SYBASE
An SAP Company

Mobiliser Framework

Development Guide

Sybase 365 Mobiliser Platform 5.1

Document ID: DC01830-01-0510-01

Last Revised: September 15, 2013

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This document covers how you can extend Mobiliser without Money Mobiliser specific functions and features. This is the business and persistence layer of Mobiliser. Please see the Money Mobiliser documents for Money Mobiliser specific functions and features.

The target audience are solution experts, platform designers and developers that work on/extend the persistence and business layer of Mobiliser.

This guide starts with the general development environment set up, afterwards the new frameworks for

- Persistence with the Data Access Object framework
- Contract first service model
- Auditing service framework
- Security model
- Mobiliser Preference service
- Event and Task service
- Message Gateway
- Channel Manager
- Reporting Service.

are introduced.

Please also read the following chapters before staring with the development of custom modules:

- Utility libraries references
- Best practices
- Testing

Last, but not least, this document provides information on troubleshooting, third party library integration in OSGi, a list of Mobiliser defined error codes and a cron expression reference.

Chapter 2

Getting Started

This chapter describes the fundamental environment setup that is required to work on Mobiliser development. This includes the developer's infrastructure as well as a description of the build process.

2.1 Development Environment

This section describes the standard set up of a development environment. It starts with a list of prerequisites and continues with the Java, database and development IDE set up.

2.1.1 Prerequisites

You should have a good understanding of Java, Spring, Maven, Subversion, Hibernate, Contract-First web services (XSD) and OSGi. This section describes the hardware and software prerequisites that need to be available in order to develop for Mobiliser. This section describes the hardware and software prerequisites that need to be available for doing development for or on Mobiliser.

2.1.2 Hardware and OS

For a working Mobiliser setup, we recommend the following minimal hardware resources:

- A dual core processor
- 4 GB of RAM (more is better)
- 5 GB free space on your hard disk [1]

Mobiliser runs on many operation systems, but the database management system may reduce your choice, if you like to run the database on the same machine. See 2.1.7 for typical setup scenarios.

Valid and tested operation systems are:

- Linux OS, such as RedHat Enterprise Linux, Debian, Ubuntu, SuSE Enterprise Linux
- Windows 7 64 Bit

[1] The packaged Mobiliser container will be about 100MB expanded, the Maven dependencies needed to build will be about 350 MB, the eclipse workspace will be about 150 MB, the eclipse plugins about 100MB, Adaptive Server Enterprize about 450 MB (on Linux), SQurrelSQL about 225 MB, the database files about 1GB, eclipse about 250 MB. Building the project and installing to your local repository will eat through another few hundred MB, giving 5 GB as a generous estimate.

- Apple Mac OS X[2]

Additionally, you need sufficient privileges to install additional applications and services on that machine. In order to access the resources from the source code and Maven repository, you need a user account on the systems.[3]

2.1.3 Java

Mobiliser runs on Java. Make sure you have Java Development Kit 6 installed and available. Java download is available from http://www.oracle.com/technetwork/java/javase. Mobiliser will also run on Java version 7, but currently does not use any Java 7 specific features. Java 7 also introduces some restrictions for XSD max occurrence element definitions.

2.1.4 Integrated Development Environment

The standard IDE for Mobiliser is the Eclipse Java EE IDE for Web Developers http://www.eclipse.org with several plug-ins.

The following plug-ins are required and mandatory for development:

- Source Contol Client such as SVN plugin Subclipse:
  - Subversion 1.6.x http://subclipse.tigris.org/update_1.6.x or
  - Subversion 1.7.x http://subclipse.tigris.org/update_1.8.x
- Maven plugin m2e: http://download.eclipse.org/technology/m2e/releases

The following plug-ins are optional:

- Spring plugins: http://dist.springsource.com/release/TOOLS/update/e3.7/. The most important component here is the Spring IDE core which makes editing Spring XML files easier.

Before starting with development make sure to change the default workspace preferences. Open Eclipse Preferences to change some default settings:

- Java->Code Style->Formatter: Active profile: “Java Conventions [built-in]”
- Java->Editor->Save Actions: Format source code (Format all lines) & Organize imports
- XML->XML Files->Editor: Indent using spaces / Indentation size: 2
- Java->Compiler->Error/Warnings->Potential Programming Problems->Boxing and unboxing conversions->Warning

Hint: After you applied the workspace preference changes you can export it as general workspace settings. Afterwards you can import these settings when creating a new workspace.

[2]Please note that there is a limited database management system (DBMS) support for Mac OS. A work around is to use a virtual machine such as VMWare Player http://www.vmware.com with Linux and a database management system. Configure port forwarding so that you can access the DBMS directly.

[3]Sybase employees using internal Sybase servers can use their Sybase login; external developers accessing customization repositories need a separate account for that. Please send an email to mcommerce-useracc@sybase.com to request a new account.

2.1.5 Source Control

Mobiliser source code is controlled in an SVN repository; you can use the Eclipse built-in Subversion support; however, it is strongly recommended to install the command line client as well to be able to work around problems that may occur when using the Eclipse integration. In addition, if you plan to run Maven releases on your machine, the command line client is a requirement.

Subversion is available from http://subversion.apache.org, you must use version 1.6 or higher. Modify the standard configuration file (located at $HOME/.subversion/config or %USERPROFILE%\Roaming\Subversion\config or similar) to include these settings:

```
1  enable-auto-props = yes
2  [auto-props]
3  *.java = svn:eol-style=native;svn:keywords=Date Revision Id Author HeadURL
4  *.xsd = svn:eol-style=native;svn:keywords=Date Revision Id Author HeadURL
5  *.xml = svn:eol-style=native;svn:keywords=Date Revision Id Author HeadURL
6  *.jmx = svn:eol-style=native;svn:keywords=Date Revision Id Author HeadURL
7  *.json = svn:eol-style=native;svn:keywords=Date Revision Id Author HeadURL
8  *.patch = svn:eol-style=native;svn:keywords=Date Revision Id Author HeadURL
9  *.tex = svn:eol-style=native;svn:keywords=Date Revision Id Author HeadURL;
       svn:mime-type=text/x-tex
10 *.ldif = svn:eol-style=native
11 *.properties = svn:eol-style=native
12 *README = svn:eol-style=native
13 *NOTICE = svn:eol-style=native
14 *.ini = svn:eol-style=native
15 *.html = svn:eol-style=native
16 *.pl = svn:eol-style=native
17 *.php = svn:eol-style=native
18 *.inc = svn:eol-style=native
19 *.py = svn:eol-style=native
20 *.css = svn:eol-style=native
21 *.js = svn:eol-style=native
22 *.jsp = svn:eol-style=native
23 *.sql = svn:eol-style=native;svn:keywords=Date Revision Id Author HeadURL
24 *.tpl = svn:eol-style=native
25 *.txt = svn:eol-style=native
26 *.sh = svn:eol-style=LF
27 *.bat = svn:eol-style=CRLF
28 *.vbs = svn:eol-style=CRLF
29 *.module = svn:eol-style=native
30 *.handlers = svn:eol-style=native
31 *.schemas= svn:eol-style=native
```

Listing 2.1: SVN Standard Configuration File

Also - a general remark for later - never commit any project meta-data like .settings, target, bin, .project, .classpath - this is all automatically generated by Maven on demand and tailored to the local machine.

2.1.6 Build Tool

Mobiliser uses Maven for building. Maven manages the Mobiliser build process and library dependencies. Download and install Maven from http://maven.apache.org. The minimum version required is 3.0.x. Please also change your Eclipse settings to use the installed Maven version rather the shipped one.

The following Paragraph is needed in case you need to connect to the paybox Maven repository or any other repository that requires authentication.

First, configure the Maven client to point to the Sybase® Maven repositories. Create a master password using

```
mvn --encrypt-master-password <password>
```

Just use any random password for that, you will not need this for future use; Maven only uses that to encrypt your actual repository password. Create a settings-security.xml file in your $HOME/.m2 directory and include the output of the previous Maven command:

```
<settingsSecurity>
  <master>{#################}</master>
</settingsSecurity>
```

Listing 2.2: File settings-security.xml

Then, encrypt your account password using

```
mvn --encrypt-password <password>
```

Finally, create a settings.xml in your $HOME/.m2 directory and replace the user name with your account and the password with the output of the previous Maven command.[4]

```
<?xml version="1.0" encoding="utf-8"?>
<settings xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven
    .apache.org/xsd/settings-1.0.0.xsd" xmlns="http://maven.apache.org/SETTINGS
    /1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <servers>
    <server>
      <id>repo</id>
      <username>USERNAME</username>
      <password>{################}</password>
    </server>
    <server>
      <id>raunheim-central</id>
      <username>USERNAME</username>
      <password>{################}</password>
    </server>
    <server>
      <id>raunheim-snapshots</id>
      <username>USERNAME</username>
      <password>{################}</password>
    </server>
  </servers>
  <mirrors>
    <mirror>
      <mirrorOf>*</mirrorOf>
      <name>repo</name>
      <url>https://repo.paybox.net/nexus/content/groups/public-devel</url>
      <id>repo</id>
    </mirror>
  </mirrors>
  <profiles>
    <profile>
      <id>nexus</id>
      <repositories>
        <repository>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
          <id>raunheim-central</id>
          <name>libs-releases</name>
          <url>https://repo.paybox.net/nexus/content/groups/public-devel</url>
        </repository>
        <repository>
          <snapshots />
          <id>raunheim-snapshots</id>
          <name>libs-snapshots</name>
```

[4] If you are a Sybase employee you can change the server URL to optimus.sybase.com and access with your Sybase account.

```
44          <url>https://repo.paybox.net/nexus/content/groups/public-devel</url>
45        </repository>
46      </repositories>
47      <pluginRepositories>
48        <pluginRepository>
49          <snapshots>
50            <enabled>false</enabled>
51          </snapshots>
52          <id>raunheim-central</id>
53          <name>plugins-releases</name>
54          <url>https://repo.paybox.net/nexus/content/groups/public-devel</url>
55        </pluginRepository>
56        <pluginRepository>
57          <snapshots />
58          <id>raunheim-snapshots</id>
59          <name>plugins-snapshots</name>
60          <url>https://repo.paybox.net/nexus/content/groups/public-devel</url>
61        </pluginRepository>
62      </pluginRepositories>
63    </profile>
64  </profiles>
65  <activeProfiles>
66    <activeProfile>nexus</activeProfile>
67  </activeProfiles>
68 </settings>
```

Listing 2.3: File -settings.xml

Hint: Maven may fail with an java.lang.OutOfMemoryError: PermGen space maven Error. If you experience this exception, you must increase the Java maximum *PermGen* size for Maven. Maven recognizes the environment variable MVN_OPTS and pass it to the Java VM. On Unix use export MVN_OPTS=-XX:MaxPermSize=512M.

