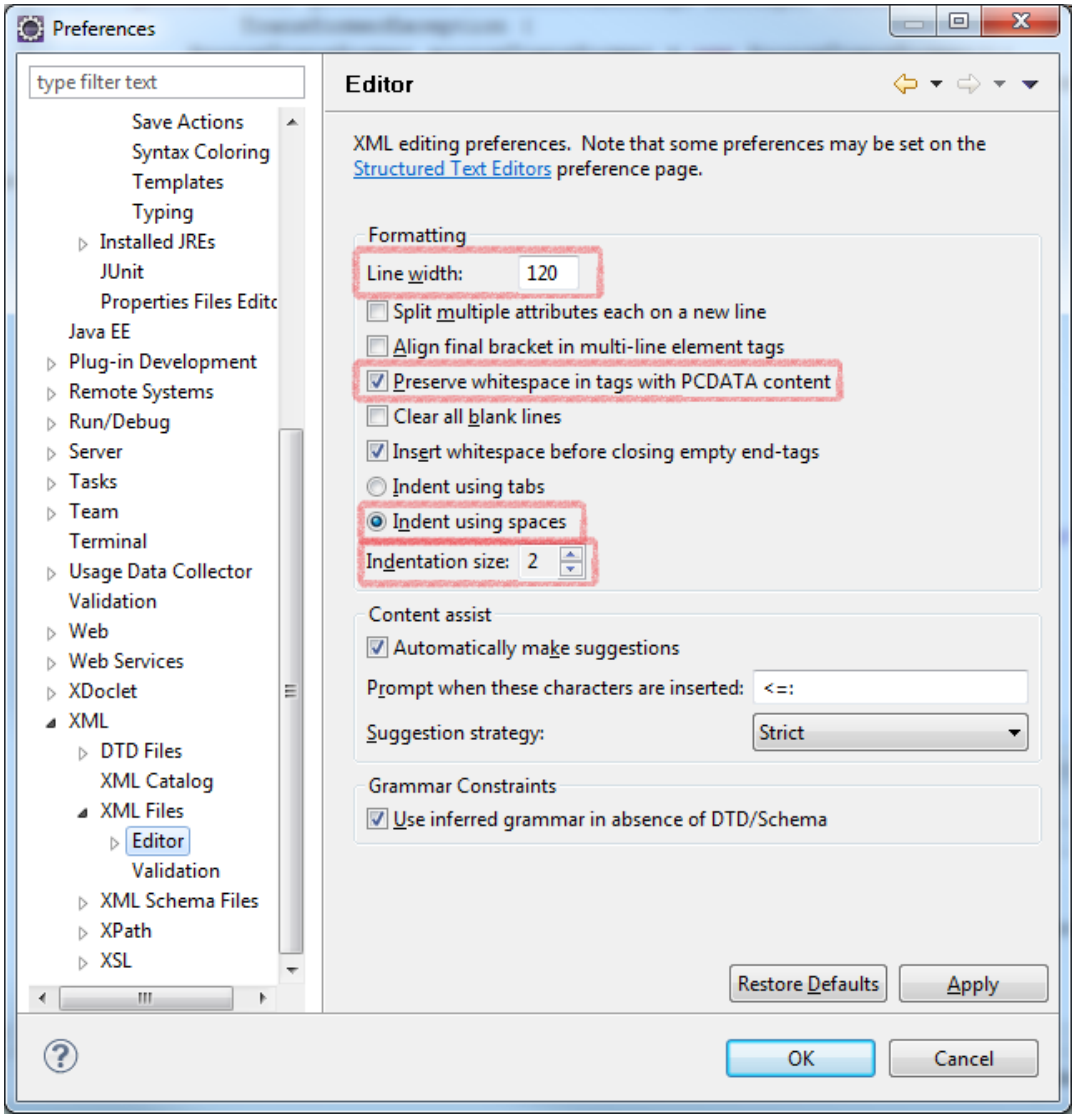
31.7. XML Formatting

31.7.1. File Format

Every XML file has to be UTF-8 encoded and has to use UNIX line endings. Indentations consist of TWO spaces, tabstops are not allowed. The line length shouldn't exceed 120 characters.

31.7.2. Eclipse Settings

If you use Eclipse please choose these settings for your OpenEngSB workspace:



Eclipse XML Settings

31.7.3. Recommended Readings

- Clean Code, Robert C. Martin, 2008
- Effective Java Second Edition, Joshua Bloch, 2008
- 7 tips on writing clean code

Chapter 32. Writing Code

This chapter is intended for developers. There are no special prerequisites. Each part describes what a developer has to look at in specific for the OpenEngSB.

