ZARS

Developer's Guide

© soomsam.net. This document is subject to the terms and conditions outlined in the [CC BY-NC-SA 3.0](http://creativecommons.org/licenses/by-nc-sa/3.0/) license.

|  |  |
| --- | --- |
| **Project** | ZARS |
| **Version** | 0.1 |
| **Status** | Draft |
| **Authors** | Liebmann, Erich |
| **Reviewers** | - |
| **Last Updated** | 2012-01-29 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Change Log** | | | |
| **Version** | **Author** | **Date** | **Comment** |
| 0.1 | Liebmann, Erich | 2012-01-29 | initial version |

Table of Contents

[1. Purpose and Scope 4](#_Toc315883899)

[1.1. Notes 4](#_Toc315883900)

[1.2. Conventions 4](#_Toc315883901)

[1.3. Prerequisites 4](#_Toc315883902)

[2. Development Environment Setup 5](#_Toc315883903)

[2.1. Setup Eclipse Installation 5](#_Toc315883904)

[2.1.1. Specify the JVM for Eclipse 5](#_Toc315883905)

[2.1.2. Install Suggested Plug-ins 5](#_Toc315883906)

[3. Eclipse Workspace Setup 7](#_Toc315883907)

[3.1. Checkout ZARS 7](#_Toc315883908)

[3.2. Building and Starting ZARS using Eclipse 7](#_Toc315883909)

[3.3. Building and Starting ZARS using Maven 7](#_Toc315883910)

[4. Building, Packaging, and Releasing 9](#_Toc315883911)

[4.1. Environment Specific Settings 9](#_Toc315883912)

[4.2. Build and Package in Eclipse 9](#_Toc315883913)

[4.3. Build and Package on the Command Line 9](#_Toc315883914)

[4.4. Releasing 9](#_Toc315883915)

[5. Setup and Installation 11](#_Toc315883916)

[5.1. Database Setup 11](#_Toc315883917)

[5.1.1. HyperSQL 11](#_Toc315883918)

[5.1.2. MySQL 11](#_Toc315883919)

[5.1.3. PostgreSQL 12](#_Toc315883920)

[5.2. Servlet Container Setup 12](#_Toc315883921)

[5.2.1. Jetty 12](#_Toc315883922)

[5.2.2. Tomcat 12](#_Toc315883923)

[5.3. Deployment 13](#_Toc315883924)

[5.4. Management and Monitoring 13](#_Toc315883925)

[6. Testing 14](#_Toc315883926)

[6.1. Unit Testing 14](#_Toc315883927)

[6.2. Integration Testing 14](#_Toc315883928)

[6.3. Acceptance Testing 14](#_Toc315883929)

[7. Development Guidelines 15](#_Toc315883930)

[7.1. System Architecture 15](#_Toc315883931)

[7.2. Software Architecture 15](#_Toc315883932)

[7.3. Best Practices 18](#_Toc315883933)

[7.3.1. Quality Assurance 18](#_Toc315883934)

[7.3.2. Source Code Commit 18](#_Toc315883935)

[7.3.3. Source Code Comments 19](#_Toc315883936)

[7.3.4. Serialization 20](#_Toc315883937)

[7.3.5. Stability and Capacity 19](#_Toc315883938)

[7.3.6. Startup and Shutdown 19](#_Toc315883939)

[7.3.7. Infrastructure Services 19](#_Toc315883940)

[7.4. Coding Guidelines 20](#_Toc315883941)

# Purpose and Scope

This Developer's Guide will help to get you started developing and maintaining ZARS. It illustrates (a) how to setup the development environment, (b) how to start and stop ZARS, (c) how to unit, integration, and acceptance test the application, and (d) how to create a dev, test, uat, and prod build of ZARS.

## Notes

This guide assumes that you are going to use the Eclipse IDE to develop and maintain ZARS. All instructions are given based on *Eclipse Indigo (3.7) Java EE IDE for Web Developers* and may differ in earlier / later / different versions of the Eclipse IDE.

**Note**All paths specified in this guide are relative to the Eclipse workspace of ZARS.

## Conventions

This guide uses the typographic conventions outlined in table .

|  |  |
| --- | --- |
| **Typographic Style** | **Usage** |
| *italic* | Used to emphasize single words or parts of a sentence, to introduce product names, and new terms where they are defined. |
| underline | Used for cross-references in the document. |
| dotted-underline | A special term defined in appendix . |
| constant width | Used for GUI labels / text, keywords, operators, datatypes, method names, variable names, class names, and interface names. |
| # | Indicates an Operating System prompt. |
| > | Indicates the prompt of a command line tool (e.g. RDBMS query tool) |

Table Typographic Conventions

## Prerequisites

To develop and maintain ZARS the following software has be installed:

* Oracle / Sun JDK 1.6
* Eclipse Indigo[[1]](#footnote-2) (3.7)

To facilitate development it is recommended to install the following Eclipse plug-ins:

* Subversive[[2]](#footnote-3)
* Eclipse m2e
* Spring IDE
* FindBugs (recommended)
* PMD (recommended)
* EclEmma/JaCoCo (optional)

# Development Environment Setup

## Setup Eclipse Installation

After downloading and unpacking [Eclipse](http://www.eclipse.org/) perform the steps outlined below to configure the IDE and to install all recommended plug-ins.