2.1.7 Database

Mobiliser runs on Oracle 10G and later, IBM DB2 9.7.4 and later, ASE 15.5 and later, Postgres 9.0 and later databases. The default Vanilla container currently comes with configuration and dependencies for all of these databases.

For an optimal development environment, you should have access to a local database instance. A free developer / express edition is sufficient for development and functional tests. This could be also a database running on a local virtual machine (and for Mac OS host this is the only way to run a local database).

The DDL scripts and configuration data is automatically installed by the DBMaintain process/program that will be created when building the customization project.

2.2 Bug Tracking

Upcoming releases are available as “target milestones” in Bugzilla. When closing a Bug this should indicate which version is supposed to have the fix in. Today possible values are 5.1.0 and 5.0.1 (as a bug-fix release on 5.0.0). All currently open bugs and the default settings so that any new bugs should default to 5.1.0 target milestone. In case you’re working on a hot-fix on 5.0.0 please remember to change this. So essentially—anything you fix in the 5_0_STAGING branch has target milestone 5.0.1 (which implies it will be in 5.1 as well); anything you fix in trunk has target milestone 5.1.0 (default). To be able to quickly and easily produce release notes it the future, it is critical to correctly set these values.

The version field for the bug must indicate the version in which the bug was found—not the version in which the bug will be fixed. We should not use 5.0.0-SNAPSHOT anymore, since we’ve a release version by now.

Since we're using Bugzilla for both right now - bug tracking on released software *and* during snapshot development, it's important to get this right. The only bugs we should consider in the release notes are obviously the ones we've fixed from 5.0.0 to 5.1.0 - any intermittent problems we've filed against 5.1.0-SNAPSHOT should not show up in the public list, obviously.

2.3 Build Environment

We use Nexus as the Maven artifact repository. Our Jenkins (http://jenkins-ci.org/) continuous integration tool is integrated with a local Sonar (http://www.sonarsource.org/) to track code quality.

2.3.1 Nexus

For Mobiliser projects to build successfully, you need to configure a Maven repository server that holds all the required Mobiliser Platform artifacts, which are not accessible through public Maven repositories. Create a new *Release* repository on your Nexus that will host all required Mobiliser artifacts. Use the provided tool to load a Nexus server with all Mobiliser Platform artifacts that are part of the distribution. The upload script is located at tools/com.sybase365.mobiliser.dist.tools-<VERSION>.jar.

To run the upload, execute:

```
java -jar com.sybase365.mobiliser.dist.tools-<VERSION>.jar
   ./dependencies.properties
   ../
   http://localhost/nexus
   repo
   admin
   password
```

You can also run the tool without any parameters to get help on the parameters printed on the console.

1. The first argument must point to the property file holding the metadata on all Mobiliser Platform artifacts. It is included in the tools directory as well.
2. The second parameter must point to the root directory of where you have extracted the Mobiliser Platform distribution ZIP file.
3. The third parameter points to your Nexus server URL.
4. The fourth parameter is the name of the Nexus repository to which to upload the Mobiliser Platform dependencies. Create a dedicated repository on your server to host the Mobiliser Platform artifacts.
5. The fifth parameter is the Nexus user name that has upload privilege.
6. The sixth parameter is the user's password.
7. The seventh parameter is optional and indicates whether to use the PUT or REST interface of the Nexus server. Use "rest" or "put". While REST is preferable since it also generates default POMs, the default is PUT, since this is supported on all Nexus servers.

In addition, you need to include other public repositories for common open source libraries. The easiest way is to create a new group on your Nexus server that lists the following in the specified order. Be sure your local Maven settings configuration point to that Nexus group.

1. Maven Central (http://maven.apache.org/ref/3.0.5/maven-settings/settings.html)
2. Spring Enterprise External (http://ebr.springsource.com/repository/app/)
3. Mobiliser Platform Repository

2.3.2 Jenkins

Jenkins[5] does both, continues builds upon each commit and nightly builds to update the Sonar statistics. Any time the build fails, Jenkins will send notification emails.

2.3.3 Sonar

The current code quality statistics is available either through the Sonar web interface directly[6], or via an Eclipse plug-in (http://dist.sonar-ide.codehaus.org/eclipse/)—also available through Eclipse Marketplace.

Access

To enable the Eclipse plug-in, configure the Sonar URL in Preferences→Sonar, then right-click a project, then Configure→Associate with Sonar. To see statistics, switch to the Sonar perspective. Find more details on the plug-in on the Sonar web page.[7]

Policy

Right now Sonar runs with the default SAP settings. We'll work with this for the time being and potentially tweak the configuration over time. Each developer is responsible of addressing any major, critical, or blocker violations on his/her code as soon as possible. Minor and info violations should be addressed as well, but we should evaluate whether or not the rules make sense to us. Also, for test coverage we still have to find a reasonable threshold for our system.

[5] http://optimus.sybase.com/hudson
[6] http://optimus.sybase.com/sonar
[7] http://docs.codehaus.org/display/SONAR/Sonar+Eclipse/

Chapter 3

Customization Project Conventions

Mobiliser projects follow some conventions, which makes it easier to navigate for people accustomed with these conventions. The following sections describe the high-level project and module structures.

3.1 Customization Template

A sample customization project that holds samples for the most common customization tasks is located in the customization project in the distribution at $\mathrm{tools/com.sybase365.mobiliser.dist.project\text{-}<VERSION>\text{-}customization.zip}$. If you extract that archive, you get a ready-to-build project that generates a dozen custom Mobiliser Platform bundles, and builds a Mobiliser Platform container that holds these custom bundles on top of a plain Mobiliser container.

3.2 Project Structure

The standard customization project structure consists of the modules:

- $\mathrm{com.sybase365.mobiliser.custom.project.ams}$: Will contain custom settlement export jobs.
- $\mathrm{com.sybase365.mobiliser.custom.project.brand.client}$: Collection of Mobiliser services used by the Brand Mobiliser states.
- $\mathrm{com.sybase365.mobiliser.custom.project.brand.states}$: Custom Brand Mobiliser states (for constructing SMS/USSD flows)
- $\mathrm{com.sybase365.mobiliser.custom.project.businesslogic}$: Contains all the business logic classes of the customization.
- $\mathrm{com.sybase365.mobiliser.custom.project.channels}$: Specific SMS, USSD, and email channels used in the customization.
- $\mathrm{com.sybase365.mobiliser.custom.project.dist}$: Responsible for wrapping up the final deployment artifacts into a single ZIP file. It will contain the web application, the Brand and Money Mobiliser containers, and the DbMaintain package.
- $\mathrm{com.sybase365.mobiliser.custom.project.handlers.authentication}$: The specific Money Mobiliser handlers used in the authorization phase of a financial transaction. This takes care of the customer authentication as part of the financial transaction processing.
- $\mathrm{com.sybase365.mobiliser.custom.project.handlers.billpayment}$: Specific Money Mobiliser handler used for handling bill payments (invoice payments). This takes care of the communication with the bill issuer when new bills are loaded or bills are being paid.

- com.sybase365.mobiliser.custom.project.handlers.exchangerate: Specific Money Mobiliser handler used mainly in the authorization phase of a financial transaction. This takes care of retrieving the correct currency exchange rate to use.
- com.sybase365.mobiliser.custom.project.handlers.payment: Specific Money Mobiliser handler used in the authorization phase of a financial transaction. This takes care of the management of a new payment instrument type, e.g. authorization/capture/cancel of a payment.
- com.sybase365.mobiliser.custom.project.jobs.cronjob-custom: Custom cron job that is running on specific point in time.
- com.sybase365.mobiliser.custom.project.jobs.event-handler-custom: Customer handlers of existing or new events.
- com.sybase365.mobiliser.custom.project.jobs.event-model: New Events that are created or consumed by the customization are configured in here. Events are usually created by the business-logic or by the endpoints.
- com.sybase365.mobiliser.custom.project.jobs.task-custom: Contains implementations of custom tasks. Tasks are executed regularly on a fix schedule (e.g. for housekeeping).
- com.sybase365.mobiliser.custom.project.persistence.dao-api: Contains the interfaces of the DAOs and the DaoFactory for all new persistence beans (models).
- com.sybase365.mobiliser.custom.project.persistence.dao-hibernate: Contains the Hibernate specific implementation of the dao-api interface.
- com.sybase365.mobiliser.custom.project.persistence.dao-model: Contains the model beans representing custom tables in the database with JPA annotations.
- com.sybase365.mobiliser.custom.project.reports: Contains CrystalReports report definitions. This is built into a jar file, not into a bundle.
- com.sybase365.mobiliser.custom.project.services.context: The context is used to register new endpoints into. This is all managed by the Mobiliser Framework. There are no Java classes in this module but only a property file that must be changed to reflect the correct URL and WSDL name for the endpoints deployed into this context.
- com.sybase365.mobiliser.custom.project.services.contract: Contains the XSD contract for any new contract that is being used. Can import existing contracts (XSDs) from Money Mobiliser and extend or use elements and types in there. It also holds the Java interface for the services implemented in the service.endpoint.
- com.sybase365.mobiliser.custom.project.services.endpoint: Contains the implementation of the interface from the contract and the methods for all the Request and Response elements of the contract. It usually transforms from the contract beans into beans used by the business-logic and then calls methods from the business-logic.

