

OpenEngSB Manual

1.0.0.RC1 "Groovy Goofy"

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. OpenEngSB Framework	5
4. Quickstart	6
4.1. Writing new projects using the OpenEngSB	6
4.2. Writing Domains for the OpenEngSB	6
4.3. Writing Connectors for the OpenEngSB	6
5. Architecture of the OpenEngSB	7
5.1. OpenEngSB Enterprise Service Bus (ESB)	7
5.2. OpenEngSB Infrastructure	8
5.3. OpenEngSB Components	8
5.4. OpenEngSB Tool Domains	8
5.5. Client Tools (Service Consumer)	8
5.6. Domain Tools (Service Provider)	8
5.7. Domain- and Client Tool Connectors	9
6. Context Management	10
7. Persistence in the OpenEngSB	11
8. Workflows	12
8.1. Workflow service	12
8.2. Rulemanager	12
8.3. Processes	12
9. Taskbox	13
10. External Domains and Connectors	14
10.1. Proxying	14
10.2. Using JMS proxying	14
11. OpenEngSB Platform	16
III. OpenEngSB Available Domains & Connectors	17
12. Notification Domain	18
12.1. Description	18
12.2. Functional Interface	18
12.3. Connectors	18
13. SCM Domain	19
13.1. Description	19
13.2. Functional Interface	19
13.3. Connectors	19
14. Issue Domain	20
14.1. Description	20
14.2. Functional Interface	20

14.3. Connectors	20
15. Report Domain	21
15.1. Description	21
15.2. Functional Interface	21
15.3. Connectors	21
16. Build Domain	22
16.1. Description	22
16.2. Functional Interface	22
16.3. Connectors	22
17. Test Domain	23
17.1. Description	23
17.2. Functional Interface	23
17.3. Connectors	23
18. Deploy Domain	24
18.1. Description	24
18.2. Functional Interface	24
18.3. Connectors	24
19. Multi-Domain Connectors	25
19.1. Connectors	25
IV. OpenEngSB Committers & Contributors	26
20. Getting Started as a Developer	27
20.1. Getting comfortable with the infrastructure	27
20.2. Prerequisites	28
20.3. Starting OpenEngSB	28
20.4. Using Eclipse	29
20.5. Using Other IDEs than Eclipse	29
20.6. Git Documentation	29
21. How To Create an Internal Connector	32
21.1. Prerequisites	32
21.2. Creating a new connector project	32
21.3. Project Structure	33
21.4. Integrating the Connector into the OpenEngSB environment	34
22. How To Create an Internal Domain	35
22.1. Prerequisites	35
22.2. Creating a new domain project	35
22.3. Components	37
22.4. Connectors	38
23. Prepare and use Non-OSGi Artifacts	39
23.1. Create Wrapped Artifacts	39
23.2. Tips and Tricks	40
24. Release and Release Process	41
24.1. Releases and the OpenEngSB	41
24.2. Configure Maven	42
24.3. Adapt Jira	43
24.4. Perform the release	43
24.5. Spread the News	43
24.6. Prepare Changelog	43

25. Admin	45
25.1. Infrastructure	45
25.2. Logo Locations and Upgrade	46
V. Appendix	48
A. Java Coding Style	49
A.1. Sun Coding Guidelines	49
A.2. General	49
A.3. Naming	52
A.4. No clutter	52
A.5. Exception Handling	52
A.6. Tests	53
A.7. XML Formatting	53
26. Recommended Eclipse Plug-ins for Developers	55
26.1. Properties Editor	55
26.2. Spring IDE	55
26.3. Eclipse CS	55
26.4. Drools	55
B. Writing Documentation	56
B.1. General Documentation Guidelines	56
B.2. Document a domain or connector	56
B.3. Using Docbook	57
C. License	60

Part I. Introduction

This part 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 environment. 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. Nevertheless, setting 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 environment into your Apache Wicket application.

3.4. Simple Development and Distribution Management

While typical ESB have to be installed separately 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. 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 4. 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.