### Specify the JVM for Eclipse

Perform the following steps to get Eclipse to run on the (correct version of the) JDK:

* open the [eclipse.ini file](http://wiki.eclipse.org/Eclipse.ini) (located in the Eclipse installation directory)
* [specify the -vm parameter](http://wiki.eclipse.org/Eclipse.ini#Specifying_the_JVM) with the correct path to your JDK installation

**Note**Setting the -vm option in the eclipse.ini file will ensure seamless operation of the Eclipse Maven Integration (Eclipse m2e) and define the Java installation used by Maven plug-ins.

### Install Suggested Plug-ins

#### Install m2e and Subversive

Perform the following steps to install the m2e and Subversive plug-ins:

* navigate to Help > Install New Software...
* select Indigo - http://download.eclipse.org/releases/indigo from the Work with dropdown box
* expand section Collaboration and select m2e - Maven Integration for Eclipse and Subversion SVN Team Provider
* proceed through the wizard to install the plug-ins and restart Eclipse when prompted
* navigate to Window > Open Perspective > Other... and select SVN Repository Exploring to trigger Subversive Connector Discovery
* in the Subversive Connector Discovery dialog select the latest version of SVN Kit and the latest version of Native JavaHL
* confirm the selection in the dialog, proceed through the wizard, and accept any security warnings to install the selected Subversive Connectors, restart Eclipse when prompted

#### Install Spring IDE

Perform the following steps to install the Spring IDE plug-in:

* navigate to Help > Install New Software...
* specify http://dist.springsource.com/release/TOOLS/update/2.8.1.RELEASE/e3.7 in the Work with dropdown box
* expand section Core / Spring IDE and select Spring IDE Core
* proceed through the wizard to install the plug-in and restart Eclipse when prompted

#### Install FindBugs

Perform the following steps to install the FindBugs plug-in:

* navigate to Help > Install New Software...
* specify http://findbugs.cs.umd.edu/eclipse in the Work with dropdown box
* expand section FindBugs and select FindBugs Feature
* proceed through the wizard and accept any security warnings to install the plug-in, restart Eclipse when prompted

#### Install PMD

Perform the following steps to install the PMD plug-in:

* navigate to Help > Install New Software...
* specify http://pmd.sf.net/eclipse in the Work with dropdown box
* expand section PMD for Eclipse 3 and select PMD Plug-in
* proceed through the wizard and accept any security warnings to install the plug-in, restart Eclipse when prompted

#### Install EclEmma/JaCoCo

Perform the following steps to install the EclEmma plug-in:

* navigate to Help > Install New Software...
* specify http://update.eclemma.org in the Work with dropdown box
* expand section EclEmma and select EclEmma Java Code Coverage
* proceed through the wizard and accept any security warnings to install the plug-in, restart Eclipse when prompted

**Hint**  
Setting up a new Eclipse workspace for ZARS? Why don't you try a new / different [programming font](http://www.google.com.sg/search?q=best+programming+font)? When it comes to programming fonts tastes and preferences definitely vary but I personally like Droid Sans Mono, Deja Vu Sans Mono, and Lucida Console.

# Eclipse Workspace Setup

## Checkout ZARS

The ZARS source code is managed in an Apache Subversion (SVN) repository. To start developing / maintaining ZARS checkout the project from the following SVN repository:

**ZARS SVN Repository URL**  
https://svn.soomsam.net/zirmegghuette/

Perform the following steps to checkout ZARS into your Eclipse Workspace:

* open the SVN Repository Exploring perspective
* right-click on an empty spot (the background) in the SVN Repositories panel and select New > Repository Location...
* specify (above mentioned) *ZARS SVN Repository URL* as the URL, set User and Password to your SVN credentials, tick Save authentication (could trigger secure storage login), and press Finish to confirm
* navigate to and expand trunk in the ZARS SVN Repository, right-click on zars and select Check Out to start the SVN checkout

After the SVN checkout is completed Eclipse should automatically start to build the project. The first build could run for quite a while since Maven might have to download a number of dependencies.

**Note**  
Please note that the project has been setup with [separate Maven and Eclipse output folders](http://wiki.eclipse.org/M2E_FAQ#How_to_configure_Maven_project_to_use_separate_output_folders_in_Eclipse). The Maven build uses output folder target/ while the Eclipse build uses output folder target-eclipse/.

## Building and Starting ZARS using Eclipse

Perform the following steps to *build ZARS with Eclipse during development*:

* navigate to Project > Clean...
* select Clean all projects

*or alternatively*

* select Clean projects selected below and tick zars.

Perform the following steps to *start ZARS with Eclipse during development*:

* navigate to Run > Run Configurations...
* expand Maven Build, select the pre-configured Run Configuration eclipse-jetty-run and execute it

**Note**ZARS will be started in an embedded Jetty instance using the Eclipse output folder. Modifications to source code files will be detected automatically and will result in a redeployment of ZARS. The in- memory DB [HyperSQL](http://hsqldb.org/) is used in this execution mode. All modifications to the DB are lost upon shutdown / restart of Jetty.

## Building and Starting ZARS using Maven

Perform the following steps to *build and start ZARS with Maven during development*:

* navigate to Run > Run Configurations...
* expand Maven Build, select the pre-configured Run Configuration maven-jetty-run and execute it

**Note**ZARS will be started in an embedded Jetty instance using the Maven output folder. Modifications to source code files will *not* be detected automatically. The in-memory DB [HyperSQL](http://hsqldb.org/) is used in this execution mode. All modifications to the DB are lost upon shutdown / restart of Jetty.

## Hot/Redeploying ZARS

TODO

# Building, Packaging, and Releasing

## Environment Specific Settings

ZARS uses [Maven profiles](http://maven.apache.org/guides/introduction/introduction-to-profiles.html) to customize the build and package process for each target environment. All target environment specific settings and configuration options are maintained in a dedicated folder under src/config/webapp/.