The dashed lines in 3.1 show dependencies between bundles (import/export). The dotted lines represent interface implementations.

3.3 Module Structure

Most bundles of a customization follow a common structure:

- src/main/java: contains all Java classes that are packaged into the bundle
- src/main/resources: contains all resource files that are packaged into the bundle
- src/main/resources/META-INF/spring: contains all the Spring bundle context configuration files used and processed by Blueprint. The configuration can be split across multiple files for easier maintenance. Internally Spring will configure all the beans into one application context.

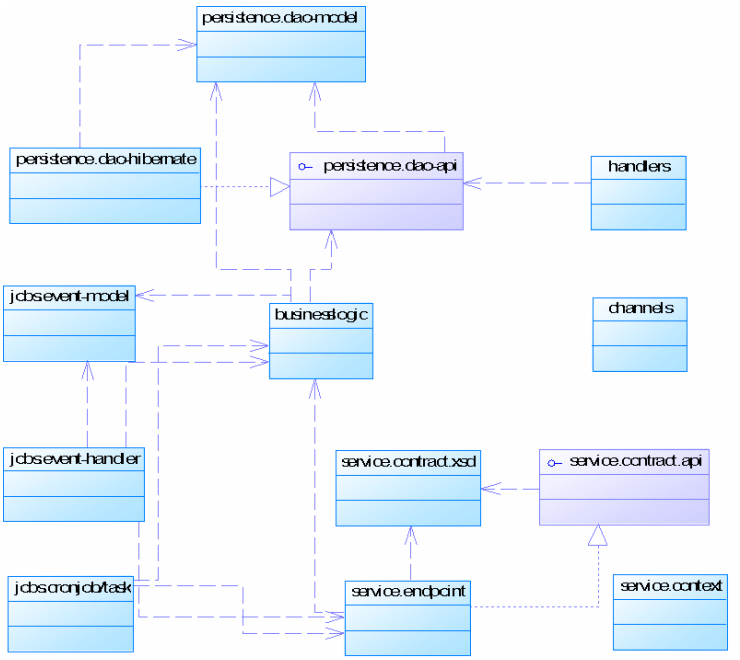

Figure 3.1: Pseudo class diagram, showing the dependencies between most of the standard modules

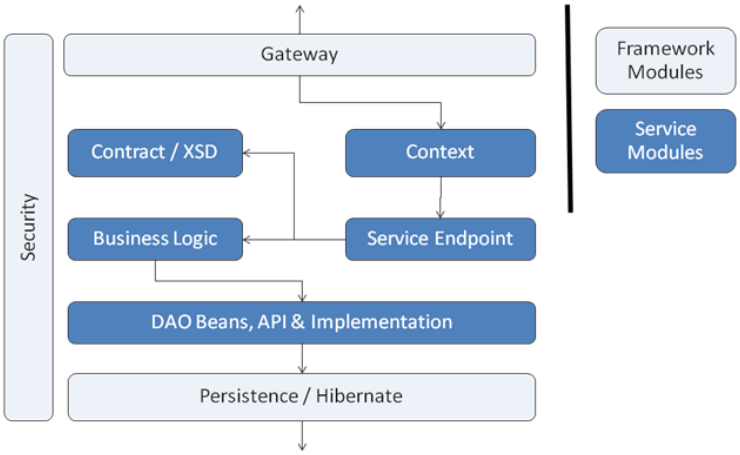

Figure 3.2: This diagram shows the interaction from various bundles to the framework components

- src/test/java: contains all Java test classes (not packaged into main bundle)
- src/test/resources: contains all resource files used for test purposes (not packaged into main bundle)
- target: the maven build process will build all artifacts into this directory. This must not be checked into SVN.
- pom.xml: the Maven build file.

3.3.1 pom.xml

This section will not describe all elements of the epom.xml but concentrate on the bundle specific parts (maven-bundle-plugin). Each pom.xml of a module that will be built into a bundle has a configuration section for the maven-bundle-plugin.

The sections mentioned below refer to 3.5. In section 1 you will list all packages that are visible and imported by other bundles. Section 2 contains all the packages that are internal and are not used (imported) by other

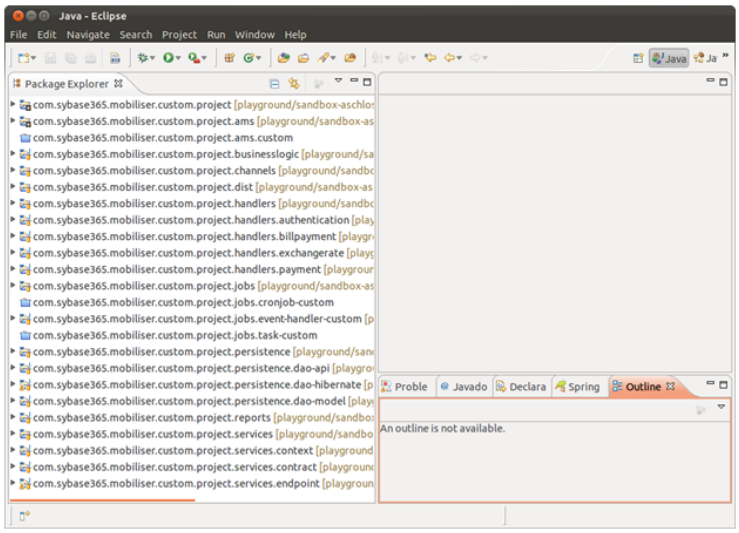

Figure 3.3: Screenshot of Eclipse showing all modules imported into workspace

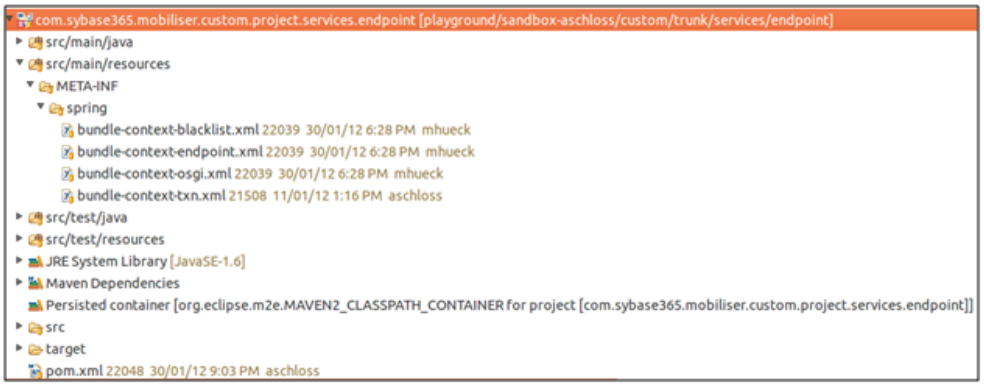

Figure 3.4: Common structure of Modules

bundles. It is still important to list all of your private java packages here, otherwise the automatic import resolution of the maven-bundle-plugin will not work properly. Section 3 finally lists all the imports from other bundles that are used in the current bundle. The direct imports from the classes in the current bundle are represented by the ",*" and do not have to be listed explicitly. However, there are certain imports that cannot be resolved (e.g. because of certain aspects or reflection or resource loading (e.g. XSD)). The import also applies to resources (e.g. XSD) that are available in the bundles. In most cases the existing import section does not need to be modified. There is more information available on the configuration options of the maven-bundle-plugin on the Internet.

3.3.2 bundle-context-osgi.xml

There are usually a couple of bundle-context*.xml files in the given folder and the spring configuration is split upon those logically. The OSGi specific information (that we are focusing on here) is usually located in the file bundle-context-osgi.xml. The other files contain mainly standard Spring bean configuration.

1. An osgi:reference is used to import an OSGi service from the OSGi service registry.
2. The osgi:service is used to register beans (usually defined in the bundle-context.xml) with the OSGi registry.
3. This shows an example of retrieving a certain service by specifying an additional filter on a service-property.

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <instructions>
      <Bundle-Category>custom</Bundle-Category>
      <Export-Package>
        com.sybase365.mobiliser.custom.project.businesslogic
        ,com.sybase365.mobiliser.custom.project.businesslogic.except
        ,com.sybase365.mobiliser.custom.project.converter
      </Export-Package>
      <Private-Package>
        com.sybase365.mobiliser.custom.project.businesslogic.configuration
        ,com.sybase365.mobiliser.custom.project.businesslogic.impl
        ,com.sybase365.mobiliser.custom.project.converter.impl
      </Private-Package>
      <Import-Package>
        com.sybase365.mobiliser.framework.contract.xsd
        ,com.sybase365.mobiliser.framework.event.model
        ,com.sybase365.mobiliser.framework.service.config
        ,org.eclipse.gemini.blueprint.service.importer.support
        ,org.apache.commons.codec.binary
        ,org.springframework.aop.framework
        ,org.aopalliance.intercept
        ,org.springframework.aop
        ,org.aopalliance.aop
        ,net.sf.cglib.proxy
        ,net.sf.cglib.core
        ,net.sf.cglib.reflect
        ,org.springframework.transaction
        ,org.springframework.transaction.interceptor
        ,*
      </Import-Package>
    </instructions>
  </configuration>
</plugin>
```

Figure 3.5: Standard pom Structure

```
<?xml version="1.0" encoding="utf-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:osgi="http://www.eclipse.org/gemini/blueprint
  xmlns:beans="http://www.springframework.org/schema/beans"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/be
  http://www.eclipse.org/gemini/blueprint/schema/blueprint http://www.eclipse.org/gemini/blueprint/schema/

  <!-- IMPORTS -->

  <osgi:reference id="daoFactory"
    interface="com.sybase365.mobiliser.custom.project.persistence.dao.factory.api.DaoFactory" />