4.1. Writing new projects using the OpenEngSB

TBW

4.2. Writing Domains for the OpenEngSB

TBW

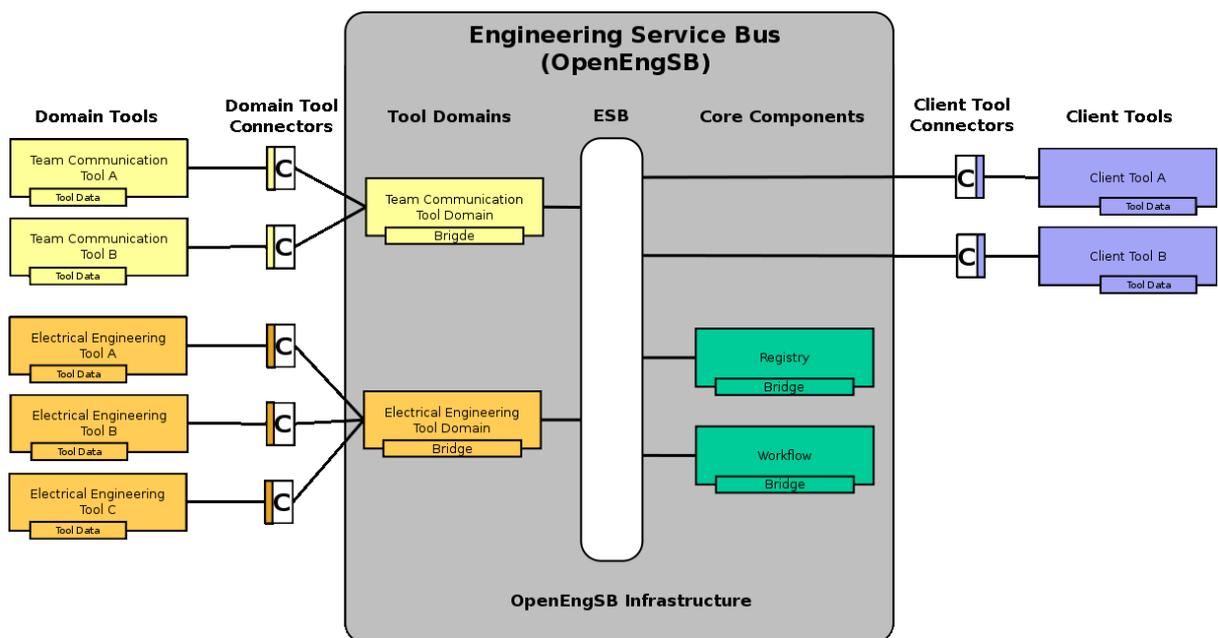
4.3. Writing Connectors for the OpenEngSB

TBW

Chapter 5. 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 5.1, “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 5.2, “OpenEngSB Infrastructure”. Within the OpenEngSB Infrastructure so called Section 5.3, “OpenEngSB Components” and Section 5.4, “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 5.5, “Client Tools (Service Consumer)” or Section 5.6, “Domain Tools (Service Provider)”, depending on their usage scenario. To integrate and use them within the OpenEngSB so called Section 5.7, “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

5.1. 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.

5.2. 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.

5.3. 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.

5.4. 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.

5.5. 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.

5.6. 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 technical differences between them they are described as two different component types.

5.7. Domain- and Client Tool Connectors

Tool Connectors connect 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 6. 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 leaves.

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 of the current context of a thread and the creation of new root context entries (which correspond to projects).

Chapter 7. 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 8. 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 8.3, "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.

8.1. 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.

8.2. 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.

8.3. 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 9. Taskbox

The Taskbox is a service which can be used when human interaction is required, e.g. by help desk applications. It consists of a core and an UI project. The core is responsible for storing tasks (via persistence), throwing events and starting workflows. Therefore it provides methods which can be called by workflows e.g. assigning a task to different user-roles (such as case worker or developer) or setting a task status.