An overview of all supported Maven profiles / target environments is provided in table .

|  |  |  |  |
| --- | --- | --- | --- |
| **Maven Profile[[3]](#footnote-4)** | **RDBMS** | **Target Environment Specific Settings** | **Tests** |
| **dev** | HyperSQL | src/config/webapp/dev/ | executed |
| test-mysql | MySQL | src/config/webapp/test-mysql/ | executed |
| test-pgsql | PostgreSQL | src/config/webapp/test-pgsql/ | executed |
| uat | MySQL | src/config/webapp/test-uat/ | not executed |
| prod | MySQL | src/config/webapp/test-prod/ | not executed |

Table Target Environment Configurations

**Note**  
The Maven profile dev is activated by default in Eclipse.

**Note**Each Maven profile contains deployment descriptors / configuration files for all supported Servlet Containers / Application Servers.

**Note**  
The goal is to keep ZARS as independent of the DB and Servlet Container / Application Server as possible.

## Build and Package in Eclipse

Perform the following steps to build and package ZARS in Eclipse:

* execute the pre-configure (Maven Build) Run Configuration package-<PROFILE>

**Note**  
The resulting WAR file zars-<PROFILE>-<VERSION>.war will be placed in the output folder target/.

## Build and Package on the Command Line

Perform the following steps to build and package ZARS on the command line:

* zars# **mvn -P<PROFILE> clean package**

**Note**  
The resulting WAR file zars-<PROFILE>-<VERSION>.war will be placed in the output folder target/.

## Releasing

Perform the following steps to create a UAT or production release of ZARS:

* ensure all changes have been committed to SVN
* ensure that the workspace is fully in sync with SVN
* edit the Maven project file pom.xml file, set / increase the project version, and commit the changes to SVN
* create a new tag zars-<VERSION> from the Head revision of trunk
* create a uat or prod build of ZARS (in Eclipse or on the command line)

# Setup and Installation

## Database Setup

Depending on the selected Maven profile[[4]](#footnote-5) ZARS is configured to either use HyperSQL, PostgreSQL, or MySQL as the database backend. Please find an overview of the configured database for each target environment in table .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Maven Profile[[5]](#footnote-6)** | **RDBMS** | **Version** | **Database** | **Username** | **Password** |
| **dev** | HyperSQL | 1.8.0.10+ | zars\_dev | sa |  |
| test-mysql | MySQL | 5.1.47+ | zars\_test | zars\_test | zars\_test |
| test-pgsql | PostgreSQL | 8.4.4-1+ | zars\_test | zars\_test |  |
| uat | MySQL | 5.1.47+ | e\_zars\_uat | zars\_uat | zars\_uat |
| prod | MySQL | 5.1.47+ | e\_zars | zars | zars |

Table Database Environments

### HyperSQL

No specific setup is required for [HyperSQL](http://hsqldb.org/), the database zars\_dev will be created / setup automatically.

### MySQL

Install[[6]](#footnote-7) [*MySQL*](http://www.mysql.com/) *Community Server*, perform the following steps to start and stop the RDBMS, and to setup or teardown the database objects for ZARS. Please use the database name, username, and password for the respective target environment as indicated in table .

#### Start RDBMS

* # mysqld

#### Shutdown RDBMS

* # mysqladmin.exe -u root -p shutdown

#### Setup Database

* # mysql -u root -p  
  > create database <DATABASE>;  
  > alter database <DATABASE> charset=utf8;  
  > create user '<USERNAME>'@'localhost' identified by '<PASSWORD>';  
  > grant all on <DATABASE>.\* to '<USERNAME>'@'localhost';  
  > quit;
* # mysql -u <USERNAME> -p  
  > show databases;  
  > quit;

#### Teardown Database

* # mysql -u root -p  
  > revoke all on <DATABASE>.\* from '<USERNAME>'@'localhost';  
  > drop database <DATABASE>;  
  > drop user '<USERNAME>'@'localhost';

### PostgreSQL

Install[[7]](#footnote-8) [PostgreSQL](http://www.postgresql.org/), perform the following steps to setup, start, and stop the RDBMS, and to setup or teardown the database objects for ZARS. Please use the database name, username, and password for the respective target environment as indicated in table .

#### Setup RDBMS

* # initdb -E UTF8 -D <DATA\_DIR>

#### Start RDBMS

* # postgres -D <DATA\_DIR>

#### Shutdown RDBMS

* # pg\_ctl.exe stop -D <DATA\_DIR>

#### Setup Database

* # createuser root
* # createuser zars
* # createdb -O zars -E UTF8 zars

#### Teardown Database

* # dropdb zars

## Servlet Container Setup

Each Maven profile contains deployment descriptors / configuration files for all supported Servlet Containers / Application Servers. Please find an overview of the supported Servlet Containser / Application Servers in table .

|  |  |  |
| --- | --- | --- |
| **Servlet Container** | **Version** | **Note** |
| Jetty | 6.1.26+ | embedded or standalone |
| Tomcat | 7.0.6+ | embedded or standalone |

Table Supported Servlet Containers

### Jetty

No specific setup is required for [Jetty](http://jetty.codehaus.org/), all infrastructure services used by ZARS are defined and configured in the application itself.

### Tomcat

No specific setup is required for [Tomcat](http://tomcat.apache.org/), all infrastructure services used by ZARS are defined in the application itself.

## Deployment

No specific instructions are required to deploy ZARS. Follow the instructions of the Servlet Container or Application Server used to deploy the ZARS WAR.