32.1. Maven POM files in the OpenEngSB

Following the guidelines of Maven Central, how a pom should be designed it is required to add the following tags into every and each pom file:

- modelVersion
- groupId
- artifactId
- version
- packaging
- name
- description
- url
- licenses
- scm/url
- scm/connection
- scm/developerConnection

The following listings shows an example of these params for a typical OpenEngSB pom.

```
<modelVersion>4.0.0</modelVersion>
      <groupId>org.openengsb.core</groupId>
      <artifactId>openengsb-core-parent</artifactId>
      <version>1.1.0-SNAPSHOT</version>
      <name>OpenEngSB :: Core :: Parent</name>
      <packaging>pom</packaging>
      <description>Parent project for all OpenEngSB Core classes</description>
      <url>http://www.openengsb.org</url>
      <licenses>
        <license>
          <name>Apache 2</name>
          <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
          <distribution>repo</distribution>
        </license>
      </licenses>
      <scm>
        <connection>scm:git:git://github.com/openengsb/openengsb.git</connection>
        <developerConnection>scm:git:git@github.com:openengsb/openengsb.git</developerConnection>
        <url>http://github.com/openengsb/openengsb</url>
    </scm>
```

32.2. Using the same dependencies as the OPENENGSB

To use the same dependencies as the OPENENGSB project you have to import the shared-plugin-settings project into your dependency management section:

```
    <dependencyManagement>
      <dependencies>
        <dependency>
          <groupId>org.openengsb.build</groupId>
          <artifactId>shared-plugin-settings</artifactId>
          <version>Version of OPENENGSB you use</version>
          <type>pom</type>
          <scope>import</scope>
        </dependency>
      </dependencies>
    </dependencyManagement>
```

This will import all the dependencies with the correct versions into the dependencyManagement section. You can now define the dependencies shared between your project and OPENENGSB in your dependencies section without setting the version.

32.3. Making UI Tests Localizable

If you want to test if specific text is shown in the UI extend LocalisedTest in your UI Test. The constructor automatically loads the correct ResourceBundle and via localization(String resourcename) you can load a localized version of a specific resource string. The default locale is used as to match the locale used by WicketTester.

32.4. How to write tests

The following listings show how to write tests according to the OpenEngSB coding style
The name of the test method has to describe what is going to be tested. After the "_" is described what are the expected results.

```
@Test
public void testBehaviorX_shouldReturnY() {
  //CODE
}
```

In addition to the normal behaviour the coder should also provide a test for the failure behavior.

```
@Test(expected = BehaviorException.class)
public void testBehaviorX_shouldThrowException() {
//CODE
}
```

32.4.2. Technologies for writing test, and how to use them

The OpenEngSB developers decided to use following testing tools:

Caution

Instead of using Assert.assertThat(....) or Mockito.mock(...) we use the static import variant: assertThat(...) and mock(...)

- Asserts: We use Hamcrest instead of JUnit. A simple example how to use Hamcrest assertions is given in the following listing:

```
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.MatcherAssert.is;
[...]
assertThat(realValue, is(expectedValue));
```

- Mocking: We use mockito. A simple example how to use Mockito in a correct way is given in the following listing:

```
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;
import static org.mockito.Mockito.times;
[...]
//mocking code
ExampleMock exampleMock = mock(ExampleMock.class);
when(exampleMock.methodX()).thenReturn(y);
[...] //testing code
//verification
verify(manager, times(1).methodX(Y);
```

Chapter 33. Recommended Eclipse Plug-ins for Developers

The following plug-ins for Eclipse are recommended for the development of the OpenEngSB. If not otherwise stated we recommend the latest stable version of the plug-ins. For information about the basic setup of this plug-ins please take a look into the corresponding plug-in documentation. This section only gives hints for setup if it is OpenEngSB specific.

33.1. Properties Editor

The properties editor can be used to edit the properties files used for internationalization and automatically escapes special characters, like the German "ü".

33.2. Spring IDE

Spring IDE adds support for the Spring Framework to the Eclipse platform. Especially editing the XML configuration files becomes a lot easier, as this plug-in provides code completion and other useful features.

33.3. Eclipse CS

The checkstyle plugin integrates checkstyle into Eclipse. Conformance with checkstyle criteria has to be checked before each push to the repository, so integrating the check into the IDE helps developers to already conform to the checkstyle criteria during development. You have to configure the plug-in to use our checkstyle configuration file, which can be found here and at /tooling/checkstyle/src/main/resources/checkstyle.xml starting from the root directory of the OpenEngSB.

33.4. Drools

The Drools plug-in is handy if you want to edit workflows or Drools rules, because it provides syntax highlighting for rules and a graphical editor for workflows.

Chapter 34. Writing Documentation

This chapter is intended for developers who write documentation. There are no special prerequisites. Part one describes how a chapter should be structured. Part two discusses how domains and connectors should be document. Part three describes how Docbook is used at OpenEngSB.

34.1. General Documentation Guidelines

A chapter should consist of these parts:

Introduction
: It should explained who the target audience for this chapter is and in what case this chapter should be read. There should also be a basic summary of what this chapter is about.

Prerequisites
: Any prerequisites should be listed. Link to the appropriate chapter or to a website to give the reader a good starting point in case they need to learn something else first.

Context
: In the context section the reader should learn in which context this chapter is applicable. If necessary abbrevations and acronyms used in this chapter can be explained here.