It is also the job of the core taskbox to choose the right wicket panel from the UI project to display the right information in a certain situation. A wicket panel contains of a HTML-snippet which can be embedded into another HTML file, an underlying data model some logic like buttonlisteners and session handling.

So the idea is that an application which wants to use the taskbox only has to define an area in a wicket page where the taskbox is to be bound. The taskbox then takes control and takes user input to fill in the domain object behind which then gets stored again and used to decide how the workflow will go on. Based on the workflow and user interaction the taskbox then decides which panel is to be shown.

For each main action, the Taskbox throws an event. Examples for that are create, assign, finish or edit events. These events are used to trigger or resume workflows and they can also be recorded by another component which then can reconstruct the flow based on them.

The [Taskbox service](#) provides the methods to be called by workflows and to bind it to a UI. Take a closer look to explore its usage and possibilities.

Chapter 10. 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 multiple protocols. This section covers the examples in different languages and protocols, how such a thing can be achieved.

10.1. Proxying

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

10.1.1. Proxying internal Connector calls

ProxyConnector automatically exports a ServiceManager for every Domain to instantiate a proxy. An InvocationHandlerFactory has to be provided for proxying any call. The proxy has to be created via the normal instantiation mechanism on the website.

10.2. Using JMS proxying

The current JMS Connector allows for internal method calls being redirected via JMS, as well as Events being raised through JMS via an external source.

10.2.1. Proxying internal Connector calls

ProxyConnector automatically exports a ServiceManager for every Domain to instantiate a proxy. The proxy has to be created via the normal instantiation mechanism on the website. Whenever now a proxy method is called the call is marshalled and sent via JMS to a queue named `<DomainID>_method_send`. The marshalling is done via JSON. The mapping has the parameters type, which can be Call, Exception or Return, message, which in case of a method call is a simple serialisation of the arguments and name, which denotes the name of the method.

After sending the method call via JMS the proxy waits for a return at `<DomainID>_method_return`. The return message can use the same parameters as the send serialisation (type, name, message), but name is ignored. The message parameter is serialised to the correct return type if type is set to RETURN. If the type is Exception a new JMSException is thrown with the message.

By default a JMS Broker is started on port 6000.

10.2.2. Event handling via JMS

For every Domain found at the start of the OPENEngSB Server JMSConnector starts a listener on the `<DomainID>_event_send` queue. The parameters used are type and event. The type parameter is the fully qualified class name that has to be used to deserialise the event and be used as the argument to raiseEvent. After the correct class is loaded the content of the event parameter gets deserialised into an instance of the type parameter. The corresponding raiseEvent method is then called for the domain supported by this EventListener.

When the Event was processed a message is sent to the `<DomainID>_event_return` queue with the type set to RETURN and message set to OK. In case of Exception the type is set to exception and the message is set to the exception message.

10.2.3. Examples

10.2.3.1. Connect With Python

To test the OPENENGSB JMS implementation with Python please follow the [instructions](#)

The example can be downloaded [here](#)

10.2.3.2. 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](#)

10.2.3.3. 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 11. 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/yourproject
- 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 III. 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 12. Notification Domain

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

12.1. Description

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

12.2. Functional Interface

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

12.3. Connectors

12.3.1. Email Connector

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

12.3.1.1. External Tool Configuration

No external tool configuration is necessary.

Chapter 13. SCM Domain

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

13.1. 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.

13.2. Functional Interface

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

13.3. Connectors

13.3.1. Git Connector

The Git Connector is a SCM tool connector for the [Git fast version control system](#).

13.3.1.1. 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 14. Issue Domain

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

14.1. Description

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

14.2. Functional Interface

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

14.3. Connectors

14.3.1. Trac Connector

The Trac Connector is a issue tool connector for the [Trac project management and issue tracker system](#).

14.3.1.1. 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>).

Chapter 15. Report Domain

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

15.1. 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.

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. Plaintext Report Connector

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

15.3.1.1. External Tool Configuration

No external configuration is needed.