**Note**The persistence framework used has been setup to automatically perform all schema modifications. Under normal circumstances it should, therefore, not be necessary to run any SQL DDL scripts before or after deployment. To migrate data it can, however, be necessary to prepare a SQL DML script which is executed before and /or after the deployment.

**Note**  
It is recommended to perform a health check after each deployment to UAT and production. As part of the health check it should be verified that there are no exceptions or errors in the logs and that it is possible to login and logout of ZARS.

## Management and Monitoring

Ideally JMX should be used to manage and monitor ZARS in UAT and production. Since ZARS is deployed on a public server directly connected to the Internet access to ports other than HTTP and HTTPS have been restricted for security reasons.

External monitoring (provided by the hosting provider) has been setup for ZARS and the server ZARS is deployed to. All application level management and monitoring capabilities should be built directly into the admin section of the web interface.

# Testing

For us developers' manual, monotonous, and repetitive work feels particularly mind numbing, boring, and tedious. Yet, surprisingly, many of us do some seemingly small tasks over and over again to test our code, to verify the performance characteristics of our application, and to determine the scalability of our platform.

Writing reliable, readable, and sustainable automated tests that cover all aspects of an application is a challenging task. This is especially true when it comes to testing persistence, threads, and asynchronous code. But as developers don't we strive to solve challenging and interesting coding tasks? Why do we treat the source code that makes up automated tests any different from the source code that makes up the application? Why don't we treat coding automated tests as an integral part of the application coding and development experience?

In fact, designing flexible automated tests that reliably test all aspects of an application by constructing complex test data in a reusable way while simplify test diagnostics is a very interesting and rewarding task. Often more so than assembling the actual application out of all these pre-packaged frameworks we usually end up chaining together.

As you might suspect from this introduction, particular focus is on testing ZARS. You might want to pick up the books [Growing Object-Oriented Software, Guided By Tests](http://www.amazon.com/Growing-Object-Oriented-Software-Guided-Tests/dp/0321503627) and [Design Driven Testing](http://www.amazon.com/Design-Driven-Testing-Smarter-Harder/dp/1430229438) to help you get started writing sustainable automated tests.

Automated tests for ZARS are organized into three distinct categories, namely unit tests, integration tests, and acceptance tests.

## Unit Testing

TODO [JDave](http://jdave.org/), [Mockito](http://code.google.com/p/mockito/), [PowerMock](http://code.google.com/p/powermock/), [Spring Unit Testing Support](http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/testing.html#unit-testing)

## Integration Testing

TODO

Backend: [Spring Integration Testing Support](http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/testing.html#integration-testing)

Frontend: [JSFUnit](http://www.jboss.org/jsfunit)

## Acceptance Testing

TODO [Thucydides](http://www.wakaleo.com/resources/thucydides-automated-testing-library), [JSFUnit](http://www.jboss.org/jsfunit)

# Development Guidelines

## System Architecture

The UML deployment diagram in figure depicts the system architecture of ZARS in production. The ZARS components outlined in the diagram are:

* **ZARS VHost**  
  The Apache Virtual Host and **mod\_jk** configuration delegating all requests to the Apache Tomcat server.
* **ZARS WebApp**  
  The Web Application containing all software layers described in the next section.
* **ZARS Database**The database / schema used to store all ZARS data.

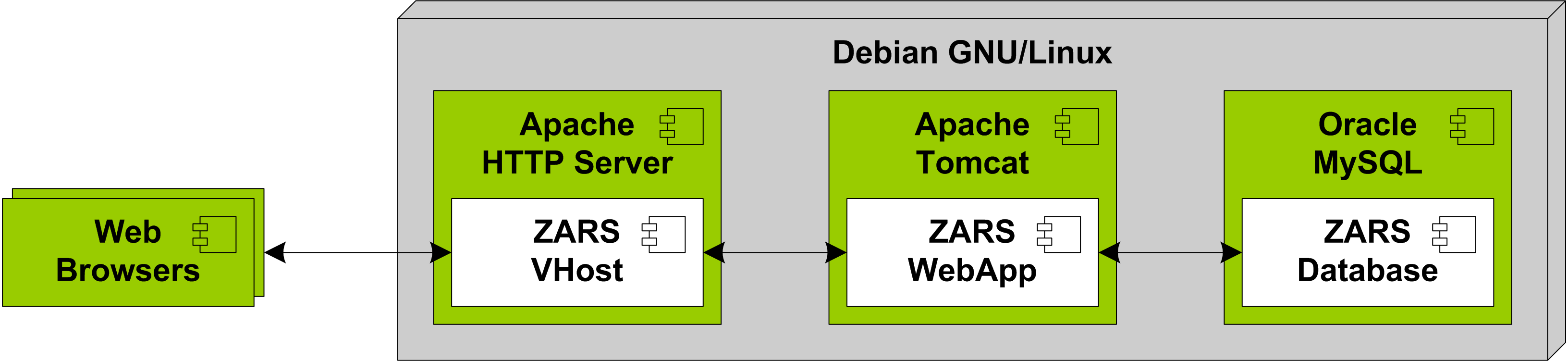


Figure System Architecture

## Software Architecture

The diagram in figure provides an overview of the software architecture of ZARS.