  <osgi:reference id="moneyDaoFactory"
    interface="com.sybase365.mobiliser.money.persistence.dao.factory.api.DaoFactory" />

  <osgi:reference id="callerUtils"
    interface="com.sybase365.mobiliser.framework.gateway.security.api.ICallerUtils" />

  <osgi:reference id="internalAut catorFactory"
    filter="(instance=default)"
    interface="com.sybase365.mobiliser.framework.gateway.security.api.InternalServiceCallAuthenticatorFact

  <!-- EXPORTS -->

  <osgi:service ref="blacklistLogic"
    interface="com.sybase365.mobiliser. om.project.businesslogic.IBlacklistLogic" />

  <osgi:service ref="blacklistAction"
    interface="com.sybase365.mobiliser.custom.project.businesslogic.IBlacklistAction" />

  <osgi:service ref="customAuthService"
    interface="com.sybase365.mobiliser.money.businesslogic.transaction.authorisation.IAuthorisationService
    ranking="10">
    <osgi:service-properties>
      <beans:entry key="service" value="CUSTOM" />
    </osgi:service-properties>
  </osgi:service>
```

Figure 3.6: Standard sections of the bundle-context-osgi.xml

4. Additional service-properties can be defined when registering a new service. Those properties can be used when filtering for a service (see above). For more details and more options please refer to the standard Eclipse blueprint documentation.

3.4 Dist Module

The purpose of the dist module is pulling all product and customization artifacts together into a single ZIP file for distribution and installation. Hence, it does not contain any source code, but is rather a meta project containing only Maven build configuration to pull the right artifacts and put them into the right place. 3.7 highlights the structure of the default customization dist module.

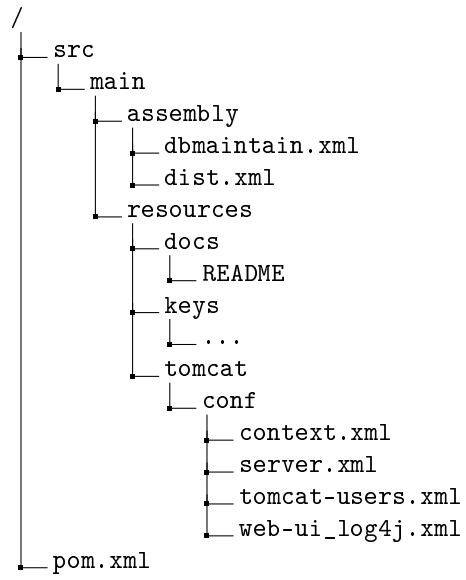

Figure 3.7: Structure of the Customization Dist module

We use the Maven Assembly plugin to arrange all the different artifacts into the right place. The assembly directory holds the configuration for that. In addition to already existing artifacts, coming either from Mobiliser or from the customization modules, the dist module holds additional resources in the resources directory. These resources are also distributed into the right places via the Maven Assembly plugin and the proper configuration in the assembly configuration files.

3.4.1 The POM File

The build process of the dist module consists of these phases, which are reflected in the pom.xml.

1. Download and extract the prepackaged Mobiliser
   - Money Mobiliser
   - Brand Mobiliser
   - Default Tomcat container for web UI
2. Change configuration values in the container's configuration files
3. Download all dependencies of the customization (customization bundles and third party libraries), which are listed in the POM file
4. Arrange the customization
   - Exclude obsolete product bundles
   - Move customization bundles into the right place in Money or Brand
   - Move the customization (or Vanilla) web application WAR file into Tomcat
   - Move Smartphone Mobiliser – Mobile Web into Tomcat
5. Wrap everything up into a single distribution ZIP

POM Header

The POM header (3.1) simply references the customization's parent POM. All version properties are provided in the parent POM and define, which version of the product components to integrate.

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
    /2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM
    /4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
2   <modelVersion>4.0.0</modelVersion>
3   <parent>
4     <groupId>com.sybase365.mobiliser.custom.project</groupId>
5     <artifactId>com.sybase365.mobiliser.custom.project</artifactId>
6     <version>1.0.0-SNAPSHOT</version>
7   </parent>
8   <groupId>com.sybase365.mobiliser.custom.project</groupId>
9   <artifactId>com.sybase365.mobiliser.custom.project.dist</artifactId>
10  <name>AIMS Mobiliser :: Custom :: Project Distributable</name>
11  <packaging>pom</packaging>
```

Listing 3.1: Money Customization pom.xml

Typically, a customization would have only a single database system as the target platform. However, the dist module is prepared to build for different target platform (3.2). By Maven profiles you can manage which platform to build for. The default profile builds for Oracle. You can change this by either setting the activeByDefault flag accordingly, or by running Maven with the -P<profilename> parameter.

```
1   <profiles>
2     <profile>
3       <id>ase</id>
4       <properties>
5         <container.type>ase</container.type>
6       </properties>
7     </profile>
8     <profile>
9       <id>db2</id>
10      <properties>
11        <container.type>db2</container.type>
12      </properties>
13    </profile>
14    <profile>
15      <id>oracle</id>
16      <activation>
17        <activeByDefault>true</activeByDefault>
18      </activation>
19      <properties>
20        <container.type>oracle</container.type>
21      </properties>
22    </profile>
23    <profile>
24      <id>postgresql</id>
25      <properties>
26        <container.type>postgresql</container.type>
27      </properties>
28    </profile>
29  </profiles>
```

Listing 3.2: Money Customization pom.xml

The next section in the POM file (3.3) takes care of downloading and extracting all required dependencies to build the customization distributable. Any Mobiliser products, which are not required, must be commented out here.

```
1   <build>
```

```
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <executions>
          <execution>
            <id>unpack-money</id>
            <phase>prepare-package</phase>
            <goals>
              <goal>unpack</goal>
            </goals>
            <configuration>
              <artifactItems>
                <artifactItem>
                  <groupId>com.sybase365.mobiliser.vanilla</groupId>
                  <artifactId>com.sybase365.mobiliser.vanilla.${container.type}</
                      artifactId>
                  <version>${version.vanilla}</version>
                  <type>zip</type>
                  <classifier>dist</classifier>
                  <overWrite>true</overWrite>
                  <outputDirectory>${basedir}/target/money</outputDirectory>
                </artifactItem>
              </artifactItems>
            </configuration>
          </execution>
          <execution>
            <id>unpack-tomcat</id>
            <phase>generate-resources</phase>
            <goals>
              <goal>unpack</goal>
            </goals>
            <configuration>
              <artifactItems>
                <artifactItem>
                  <groupId>org.apache.tomcat</groupId>
                  <artifactId>apache-tomcat-dist</artifactId>
                  <version>${version.tomcat}</version>
                  <type>tar.gz</type>
                  <overWrite>true</overWrite>
                  <outputDirectory>${basedir}/target</outputDirectory>
                </artifactItem>
              </artifactItems>
            </configuration>
          </execution>
<!--          <execution>
            <id>unpack-brand</id>
            <phase>prepare-package</phase>
            <goals>
              <goal>unpack</goal>
            </goals>
            <configuration>
              <artifactItems>
                <artifactItem>
                  <groupId>com.sybase365.mobiliser.brand</groupId>
                  <artifactId>aims-brand-mobiliser</artifactId>
                  <version>${version.brand}</version>
                  <type>zip</type>
                  <overWrite>true</overWrite>
                  <outputDirectory>${basedir}/target/brand</outputDirectory>
                </artifactItem>
              </artifactItems>
            </configuration>
          </execution>
-->
```

```
66            <execution>
67              <id>copy-dependencies</id>
68              <phase>package</phase>
69              <goals>
70                <goal>copy-dependencies</goal>
71              </goals>
72              <configuration>
73                <outputDirectory>${project.build.directory}/classes/bundles</
                      outputDirectory>
74                <overWriteReleases>true</overWriteReleases>
75                <overWriteSnapshots>true</overWriteSnapshots>
76                <includeScope>runtime</includeScope>
77              </configuration>
78            </execution>
79          </executions>
80        </plugin>
81        <plugin>
82          <artifactId>maven-antrun-plugin</artifactId>
83          <executions>
84            <execution>
85              <phase>generate-resources</phase>
86              <goals>
87                <goal>run</goal>
88              </goals>
89              <configuration>
90                <target>
91                  <move file="${basedir}/target/apache-tomcat-${version.tomcat}"
                        tofile="${basedir}/target/tomcat" />
92                  <delete dir="${basedir}/target/tomcat/webapps/docs" />
93                  <delete dir="${basedir}/target/tomcat/webapps/examples" />
94                </target>
95              </configuration>
96            </execution>
97          </executions>
98        </plugin>
99        <!-- plugin>
100         <artifactId>maven-scm-plugin</artifactId>
101         <executions>
102           <execution>
103             <id>mobileweb</id>
104             <phase>prepare-package</phase>
105             <goals>
106               <goal>checkout</goal>
107             </goals>
108             <configuration>
109               <checkoutDirectory>${project.build.directory}/mobileweb</
                      checkoutDirectory>
110               <connectionUrl>scm:svn:http://orinoco.sybase.com/svn/mobiliser/m5/
                      ui/smartphone/products/WebApp/Mobiliser/${version.mobileweb}</
                      connectionUrl>
111             </configuration>
112           </execution>
113         </executions>
114       </plugin -->
```