Chapter 16. Build Domain

The build domain is a domain for all build tools, like [Maven](#) or [Ant](#).

16.1. Description

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

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

This domain is implemented by the Section 19.1.1, “Maven Connector”, which supports multiple domains.

Chapter 17. Test Domain

The test domain is a domain for all test tools, like [Maven](#).

17.1. Description

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

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

This domain is implemented by the Section 19.1.1, “Maven Connector”, which supports multiple domains.

Chapter 18. Deploy Domain

The deploy domain is a domain for all deploy tools, like [Maven](#).

18.1. Description

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

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

This domain is implemented by the Section 19.1.1, “Maven Connector”, which supports multiple domains.

Chapter 19. Multi-Domain Connectors

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

19.1. Connectors

19.1.1. Maven Connector

The Maven Connector is a build, test and deploy tool connector for [Maven](#).

19.1.1.1. External Tool Configuration

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

Part IV. OpenEngSB

Committers & 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 20. Getting Started as a Developer

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

20.1. 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.

20.1.1. 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!

20.1.2. 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.

20.1.3. 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.

20.1.4. 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>.

20.1.5. 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.

20.2. 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.

20.2.1. 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](http://cygwin.com) or [msysgit](http://msysgit.com). 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.autocrlf input
```

20.2.2. 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.

20.3. 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
```

20.4. 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.

20.5. 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.

20.6. Git Documentation

20.6.1. 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](#).

20.6.2. 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.

20.6.3. 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).

20.6.4. Contributor Workflow

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

Please 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). Clone you're fork, but also add the original openengsb repository as remote repository. Please create a new branch named OPENENGSB-ISSUE_NUMBER_YOURE_WORKING_ON. Optionally append /DESCRIPTION. This is the OpenEngSB schema for naming branches and we'll really appreciate if you work according to it. In addition create the branch based on the origin/master oder origin/integration and not the branches of your fork. In this case you don't have to bother with updates of these branches in the OpenEngSB.

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. 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.

20.6.5. 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 committers 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.

20.6.6. 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.

Chapter 21. 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.

21.1. Prerequisites

In case it isn't known what a tool domain is and how it defines the interface for the tool connector then Section 5.4, "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 22, *How To Create an Internal Domain*.

21.2. 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, a shell script can be found at `etc/scripts/gen-connector.sh` for further help in creating a new connector project.

21.2.1. Using the Maven Archetype

It is not recommended to use the maven archetype directly, because the `gen-connector.sh` script executes additional tasks, like the renaming of the resulting project. Furthermore the shell script 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. Do not forget to check that the new connector is included in the modules section of the domain parent pom.

21.2.2. Using the `gen-connector.sh` shell script

Calling the script should be done from the domain-specific directory. I.e. if your are developing a new connector for the Notification-Domain your current directory should be `domains/notification`. Inside your favourite shell execute the script.

```
notification $ ../../etc/scripts/gen-connector.sh
```

The script tries to guess as much as possible from your current location and previous input. Gussed 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 `domains/notification` directory:

```
Domain Name (is notification): <Enter>
Domain Interface (is NotificationDomain): <Enter>
Connector Name: twitter <Enter>
Version (is 1.0.0-SNAPSHOT): <Enter>
Project Name (is OpenEngSB :: Domains :: Notification :: Twitter): <Enter>
```

Only the connector name was set, everything else has been guessed correctly by the script. 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 script. If the script finishes successfully the new connector project has been created and you may start implementing.

21.3. Project Structure

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

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


Chapter 22. 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 5, *Architecture of the OpenEngSB*.

22.1. Prerequisites

In case it isn't known what a domain is and how it defines the interface and events for connectors, then Section 5.4, "OpenEngSB Tool Domains" is a good starting point.

22.2. 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, a shell script can be found at `etc/scripts/gen-domain.sh` as further convenience.

22.2.1. Using the Maven Archetype

It is not recommended to use the maven archetype directly, because the `gen-domain.sh` script executes additional tasks, like the renaming of the resulting project. Furthermore the shell script 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`.