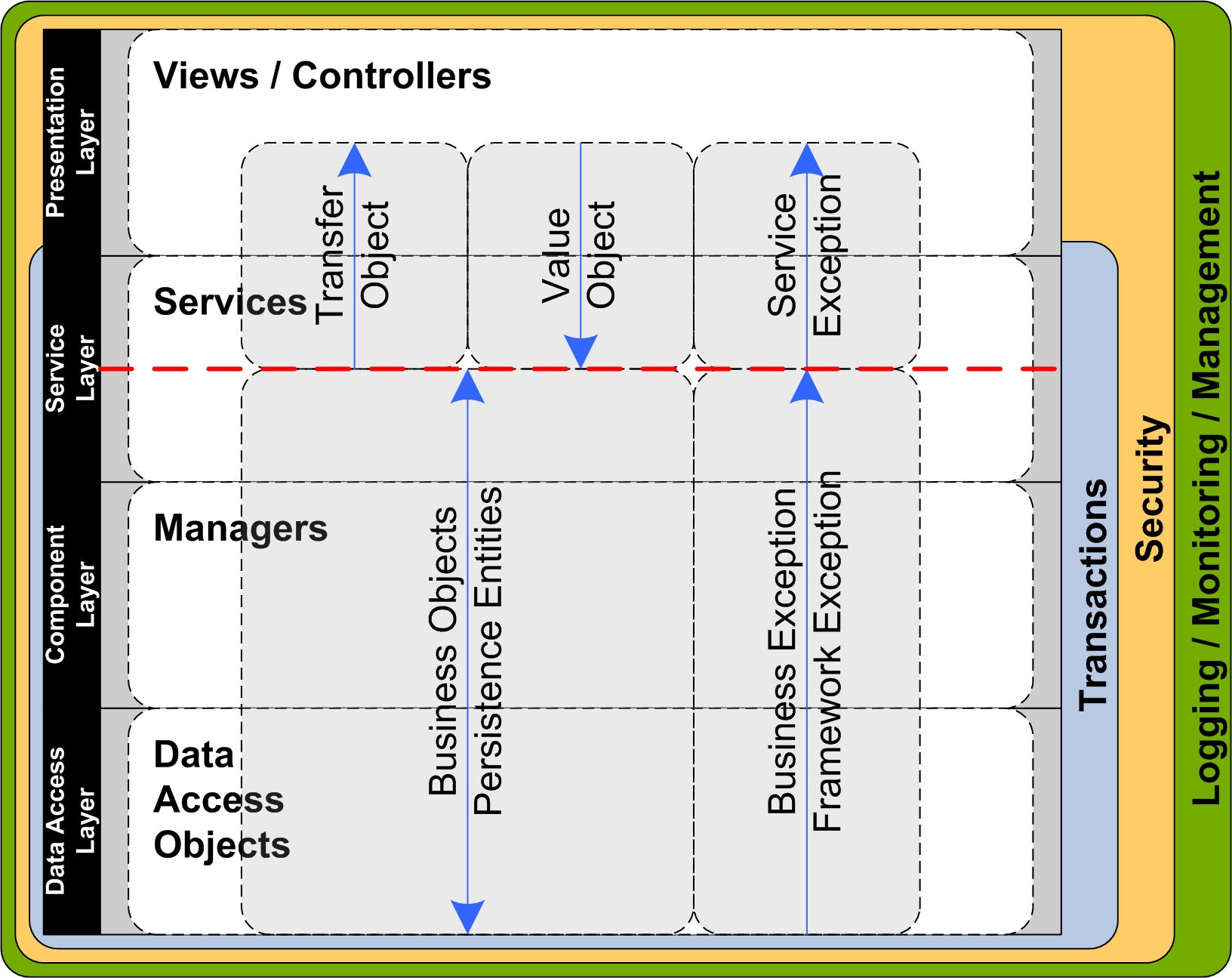


Figure Software Architecture

ZARS consists of the following software layers (structural and logical components):

* **Presentation Layer**  
  The Presentation Layer consists of (1) controllers, (2) view definitions and the (3) data model exchanged with the Service Layer in form of Transfer Objects and Value Objects. Please note that the Persistence Layer only ever interacts with the Service Layer and with no other layer of the software architecture.
* **Service Layer**  
  The Service Layer consists of the service implementations and acts as the transaction demarcation boundary. The Service Layer maps between the internal (Business Objects, Persistence Objects, Business Exceptions, and Framework Exceptions) and external (Transfer Object, Value Object, Service Exception) data representation highlighted by a red line in the diagram.  
  Usually service implementations rely on one or more managers to implement functionality. In rare cases it can, however, be necessary to call services from other services. This kind of service orchestration should be the exception rather than the norm.
* **Component Layer**  
  The Component Layer consists of manager implementations that implement the actual business logic and business workflows. Managers operate directly on Business and Persistence Objects and do *not* perform any translation of business or framework exception. This effectively makes reusing logic implemented in other managers efficient and straight forward and is advocated.
* **Data Access Layer**The Data Access Layer uses a persistence framework to map data from the relational database to persistence entities. The Data Access Layer is also the place where all database queries and bulk operations are implemented.