Listing 3.3: Money Customization pom.xml

Some basic configuration values coming with the configuration files of Mobiliser distributables are also changed through the Maven build process. We use a Maven plugin to perform replacements based on regular expressions (3.4). Alternatively, you could provide readily configured resource files and make sure that the Maven assembly process later on would exclude the original configuration files.

```
1   <!--    <plugin>
2         <groupId>com.google.code.maven-replacer-plugin</groupId>
```

```
3           <artifactId>maven-replacer-plugin</artifactId>
4           <version>1.4.0</version>
5           <executions>-->
6             <!-- execution>
7               <id>smartphone-mobiliser-location</id>
8               <phase>prepare-package</phase>
9               <goals>
10                <goal>replace</goal>
11              </goals>
12              <configuration>
13                <file>target/mobileweb/mobiliser/SY_Data_Objects.js</file>
14                <replacements>
15                  <replacement>
16                    <token>this.ipaddress = '.*';</token>
17                    <value>this.ipaddress = 'localhost';</value>
18                  </replacement>
19                  <replacement>
20                    <token>this.port = '8080';</token>
21                    <value>this.port = '8080';</value>
22                  </replacement>
23                </replacements>
24              </configuration>
25            </execution -->
26            <!--<execution>
27              <id>brand-mobiliser-location</id>
28              <phase>prepare-package</phase>
29              <goals>
30                <goal>replace</goal>
31              </goals>
32              <configuration>
33                ...
34              </configuration>
35            </execution>
36          </executions>
37        </plugin>
38  -->
```

Listing 3.4: Money Customization pom.xml

The two Maven assembly configurations (3.5) take care of building the customization DbMaintain executable and the complete distributable as a ZIP file, which will include all required modules. Usually, you do not need to modify this part of the POM.

```
1         <plugin>
2           <artifactId>maven-assembly-plugin</artifactId>
3           <executions>
4             <execution>
5               <id>make-assembly-dbmaintain</id>
6               <phase>package</phase>
7               <goals>
8                 <goal>single</goal>
9               </goals>
10              <configuration>
11                ...
12              </configuration>
13            </execution>
14            <execution>
15              <id>make-assembly</id>
16              <phase>package</phase>
17              <goals>
18                <goal>single</goal>
19              </goals>
20              <configuration>
21                ...
```

```
22              </configuration>
23            </execution>
24          </executions>
25        </plugin>
26      </plugins>
27    </build>
```

Listing 3.5: Money Customization pom.xml

It is important to list out all dependencies in the POM's dependency section (3.6) to make sure they are downloaded by the copy-dependency Maven plugin. The plugin does also respect transitive dependencies, but it is good practice to explicitly spell all required dependencies out. Only then, they are available for the Maven assembly so that they can be pushed into the right places for final distribution. **Note: Make sure that any third party dependencies you include are ready OSGi bundles.**

```
1   <dependencies>
2
3     <!-- PROJECT BUNDLES -->
4     <dependency>
5       <groupId>com.sybase365.mobiliser.custom.project</groupId>
6       <artifactId>com.sybase365.mobiliser.custom.project.ams</artifactId>
7       <version>${project.version}</version>
8     </dependency>
9     ...
10
11    <!-- SQL / DBMAINTAIN DEPENDENCY -->
12    <dependency>
13      <groupId>com.sybase365.mobiliser.custom.project</groupId>
14      <artifactId>com.sybase365.mobiliser.custom.project.persistence.dao-model</
            artifactId>
15      <version>${project.version}</version>
16      <classifier>db-platform-${container.type}</classifier>
17      <type>zip</type>
18    </dependency>
19
20    <!-- STANDARD PRODUCT / CONTAINER DEPENDENCIES -->
21    <dependency>
22      <groupId>com.sybase365.mobiliser.ui.web</groupId>
23      <artifactId>com.sybase365.mobiliser.ui.web.application</artifactId>
24      <version>${version.web}</version>
25      <type>war</type>
26    </dependency>
27 <!--
28    <dependency>
29      <groupId>com.sybase365.mobiliser.brand</groupId>
30      <artifactId>aims-brand-mobiliser</artifactId>
31      <version>${version.brand}</version>
32      <type>zip</type>
33    </dependency>
34 -->
35    <dependency>
36      <groupId>com.sybase365.mobiliser.vanilla</groupId>
37      <artifactId>com.sybase365.mobiliser.vanilla.${container.type}</artifactId>
38      <version>${version.vanilla}</version>
39      <type>jar</type>
40      <classifier>scriptarchive-${container.type}</classifier>
41    </dependency>
42    <dependency>
43      <groupId>com.sybase365.mobiliser.vanilla</groupId>
44      <artifactId>com.sybase365.mobiliser.vanilla.${container.type}</artifactId>
45      <version>${version.vanilla}</version>
46      <type>zip</type>
47      <classifier>dist</classifier>
48    </dependency>
```

```
49   </dependencies>
50 </project>
```

Listing 3.6: Money Customization pom.xml

3.4.2 Dist Assembly

The dist.xml configuration file defines where to collect all artifacts from, which must be included in the distribution, and pushes them into the right place in the ZIP file. The content for the distribution ZIP may typically either come from the src/main/resources folder of the *dist* module, or from subfolders of the Maven target folder, where we copied and unzipped required artifacts into in an earlier Maven lifecycle stage (3.3). **Note:** Changes in the POM's dependencies section go hand-in-hand with a change required in the dist.xml to make sure that the new dependency is also considered when wrapping up the distributable ZIP file.[1]

The high-level structure of the default customization ZIP is illustrated in 3.8.

```
/
├── brand ..... Brand Mobiliser with custom states
├── docs
├── money ..... Money Mobiliser with custom bundles
├── sql ..... DbMaintain
└── web
    └── webapps
        ├── mobileweb ..... Smartphone mobile web
        └── portal ..... the web UI
```

Figure 3.8: The customization ZIP

Brand Mobiliser

The default settings will automatically take care of extracting Brand Mobiliser into the ./brand directory. Any custom state bundle that you want to include must be moved into Brand's bundle directory by adding the bundle name to the dist.xml.

Documentation

The full content of the src/main/resources/docs folder is automatically included into the ./docs folder. This is of course only an example. You may exclude the directory completely or you can implement any other additional directory structure using the full flexibility of the assembly plugin.

Money Mobiliser

Money Mobiliser is placed in the ./money directory. The sample dist.xml file contains configuration to exclude some standard product bundles, like payment handlers. Additionally, any customization bundles and their (third-party) dependencies must explicitly be listed as well and included in the proper sub-directory of the ./money/bundles folder. In case you're adding a new sub-folder to hold additional customization bundles, you must also add this folder to the ./money/conf/config.properties file—either via replacing content in the existing file or by providing a new one and replacing the standard version of the file.

DbMaintain

Usually, you do not need to modify the DbMaintain section in the dist.xml. As long as the customization's persistence project follows the given default layout, all customization scripts will automatically be included and

[1] See http://maven.apache.org/plugins/maven-assembly-plugin/ for a detailled description on the Maven Assembly plugin.

a proper DbMaintain executable is generated. The ./sql folder of the distribution will hold the DbMaintain executable and the dbmaintain.properties file.

Tomcat

By default the distribution comes with a pre-packaged Tomcat 7 included. All web applications will be copied into the ./webapps subdirectory, and additional server configuration is copied from the module's src/main/resources into the proper places inside Tomcat.

Smartphone Mobile Web

The mobile web version of Smartphone Mobiliser is checked out of the SVN repository and placed into the ./webapps/mobileweb folder of the packaged Tomcat. Additionally, the dist.xml holds templates to change the configuration to make the mobile web application point to the right Money Mobiliser container

Web Portals

The web application is exploded into the ./webapps/portal folder of the embedded Tomcat. Any additional required configuration must obviously also be copied into the proper places in Tomcat through dist.xml configuration when required.

Chapter 4

Persistence

One of the major components of the Mobiliser architecture is the persistence layer. The persistence layer is responsible for storing and retrieving information from a database system. The persistence layer abstracts the actual operations from the other Mobiliser components

4.1 Schema

We maintain the physical data model in a file, located in the $\mathrm{dao\text{-}mode}$ bundle at $\mathrm{src/main/schema/pd\text{-}model}$. This file is the authoritative source for the data model, any DDL for various target databases will be generated from it.

Note: Using PowerDesigner is optional, once you are working on a customization project where the target DB system is fixed anyway, you may simply go ahead and create the DDLs manually.

Mobiliser supports the following Database Management Systems:

- Sybase Adaptive Server Enterprise 15.7
- IBM DB2 UDB 9.7.4 Common Server
- PostgreSQL 9
- Oracle 10gR2

Please choose the your target platform in PowerDesigner. If you plan to support more than one target platform, choose Oracle 10G as base model type and migrate from this platform to the other platforms.

We restricted the number of possible field types in order to avoid confusion. These data types are also mapped in the base contract XML Schema Definitions to keep incoming beans and database restrictions in sync. Please use only the following types:

Type	ANSI SQL	Oracle	XSD-Type
Short number	numeric(5)	NUMBER(5)	idShort
Long number	numeric(18)	NUMBER(18)	idLong
Exact one character	char(1)	CHAR(1 BYTE)	charOne
Exact two characters	char(2)	CHAR(2 BYTES)	charTwo
Exact three characters	char(3)	CHAR(3 BYTES)	charThree
Short size string	varchar(6)	VARCHAR2(6 CHAR)	strShort
Medium size string	varchar(80)	VARCHAR2(80 CHAR)	strMedium
Long size string	varchar(200)	VARCHAR2(200 CHAR)	strLong
Text string	varchar(2048)	VARCHAR2(2048 CHAR)	strHuge
Fraction Amounts 1	numeric(29,9)	NUMBER(29,9)	decimal [1]
Fraction Amounts 2	numeric(38,19)	NUMBER(38,19)	decimal
Medium numbers	numeric(38,19)	NUMBER(38,19)	decimal totalDigits=12, fractionDigits=6
Timestamp	datetime	TIMESTAMP(6)	dateTime
Date without time	date	DATE	date