Content
: The actual content of this chapter. This should be structured in as many sections as appropriate.

Example
: If possible there should be an example to illustrate the points of the chapter.

Common Problems
: If there are some known pitfalls or bugs they should be described in this section.

Closing Remarks
: In this section the content of the chapter can be summarized once more. The reader should get information on what to do next.

It is not necessary that every part is a docbook section. Parts can be combined if it seems appropriate.

34.2. Document a domain or connector

34.2.1. Domain

Each domain gets their own directory in the user guide at `domains/<the-domain-name>`. The domain-specific documentation should be put in a file named `domain.xml`. The directory will be used to document connectors for the domain.

The documentation of a domain should at least consist of the following parts:

Description
: Describe briefly what the purpose of the Domain is.

Functional interface
: The link to the actual java interface (and any domain models used in the interface) at Github. The domain interface and models should have enough Javadoc to explain the usage.

Events
: If the domain adds new events to the OpenEngSB, the link to the events package at Github should be provided. The meaning of each events should be documented through the Javadoc at the actual class.

34.2.2. Connector

A connector for a specific domain should be documented in the domain-specific directory. Add a new file with the unique name of the connector.

The documentation of a connector should at least conisst of the following parts:

Description
: Provide a description of the external tool and its purpose.

External tool configuration
: A section on how to configure the actual external tool for usage with the OpenEngSB has to be provided.

Support for domain interface
: Any deviation to the provided functionality of the domain should be documented. E.g a connector may only implement parts of the domain interface.

34.3. Using Docbook

This is not a DocBook manual but rather an explanation what type of docbook tags are used in this documentation. If you are new to DocBook you should read DocBook 5: The Definitive Guide.

34.3.1. Tags

DocBook has many tags to choose from. This list describes which tags should be used in which cases.

Tag	Description	Example
<command>	Used for executables	Type <command>ls</command> to get the contents of the directory.
<envar>	Used for environment variables	PATH
<emphasis>	Used to emphasize words in a sentence	This chapter explains only the *very* basics of Git.
<filename>	Used for files and directories	You can set environment variables in <filename>~/.profile</filename>.
<guibutton>	Used to describe buttons in a GUI	Press <guibutton>Next</guibutton> to continue with the process.

Tag	Description	Example
<guilabel>	Used to describe labels in a GUI	Select <guilabel>Copy projects into workspace</guilabel>
<guimenu>	Used to describe menus in a GUI	Go to <guimenu>File</guimenu>, <guimenu>Import...</guimenu>.
<itemizedlist>	Used for bullet type lists	<itemizedlist><listitem>One</listitem><listitem>Two</listitem></itemizedlist>
<listitem>	Used for entries in a list	<itemizedlist><listitem>One</listitem><listitem>Two</listitem></itemizedlist>
<option>	Used for options of commands	<command>mvn</command> <option>clean</option> is used to clean the project.
<orderedlist>	Used for numbered lists	<orderedlist><listitem>One</listitem><listitem>Two</listitem></orderedlist>
<para>	Used for paragraphs	<para>This is a paragraph.</para>
<programlisting>	Used to display code (e.g. XML or Java). Generally it is a good idea to wrap the contents of this tag in a CDATA section.	<programlisting><![CDATA[System.out.println("Hello, world!");]]></programlisting>
<replaceable>	Used for placeholders in examples	Type <command> <replaceable>/path/to/maven</replaceable>
<link>	Used for links to external resources	You should read <link xlink:href="http://www.docbook.org/tdg5/en/html/docbook.html">DocBook 5: The Definitive Guide</link>.
<xref>	Used for internal links	This inserts a link to the description of the the OpenEngSB <xref linkend="architecture" />.
<userinput>	Used for data which is entered by the user	Type <userinput>n</userinput> to overwrite the default values.
<warning>	Used for warnings about a chapter	<warning><para>This chapter is out of date.</para></warning>

34.3.1.1. Including an image

Images can be included in this way:

```
<mediaobject>
```

```
  <imageobject>
    <imagedata id="new" fileref="graphics/testclient_message.png"
      format="png" width="400" align="center" />
    </imageobject>
  <caption>Messaging</caption>
</mediaobject>
```

34.3.1.2. Using a table

There are two types of tables. Normal tables (<table>) and informal tables (<informaltable>)which don't have a caption. Using informal tables should be fine most of the time. Example:

```
<informaltable>
  <colgroup>
    <col width="50" />
    <col width="100" />
  </colgroup>
  <thead>
    <tr>
      <td>
        Name
      </td>
      <td>
        Description
      </td>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>
        table
      </td>
      <td>
        A table with a caption
      </td>
    </tr>
    <tr>
      <td>
        informaltable
      </td>
      <td>
        A table without a caption
      </td>
    </tr>
  </tbody>
</informaltable>
```

34.3.1.3. Generating the documentation

To build the documentation maven with some plugins is used. The full documentation can be generated in one simple step:

```
cd docs
mvn clean install -Pdocs
```

The documentation can be found in `docs/target/docbkx` in HTML and PDF format.