ZARS relies on the following data / object models:

* **Transfer Objects**Transfer Objects are used to send data from the Service Layer to the Presentation Layer. They should be designed as generic as possible and intentionally contain all properties of the backing business objects and persistence entities even if they are currently not required for the use case at hand. Transfer Objects should confirm to the Java Bean specification.  
  If required, Transfer Objects can be nested in other Transfer objects effectively creating a hierarchical data model for presentation needs. In rare cases it can be necessary to create use case specific Transfer Objects hierarchies that bundle the data in specific ways.
* **Value Objects**  
  In general it is preferred to use primitive types as input parameters for all service methods. For complex input data, especially when Sets, Lists, or Maps of complex structures have to be transferred back to the service layer the creation of a dedicated Value Object is suggested. Make Value Objects as specific as possible by creating dedicated versions for create, update, or delete operations.
* **Service Exceptions**  
  All exceptions thrown in the service layer should be part of a single service exception hierarchy. All exceptions propagating up from lower levels should be converted into or wrapped by a service exception. To guarantee that all kind of expected and unexpected service exceptions are always translated into a service exception an around advice applying to all service methods should be put in place.  
  To avoid ClassNotFoundExceptions for unknown exception classes nested in service exceptions for potential remote clients it is suggested to include the entire stack trace of the cause as a String not the original Exception.
* **Persistence Entities**All tables in the ZARS database are mapped to persistence entities. Avoid the [anemic domain model](http://en.wikipedia.org/wiki/Anemic_domain_model) anti-pattern and feel free to implement business logic directly in the persistence entities.
* **Business Objects**  
  Business Objects are objects used in addition to persistence entities to implement business logic, business workflows, and pass data back and forth between layers.
* **Business and Framework Exceptions**  
  All exceptions defined in the Data Access Layer or Component Layer to communicate errors and all exceptions thrown by third party frameworks.

**Note**  
We will use the term *Domain Objects* to refer to Persistence Entities *and* Business Objects.

**Note**We will use the term *Application Exceptions* to refer to Business Exceptions *and* Framework Exceptions.

Please refer to table and for a tabular view of the software layer access and data model usage policies outlined in diagram .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Views / Controllers** | **Services** | **Managers** | **Data Access Objects** |
| **Views / Controllers** |  |  |  |  |
| **Services** |  |  |  |  |
| **Managers** |  |  |  |  |
| **Data Access Objects** |  |  |  |  |

Table Software Layer Usage

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Views / Controllers** | **Services** | **Managers** | **Data Access Objects** |
| **Transfer Objects** |  |  |  |  |
| **Value Objects** |  |  |  |  |
| **Service Exceptions** |  |  |  |  |
| **Domain Objects** |  |  |  |  |
| **Application Exceptions** |  |  |  |  |

Table Data Model Usage

A mapping of the software artifacts to Java packages can be found in table .

|  |  |
| --- | --- |
| **Software Architecture Item** | **Java Source Code Package** |
| Controllers | net.soomsam.zirmegghuette.zars.web.controller |
| Services | net.soomsam.zirmegghuette.zars.service.\* |
| Managers | net.soomsam.zirmegghuette.zars.manager.\* |
| Data Access Objects | net.soomsam.zirmegghuette.zars.persistence.dao |
| Transfer Objects | net.soomsam.zirmegghuette.zars.service.to |
| Value Objects | net.soomsam.zirmegghuette.zars.service.vo |
| Service Exceptions | net.soomsam.zirmegghuette.zars.service.\* |
| Persistence Entities | net.soomsam.zirmegghuette.zars.persistence.entity |
| Business Objects | net.soomsam.zirmegghuette.zars.service.\*  net.soomsam.zirmegghuette.zars.manager.\*  net.soomsam.zirmegghuette.zars.persistence.dao |
| Business Exceptions | net.soomsam.zirmegghuette.zars.service.\*  net.soomsam.zirmegghuette.zars.manager.\*  net.soomsam.zirmegghuette.zars.persistence.dao  net.soomsam.zirmegghuette.zars.persistence.entity |

Table Java Package Overview

Due to the size of the application, individual software artifacts have *not* been defined as separate Maven modules. The software architecture is defined solely by the Java package conventions outlined in table . A split by function / service (such as a separate user, reservation, and billing module) has *not* been addressed due to the size of the application but would make for an interesting test case.

A carefully designed package structure is crucial since most structural code analysis tools and dependency checkers analyze Java projects at the package level. A static code analysis tool such as [Headway Structure 101](http://www.headwaysoftware.com/products/?code=Structure101), [Lattix LDM/LDV](http://www.lattix.com/products/ldm-ldv), [hello2morrow Sonargraph](http://www.hello2morrow.com/products/sonargraph), or [SonarSource](http://www.sonarsource.org/) will be used to review dependencies and linkage and used as the basis to improving the code structure.

**Note**  
[ClassCycle](http://classycle.sourceforge.net/) and/or [Macker](http://innig.net/macker/) will be used to automatically check the conventions and architectural rules outlined above.

TODO consider using annotations to identifying classes belonging to particular design patterns and to associate classes with components, modules, or layers in an application

## Best Practices

### Quality Assurance

The static source code analysis tools [FindBugs](http://findbugs.sourceforge.net/) and [PMD](http://pmd.sourceforge.net/) will be used *with project specific settings* to ensure a certain level of code quality. The software architecture outlined in will be validated with [ClassCycle](http://classycle.sourceforge.net/) and/or [Macker](http://innig.net/macker/).

A manual review of source code dependencies and linkage across ZARS will be performed with one of the static code analysis tool such as [Headway Structure 101](http://www.headwaysoftware.com/products/?code=Structure101), [Lattix LDM/LDV](http://www.lattix.com/products/ldm-ldv), [hello2morrow Sonargraph](http://www.hello2morrow.com/products/sonargraph), or [SonarSource](http://www.sonarsource.org/).