Table 4.1: Standard Column Types

Oracle	DB2	Sybase ASE
NUMBER(5)	NUMERIC(5)	numeric(5)
NUMBER(18)	NUMERIC(18)	numeric(18)
CHAR(1 BYTE)	CHAR(1)	char(1)
CHAR(2 BYTES)	CHAR(2)	char(2)
CHAR(3 BYTES)	CHAR(3)	char(3)
VARCHAR2(6 CHAR)	NVARCHAR(6)[2]	nvarchar(6)[3]
VARCHAR2(80 CHAR)	NVARCHAR(80)	nvarchar(80)
VARCHAR2(200 CHAR)	NVARCHAR(200)	nvarchar(200)
VARCHAR2(2048 CHAR)	NVARCHAR(2048)	nvarchar(2048)
NUMBER(29,9)	NUMERIC(29,9)	numeric(29,9)
NUMBER(38,19)	NUMERIC(31,19)[4]	numeric(38,19)
TIMESTAMP(6)	TIMESTAMP	datetime
DATE	DATE	date
date		

Table 4.2: Comparison of Oracle, DB2 and Sybase ASE Standard Column Types

The column names should start with:

ID for any identifier types

BOL for boolean type

STR for string types

DAT for timestamp types and dates

AMT for amount types

Varchar Handling Fist, there is no such thing like a standard varchar handling by the different DBMS. It varies from DBMS to DBMS. The mobiliser XSD schemas express the length of xsd:string as the number of unicode code points. In many places mobiliser relies exclusively on the schema validation to prevent strings which are

[4]IBM DB2 UDB 9.7.4 does not support numeric columns with more than 31 digits. Therefore use NUMBER(31,19) instead

[4]IBM DB2 UDB 9.7.4 does not have a direct equivalent to Oracle's VARCHAR(N char). The standard VARCHAR expresses the number of bytes the column can hold. Since we are using UTF-8 as the database character set, each character may have a variable number of bytes. We are therefore forced to use NVARCHAR which expresses the length in characters, but requires storing the characters as UTF-16BE thus requiring a constant 2 bytes per character.

[4]Sybase ASE equivalent to VARCHAR2(N char) is nvarchar(N) which expresses the length in the number of characters. The amount of stored used depends on the characters, just as in Oracle.

too long from being inserted into the database (and then failing). Therefore a few guidelines for picking your types.

Sybase ASE VARCHAR columns always mean the number of bytes in the database's default charset. There is no option like oracle to say the number of characters. To do that, you should always use NVARCHAR columns which specifies the number of characters. According to the documentation, the types are the same except using nvarchar will make the storage required be "Actual number of characters *@@ncharsize". Using UTF-8 will give you ncharsize == 3

IBM DB2 VARCHAR columns always express the number of bytes in the database charset. Unfortunately, DB2 does not have a type to use the database charset and also at the same time express the length in characters. There is the type NVARCHAR but it is always stored as UTF-16BE which means the storage requirements are fixed, but in some cases larger than UTF-8. UTF-16BE will always uses 2 bytes per character, even for ascii characters, but even Japanese characters only need 2 bytes where as they need 3 in UTF-8. You don't really have a choice though, so you should use NVARCHAR when converting your schema to DB2. Just be aware that the required disk space and memory for sorts etc may increase depending on the type of data being stored. You specify the database character set when creating it, though for NVARCHAR columns it is not used.

CREATE DATABASE mobr5 AUTOMATIC STORAGE YES USING CODESET UTF-8 TERRITORY US COLLATE USING SYSTEM PAGESIZE 32 K;

Oracle The standard varying character in Oracle is VARCHAR2. The number specified in the length of VARCHAR2(n) specifies the number of bytes this column can hold in the database's default character set. So if you have your database in a variable-byte character set (like UTF-8) using the unqualified N is misleading because the number of characters which can be stored in the column depends on what the character is. If you qualify the N with char, VARCHAR(n char), then oracle interprets the number to be the number of characters in the database's default character set.

4.1.1 Create a PowerDesigner Physical Data Model

Sybase PowerDesigner is a convenient design tool for a variety of models such as object oriented, business and physical data models. It allows you to manage even complex models easily. For the database schema design please obtain a Database Developer or Enterprise license of PowerDesigner 15.0.3 or later.

After installing Powerdesigner create a new Physical Data Model from the menu File / New Model

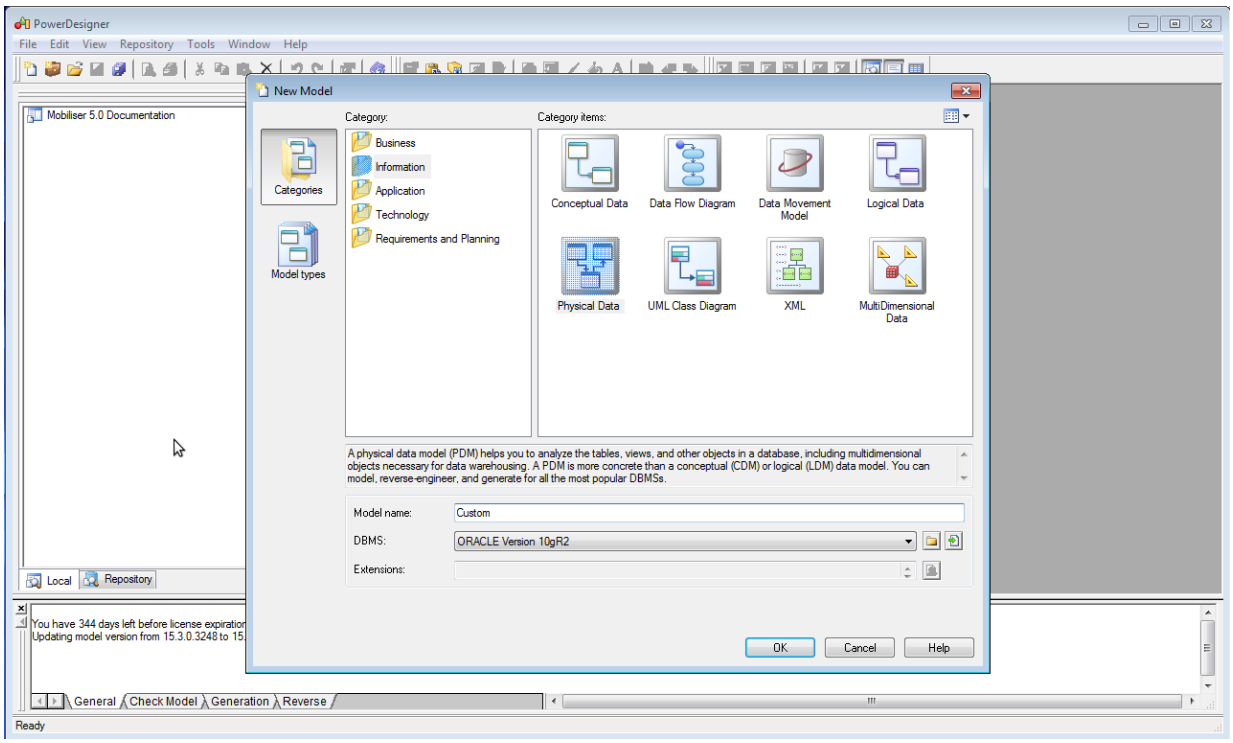

Figure 4.1: PowerDesigner: New Model Dialog

Choose your target database management system. If you plan to support more than one system, choose Oracle 10gR2.

4.1.2 Create or Modify a Table

After you created a new model or opened an existing model, open the physical diagram. Now select the table from the palette [5]. Now point the cursor to an empty area and click to place/create a new table.

[5] If you accidentally closed the palette you can open it by selecting/checking it in Tools/Customize Toolbars . . . and check the Palette box

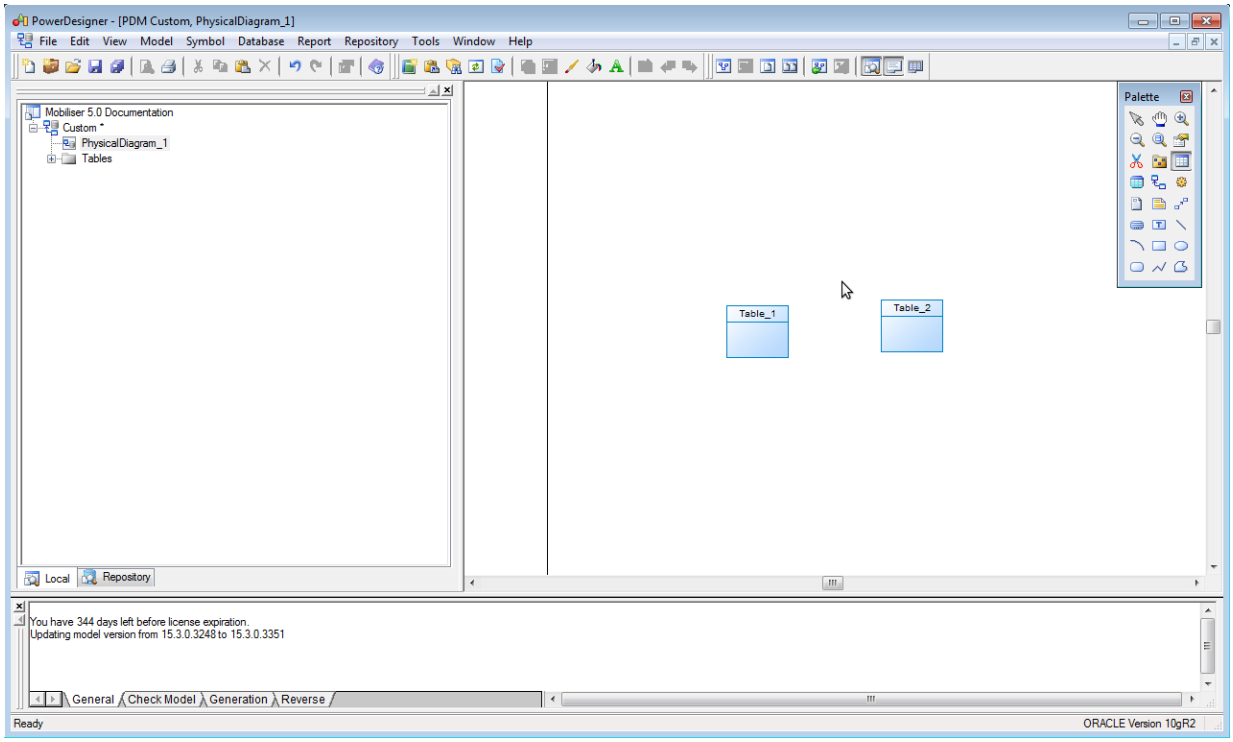

Figure 4.2: PowerDesigner: Physical Diagram with new Tables

To change the table name and add column choose the selector from the palette and double click the table.