22.2.2. Using the `gen-domain.sh` shell script

The script should be executed from the `domains` directory in your OpenEngSB repository.

```
domains $ ../etc/scripts/gen-domain.sh
```

You'll be asked to fill in a few variables the script needs to create the initial project structure. Based on your input, the script tries to guess further values. Gussed 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 (is mydomain): test <Enter>
Version (is 1.0.0-SNAPSHOT): <Enter>
Prefix for project names (is OpenEngSB :: Domains :: Test): <Enter>
```

Only the domain name has been filled in, while the rest has been correctly guessed by the script. 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 by the script. If the script 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.

22.2.3. 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
|     -- 110n
|         -- 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.

22.3. 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.

22.4. Connectors

For information regarding the implementation of connectors for the newly created domain see Chapter 21, *How To Create an Internal Connector*.

Chapter 23. 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.

23.1. 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.bnd` 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
```

23.2. 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 24. 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.

24.1. 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).

24.2. 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>OPENENGSB_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](#)).

24.3. 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. For simplicity we provide for all release candidates of 1.0.0 only one 1.0.0.RCX release target. This is closed with the final release and replaced by 1.0.X.RELEASE target. This is replaced by 1.1.X.RELEASE. We don't know for the moment how we should handle older releases, but for the moment the OpenEngSB team simply do not have the man power to support older releases.

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.
- If you release a final release (MAJOR RELEASE) create a new release target 1.0.X.RELEASE.
- If you release a minor release do nothing with Jira.

24.4. 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`

24.5. 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
```

24.6. Prepare Changelog

Finally 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
-----
### Bug fixes
```

```
* Bugfix

### Changed Components
* New Projects
  * openengsb-core-taskbox
* Removed Projects

### New Features & Changed Behaviour
* Feature

### Deprecated or removed Features
* interface xyz
```

Chapter 25. 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.

25.1. Infrastructure

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

25.1.1. 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

25.1.2. 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.

25.1.3. 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.

25.1.4. 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.

25.1.5. OpenEngSB Maven

25.1.5.1. internal

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

25.1.5.2. external

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

25.1.6. 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

25.2. 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, reduced to 256x256 and converted to .ico. In unix systems `convert openengsb.png openengsb.ico` in the command line should do the job.

25.2.1. 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).

25.2.2. 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`)

25.2.3. 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.

Part V. Appendix

Appendix A. Java Coding Style

A.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.

A.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.

A.1.2. Wrapping

Use the auto-formatter of your IDE. Import the [Eclipse Formatter file](#).

A.1.3. Number of declarations per line

Only one declaration per line is allowed.

A.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.

A.1.5. Blank lines

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

A.2. General

A.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.

A.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.

```

```
*/
```

A.2.3. Duplication

Code duplication has to be avoided at all costs.

A.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
}

```

A.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.

A.2.6. Use enums

Prefer typesafe enumerations over integer constants.

A.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.

A.2.8. Use fully qualified imports

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

A.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.

A.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.

A.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.

A.2.12. Use String.format()

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

A.2.13. Array declaration style

Always use

```
Type[] arrayName;
```

instead of the C-like

```
Type arrayName[];
```

A.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.

A.3. Naming

A.3.1. Interfaces

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

A.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.

A.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.

A.5. Exception Handling

- Don't log and throw. Either an 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.

A.6. Tests

A.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()`.