### Source Code Commit

When committing source code to the ZARS project ensure that the following best practices are followed:

* **ensure to commit only Java source code *without* errors and warnings**  
  Try to avoid writing Java code that is marked with a warning in Eclipse, if it can't be avoided suppress the warning with @SuppressWarnings
* **ensure to supply a meaningful description of your changes in the commit message**  
  Never commit with an empty commit message
* **ensure that every commit either references, refers, closes, or fixes a** [**Trac**](http://trac.edgewall.org/) **ticket**  
  Every commit should be associated with at least one Trac ticket. If no ticket for the current commit exists create one before committing your changes. Please refer to the Trac documentation on [how to reference Trac tickets in the commit log message](http://trac.edgewall.org/wiki/CommitTicketUpdater).
* **try to ensure that each commit contains only one logical change**Unrelated / independent code changes should be committed separately. In some cases, when a change leads to the necessarily to perform other code changes or refactorings these additional changes should be committed first and independent of the initial change
* **try to ensure to commit only Java source code *without*** [**FindBugs**](http://findbugs.sourceforge.net/) **errors or warnings**  
  Respect the project specific FindBugs settings and commit only code that is warning and error free. If a warning can't be avoided [suppress the warning using the appropriate FindBugs annotation](http://findbugs.sourceforge.net/manual/annotations.html)
* **try to ensure to commit only Java source code without** [**PMD**](http://pmd.sourceforge.net/) **errors or warnings**  
  Respect the project specific PMD settings and commit only code that is warning and error free. If a warning can't be avoided [suppress the warning using the appropriate PMD annotation](http://pmd.sourceforge.net/suppressing.html)
* **minimize the number of other warnings and errors**  
  Strive to create source code and project files *without* warnings or errors shown by the respective tooling (e.g. XML, JSF, Spring, ...) in Eclipse

### Source Code Comments

Please keep the following best practices in mind when writing source code / Javadoc comments:

* describe why the logic is implemented/necessary not what it does (which is apparent from the code itself)
* describe what a class is for
* describe how the parameters of a method effect the result
* document the non-obvious background knowledge, the intention and not the result
* try to capture the concepts in a central place (preferable in Trac) and reference them
* escape obvious facts with marker tags - don't describe them over and over again, be dry
* include samples, how-to's, and so on in your documentation
* don't allow default Javadoc comments generated by the IDE
* sometimes no doc is the best doc - try to minimize the amount of documentation and describe only the key concepts

### Stability and Capacity

Develop ZARS with stability and capacity in mind; in particular ensure the following when developing:

* avoid memory leaks
* avoid thread leaks
* avoid [classloader leaks](http://mediacast.sun.com/users/Frank.Kieviet/media/JavaOne07-BOF9982-PermGen.pdf)
* ensure graceful startup, shutdown, and restart

### Startup and Shutdown

TODO Ensure graceful startup and shutdown

### Infrastructure Services

Since we are deploying ZARS to a Servlet container (as opposed to a Java EE application server) a number of infrastructure services (that would otherwise be provided by the container) may have to be configured directly in the application. These infrastructure services include:

* **JDBC connection pool**One of numerous standalone JDBC connection pools such as C3P0, Jakarta Commons, DBCP, or XAPool can be used for this purpose
* **JTA transaction manager**Standalone transaction managers such as Atomikos Transactions and JOTM can be bundled with the application to obtain JTA capabilities such as transaction suspension (REQUIRES\_NEW, NOT\_SUPPORTED) frequently required for logging and auditing to the database.
* **JMS connection pool**  
  When using JMS to send or process messages it is important to ensure that connections and sessions are pooled. Pooling of JMS connections and sessions is usually performed by the underlying JCA/MDB container. In a standalone application a dedicated pooling JMS connection provider has to be configured and used.  
    
  When using Spring's JmsTemplate[[8]](#footnote-9) be aware of it's [gotchas](http://activemq.apache.org/jmstemplate-gotchas.html), configure it with either a pooling JMS connection factory (such as [ActiveMQ's PooledConnectionFactory](http://activemq.apache.org/spring-support.html#SpringSupport-WorkingwithSpring%27sJmsTemplate)[[9]](#footnote-10)) or have a look at Spring's [CachingConnectionFactory](http://go2.wordpress.com/?id=725X1342&site=codedependents.wordpress.com&url=http%3A%2F%2Fstatic.springsource.org%2Fspring%2Fdocs%2F2.5.6%2Fapi%2Forg%2Fspringframework%2Fjms%2Fconnection%2FCachingConnectionFactory.html)[[10]](#footnote-11) and the Spring manual to achieve efficient JMS message handling with pooled message senders and listeners for MDPs (message driven POJO). When consuming messages the preferred way is to setup a DefaultMessageListenerContainer[[11]](#footnote-12).
* **Scheduling / Thread Pool**  
  There are standalone thread pool implementations and scheduling libraries such as OpenSymphony Quartz and the JDK Timer that can be bundled with the application. The simplest way to configure a thread pool is probably the use of Spring’s ThreadPoolTaskExecutor[[12]](#footnote-13).

## Coding Guidelines

### Serialization

There are many different opinions when it comes to Java serialization and the dreaded serialVersionUID. Some suggest using the serialver command line utility or to rely on the IDE to auto-generate a serialVersionUID.

As a best practice on ZARS, manually set the serialVersionUID to the current project version of ZARS each time a serializable class is introduced or an incompatible type change is made to a serializable class. Set the serialVersionUID in the following format but left pad version numbers with a leading zero if they have one digit only:

**Format**  
private static final long serialVersionUID = <MAJOR><INTERMEDIATE><MINOR>L;

**Example for ZARS version 1.0.1**  
private static final long serialVersionUID = 0**1**0**0**0**1**L;

Setting the serialVersionUID to the project version helps to identify which version introduced an incompatible type changes for a particular serializable class.

### Encoding

All source code files, static content, input and output, and all DB content is supposed to be encoded in UTF8.

### Internationalization

All elements of ZARS visible to the end user (GUI and Notifications) should be internationalized. There is no need to internationalize exceptions and log messages.