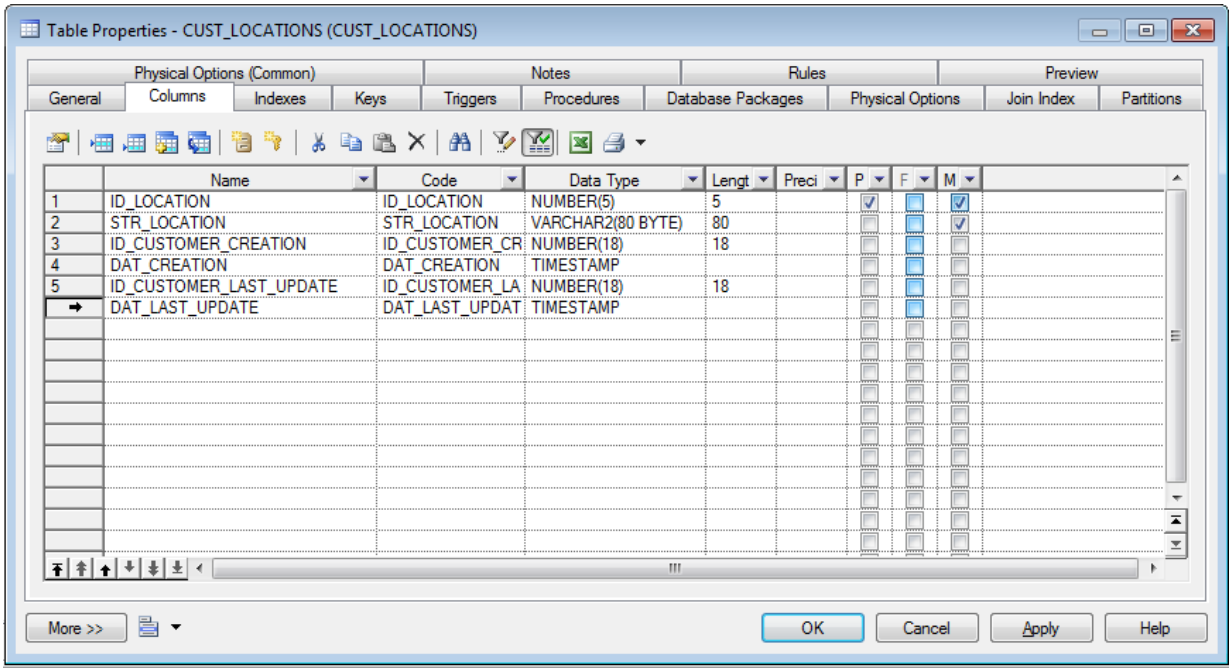

Figure 4.3: PowerDesigner: Table Columns Pane

Change the table name on the first pane and add columns on the column pane. All tables must have the following columns:

Name	Type	Description
DAT_CREATION	datetime	creation date
ID_CUSTOMER_CREATION	numeric(18)	creator
DAT_LAST_UPDATE	datetime	last change date
ID_CUSTOMER_LAST_UPDATE	numeric(18)	last updater

Table 4.3: Standard Table Columns

These columns are managed by the data access layer and used to track the creator and updater of each table row.

Identity Columns

While DB2 supports identity columns as well as sequences, and Sybase ASE only identity columns, we have chosen not to use them anywhere, the primary reason being performance considerations. Coupled with Hibernate's HiLo algorithm, we can achieve much greater throughput than identity columns would give us. Since DB2 supports sequences, we are using them, and while on ASE, we make use of Hibernate's table based sequences which emulate a sequence using a table.

Constraints

Constraints make it possible to further restrict the domain of an attribute. Constraints provide one method of implementing business rules in the database.

Constraints restrict the data that can be stored in relations. These are usually defined using expressions that result in a boolean value, indicating whether or not the data satisfies the constraint. Constraints can apply to single attributes, to a tuple (restricting combinations of attributes) or to an entire relation.

Since every attribute has an associated domain, there are constraints (domain constraints). The two principal rules for the relational model are known as entity integrity and referential integrity. ("Referential integrity" is the state in which all values of all foreign keys are valid. Referential integrity is based on entity integrity. Entity integrity requires that each entity have a unique key. For example, if every row in a table represents relationships for a unique entity, the table should have one column or a set of columns that provides a unique identifier for the rows of the table. This column (or set of columns) is called the parent key of the table. To ensure that the parent key does not contain duplicate values, a unique index must be defined on the column or columns that constitute the parent key. Defining the parent key is called entity integrity.

Even if the Mobiliser persistence layer will also enforce constraints it is useful to define these constraints/business rules in the model too and enforce them when inserting/loading/changing/deleting rows in the tables directly.

Foreign Keys Foreign key constraints ensures data integrity when a table references another table via its key. PowerDesigner easily allows you to model those references. Simply select the reference from the palette and draw a line from the table that references another table.

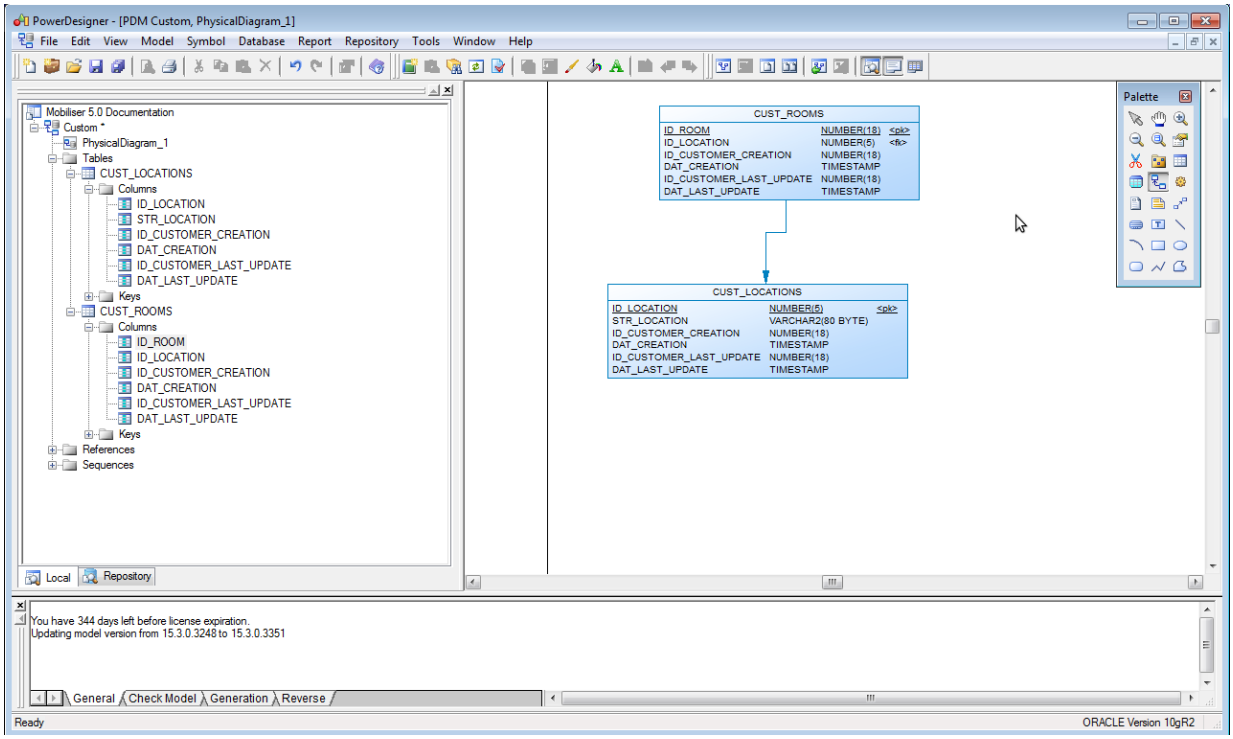

Figure 4.4: PowerDesigner: Foreign Key Reference

PowerDesigner adds the foreign key to the column and marks it as foreign key

Unique Constraints When multiple columns of a table defines a unique value you can define this by adding a unique index. Simply go to the index pane of the table and create a new index. Afterwards double click on the index select the "unique" check box.

Unique constraints in IBM DB2 UDB 9.7.4 may not contain nullable columns, but unique indexes may, which achieves the same result. PostgreSQL interprets the SQL spec here strictly by enforcing NULL != NULL, so such a unique index will actually not work properly on it. In most cases, you can ask yourself it such an index actually makes sense. If you find it does, the only way to implement this is to use a function based index.

If the columns are numeric and never negative, you can use a $\mathrm{coalesce(col,\text{-}1)}$ in the index to enforce the uniqueness of the numeric column. If it is a string and using an empty string is not an option, you will need to use multiple indexes and use expressions for the nullable columns like this (1 nullable column):

$$\mathrm{create\ unique\ index\ idx\_name1\ on\ MOB\_TABLE(COL1,\ COL2)\ WHERE\ COL2\ IS\ NOT\ NULL;}$$
$$\mathrm{create\ unique\ index\ idx\_name2\ on\ MOB\_TABLE(COL1)\ WHERE\ COL2\ IS\ NULL;}$$

If you have multiple nullable columns, you'll need to create as many indexes as combinations. Be especially careful when all columns are nullable. In this case, your last index will be:

$$\mathrm{create\ unique\ index\ idx\_name3\ on\ MOB\_TABLE((0))\ WHERE\ COL1\ IS\ NULL\ AND\ COL2\ IS\ NULL;}$$

This enforces having only one NULL, NULL entry.