A.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`

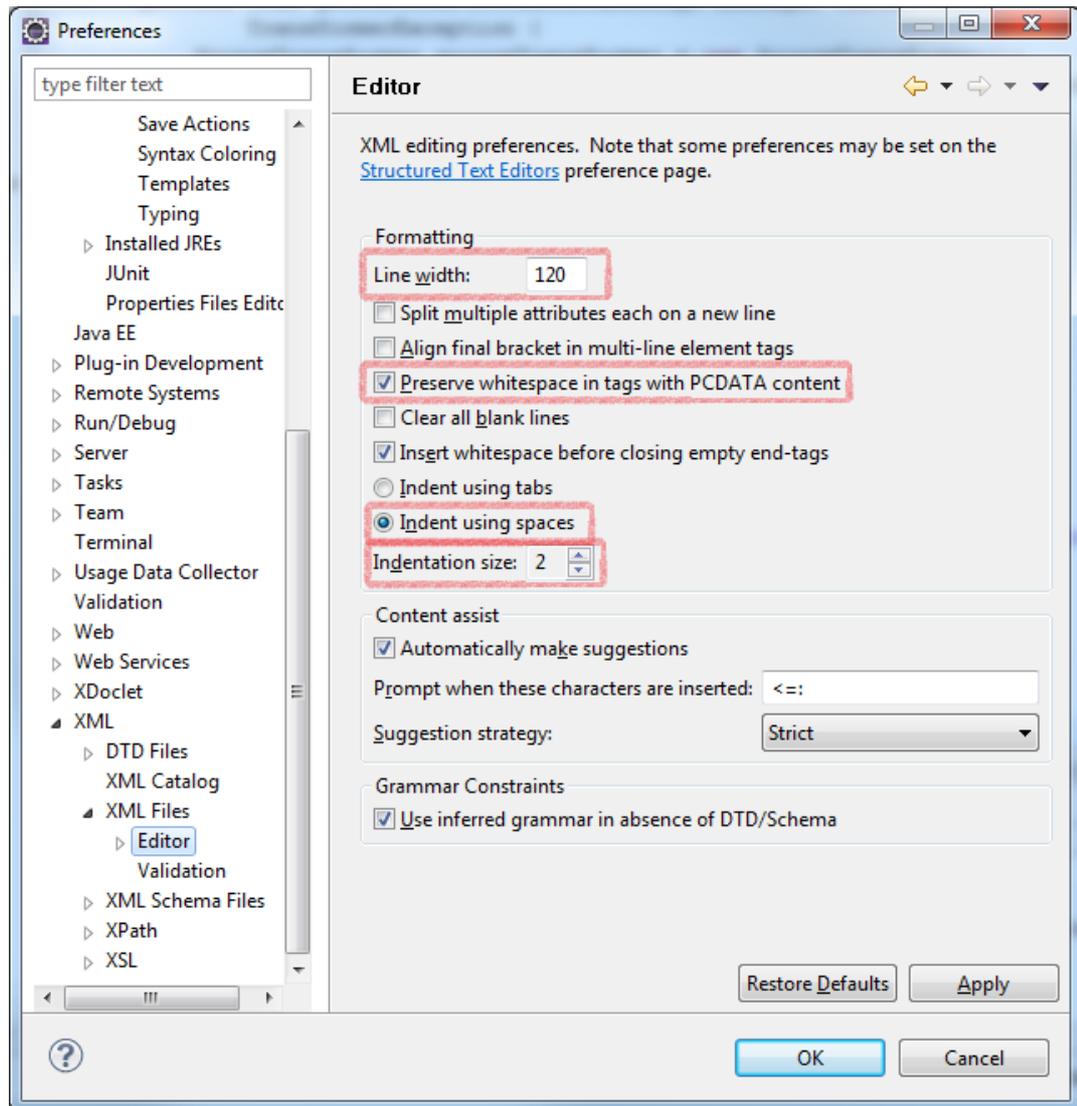
A.7. XML Formatting

A.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.

A.7.2. Eclipse Settings

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



Eclipse XML Settings

A.7.3. Recommended Readings

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

Chapter 26. 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.

26.1. 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".

26.2. 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.

26.3. 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.

26.4. 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.

Appendix B. 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.

B.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 abbreviations 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.

B.2. Document a domain or connector

B.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.

B.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 consist 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.

B.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](#).

B.3.1. Tags

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

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

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

B.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>

```

B.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>

```

B.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.

Appendix C. License

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition,

"submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their

Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole,

provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only

on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

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.