### Service Interface

TODO service signature

### Testing

TODO

### Documenting

TODO

### Source Code Cleanup and Formatting

TODO specific clean up/save/formatter settings

### Logging

• *Support* logging (errors and info) is part of the user interface of the application.

These messages are intended to be tracked by support staff, as well

as perhaps system administrators and operators, to diagnose a failure or

monitor the progress of the running system.

• *Diagnostic* logging (debug and trace) is infrastructure for programmers.

These messages should not be turned on in production because they’re intended

to help the programmers understand what’s going on inside the

system they’re developing.

or even use Notification Rather Than Logging

**logging errorhandlers**  
a pluggable ErrorHandler mechanism in Log4J allows to defined the behavior when exceptions (such as an java.io.IOException while writing to the logfile) occure in one of the appenders. this feature should be used to ensure that logging errors don't cause the application to fail. in particular one should avoid self denial with universal/aspectized/reentrant exception handlers that log any kind of exception *including those occurring while logging an exception*.

**context information for logging and exception messages**  
from time to time it is necessary to pass (additional) context information to methods (via dedicated parameters) in order to generate meaningful and helpful log messages. with exceptions, passing context information down the call tree can be avoided in many cases. exceptions can be catched further up the call tree and wrapped with other exceptions that add required / additional context information.  
  
a good way to avoid additional / dedicated parameters required to generate meaningful log messages (with sufficient context information) is the use of ThreadLocal. required [context information can be stored in a ThreadLocal](http://www.ibm.com/developerworks/library/j-threads3.html#h11704) variable and accessed when generating log messages.

**log entries should include the thread name or transaction identifier**a logging library, such as log4j, should be configured in a way to output the name of the thread (or transaction id) for each log entry. otherwise it's almost impossible to associate log entries with individual transactions on a busy system. it is crucial to be able to identify the steps performed by particular (request handling) threads/transactions.

### Exception and Error Design and Handling

TODO

service exception (aspect, nested exception, ...)

business exception

exceptions

The BusinessException should be used as the super-class for all checked exceptions throughout the Saturn Services project. The BusinessException should be used to express expected conditions demanding an alternative response from the caller. The BusinessException should be seen as an alternative to returning an error code, hence creating a separate subclass for each different business error type is encouraged.

Use the following considerations to determine whether to throw a BusinessException or a RuntimeException

• BusinessException (contingency)

o is considered to be a part of the design

o is expected to happen regularly but rarely

o is taken care of by the upstream code that invokes the method

o should be seen as an alternative return mode

• RuntimeException (fault)

o is considered to be a nasty surprise

o is expected to happen never or very rarely

o is taken care of the people who know the method's implementation details and those who need to fix the problem

o should be used for missing files, unavailable network resources, networking problems, programming bugs, hardware malfunctions, and configuration mistakes

The distinction between contingency and fault is based on the Effective Java Exceptions Dev2Dev article.

numerous stability and capacity patterns as well as availability, monitoring, and transparency are significantly influenced and depend on an [elaborate and consistent exception and error design and handling](http://dev2dev.bea.com/lpt/a/541). ensure that system failures (resource not available) are reported differently than application failures (parameter violations or invalid state) and provide mechanisms to transmit standardized context information to the calling system with each error, failure, and exception. make sure to put enough effort into exception and error design and handling upfront as changing it later is a major undertaking/refactoring task.

### Thread Safety

TODO

if you haven't considered concurrent access in your class design, introducing thread safety might not be as trivial as it may seem and can require significant design modifications. it is far easier to design a class to be thread-safe than to retrofit it for thread safety later.

1. Maven Goal Reference

TODO

used maven build goals

|  |  |
| --- | --- |
| **Maven Goals** | **Description** |
| jetty:run | executes Jetty with the web application deployed |
| clean compile jetty:run | TODO |
| xml:transform | creates XSD schemas out of the taglibs |
| clean compile | TODO |
| clean package | TODO |
| site | TODO |
| clean test | TODO |
| clean verify | TODO |

TODO add and describe all reports created

<http://pietrowski.info/2008/09/maven-project-raports/>

1. Terms and Definitions

|  |  |
| --- | --- |
| **Term** | **Definition** |
| infrastructure services | Services and SPIs conventionally provided by the runtime container (e.g. Java EE application server or Servlet container) such as (1) JDBC data sources and connection pools, (2) transaction managers, (3) JMS connection factories, destinations, and connection pools, (3) scheduling, and (4) thread pools. |

1. preferable the *Java EE Developers* edition [↑](#footnote-ref-2)
2. Subversive is the recommended Subversion plug-in for Eclipse [↑](#footnote-ref-3)
3. The Maven default profile is highlighted in bold. [↑](#footnote-ref-4)
4. Please refer to section for details. [↑](#footnote-ref-5)
5. The Maven default profile is highlighted in bold. [↑](#footnote-ref-6)
6. On Windows it's not necessary to use the installer, simply download the ZIP archive und extract it. [↑](#footnote-ref-7)
7. On Windows it's not necessary to use the installer, simply download the ZIP archive und extract it. [↑](#footnote-ref-8)
8. org.springframework.jms.core.JmsTemplate [↑](#footnote-ref-9)
9. org.apache.activemq.pool.PooledConnectionFactory [↑](#footnote-ref-10)
10. org.springframework.jms.connection.CachingConnectionFactory [↑](#footnote-ref-11)
11. org.springframework.jms.listener.DefaultMessageListenerContainer [↑](#footnote-ref-12)
12. org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor [↑](#footnote-ref-13)