Check Constraints When multiple column values of a table relate to each other, for example if column a is not null then column b must not be null too, you can define this rule in the rules tab of a table. Please note: Some database systems only support one check rule/constraint per table, therefore try to add all check logic into one rule/constraint.

Index

An index may speed up access of data via select. You may define them on the index pane of the table.

4.1.3 Trigger

You can define triggers for tables. Please select the trigger pane from the database view and add a new. You can use the provided templates which also allow easy migration from one database system to another. This migration may not work on certain situations and therefore we suggest to use triggers only in special cases, such as history audit or complicated checks.

4.1.4 Sequences

Sequences are required and used for IBM DB2, PostgreSQL and Oracle database management systems only only. Simply add a new sequence via File/new/Sequence. You can define physical options for triggers too. Please note that the Mobiliser persistence layer is using a cached ID generator cache to reduce access to the sequence. Sequence names must start with $\mathrm{SEQ\_}$.

4.1.5 Migrate Model

If you decide to add a new database system you can easily perform this with Tools/Generate Physical Data Model. Select the desired DBMS. PowerDesigner will change the system by translating as much as possible. After migration please check the following open items

- Triggers and Procedures definitions will fail due to the different definition languages. Please review the definition and migrate the definition based on the target language
- Timestamp support varies from language to language. Sybase AS Enterprise only supports one timestamp column per table and timestamps are created automatically, therefore change the column type to datetime.
- Sybase AS Enterprise supports no sequences. When you use a sequence for ID columns and migrate to Sybase AS Enterprise, this column become an identity column. Uncheck the identity flag, because the persistence framework will generate the unique id without database overhead.
- DB2 restricts the constraint name length to 32 characters. Therefore check the model before generating the DDL via Tools/Check Model or pressing F4. Mark all name length errors and select 'auto correct' from the context menu. This restriction is outdated, because the current version allows longer names. You can edit the DB2 template in order to allow longer names.

The original database model is now the master model and linked to the generated ones. The Mobiliser master model is the Oracle model.

4.1.6 Apply Schema Changes

You must apply schema changes to the master model when supporting more than one DBMS. After you applied the changes to the master model you must apply these changes to the dependent models too. To migrate a model, use the Generate Physical Data Model tool from the tools menu in PowerDesigner. Instead of selecting a new model, you apply these changes to the existing child/dependent models.

4.1.7 Generate DDL

After you completed the schema design you may create the DDL script via Database/Generate Database . . . or Crtl + G . Select a target directory and file name and follow the menu. PowerDesigner checks the model and then creates the DDL.

4.1.8 Generate Update Script

One nice feature of PowerDesigner is that it can create update scripts for existing databases which also migrates the existing data. Mobiliser comes with archive models for all DBMS models. First open the current model. Then open the update pane at Database/Apply Model Changes to Database.

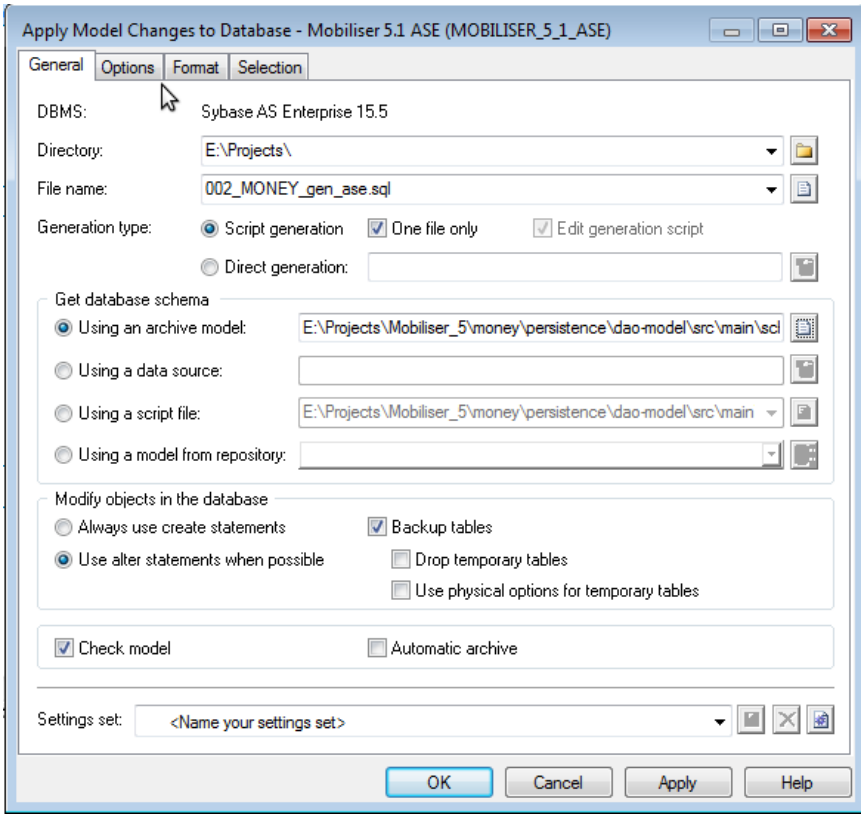

Figure 4.5: PowerDesigner: Apply Model Changes to Database

Select Using an archive model in Get Database schema and select one of the Mobiliser archive schema's. You can also create an archive from your customized Schema by using Save As and select the file type archive.

4.2 Data Model Beans

Data model beans must only use JPA compatible annotations to build up the entity bean class, it is not valid to use any persistence framework specific annotations (e.g. Hibernate,Eclipselink). Also, avoid the usage of inner classes to enable a clean class structure. The Mobiliser data model is defined in the dao-model bundle.

Mobiliser JPA beans extend one of these classes:

com.sybase365.mobiliser.framework.persistence.model.DbEntry
: This is the generic base class for entity beans. All beans that inherit from this class are not updatable and do not have a generated primary key, which is a long value. Each bean that extends this class defines

a primary key as well as proper implementations of equals(), hashCode() and toString(). It always assumes that the two columns DAT_CRATION and ID_CUSTOMER_CREATION are available with the table.

com.sybase365.mobiliser.framework.persistence.model.NoneUpdatableGeneratedIdEntry
This is for entity beans that are not updatable but just created, never changed and that use a database generated primary key, which must be a long value. Usually, this applies for history tables that are only used to permanently store immutable data. The NoneUpdatableGeneratedIdEntry already provides a getter for the id as well as implementations of equals(), and hashCode(), based on the entity id. It also provides an implementation of toString() which is based on getClass() and the entity id. The default implementation maps the ID_ENTITY column to the primary key. There is the possibility to override this default mapping by using the @AttributeOverride annotation.

com.sybase365.mobiliser.framework.persistence.model.UpdatableDbEntry
This is the generic base class for updatable entities; in virtually all cases one does not want to inherit from this class, but from one of the four more specific classes per below. Beans which extend this class take care of defining a PK and proper equals(), hashCode() and toString() implementations. The columns DAT_LAST_UPDATE and ID_CUSTOMER_LAST_UPDATE are added to the two columns defined in the DbEntry.

com.sybase365.mobiliser.framework.persistence.model.CompositeIdEntry<ID>
This is the base class for entities with a composite primary key. The primary key class is provided as a generic to this class. The CompositeIdEntry provides a constructor that is expecting the id, a getter method for the id, as well as implementations of equals(), and hashCode(), based on the entity id. It also provides an implementation of toString() which is based on getClass() and the entity id.

com.sybase365.mobiliser.framework.persistence.model.GeneratedIdEntry

This is the base class for updatable entities that use a database generated primary key, which has to be a long value. The GeneratedIdEntry already provides a getter for the id as well as implementations of equals(), and hashCode(), based on the entity id. It also provides an implementation of toString() which is based on getClass() and the entity id. The default implementation also maps the ID_ENTITY column to the primary key. As with NoneUpdatableGeneratedIdEntry there is the possiblity to override the PK mapping by using the @AttributeOverride annotation.

com.sybase365.mobiliser.framework.persistence.model.IdEntry<ID>
This is the base class for entities with a primitive primary key. The primary key class is provided as a generic to this class. The IdEntry already provides a constructor that is expecting the id, a getter method for the id as well as implementations of equals(), and hashCode(), based on the entity id. It also provides an implementation of toString() which is based on getClass() and the entity id. The default implementation maps the ID_ENTITY column to the primary key. As with NoneUpdatableGeneratedIdEntry there is the possiblity to override the PK mapping by using the @AttributeOverride annotation.

com.sybase365.mobiliser.framework.persistence.model.NamedLookUpEntry
This is the base class for lookup tables, which have an id/name pair (and potentially other information). The NamedLookUpEntry extends IdEntry and comes with the same behaviour with regards to its primary key. The default mapping for the name maps to STR_NAME and sets the length to LENGHT_STRING_HUGE;. To change this behaviour the @AttributeOverride can be used.

Classes that extends the NamedLookUpEntry must implement the protected constructor without parameters and public constructor with primitive ID type as constructor:

```
1 @Entity
2 @Table(name = "MY_CODE_TYPES")
3 @AttributeOverrides({
4     @AttributeOverride(name = "id", column = @Column(name = "ID_CODE_TYPE"))
          ,
5     @AttributeOverride(name = "name", column = @Column(name = "
          STR_DESCRIPTION", nullable = false, length = CouponCodeType.
          LENGTH_STRING_MEDIUM)) })
6 public class MyCodeType extends NamedLookUpEntry {
7     /** Serial version ID. */
8     private static final long serialVersionUID = 1L;
```

```
 9
10      @Deprecated
11      protected MyCodeType() {
12      super();
13      }
14
15      @Deprecated
16      public MyCodeType(int i) {
17      super(i);
18      }
```

Listing 4.1: “NamedLookupEntity Constructors”

In addition to the details mentioned above, the entity beans provide constructor(s), annotated fields and corresponding getters, setters and isSet-methods.

Note: take care of defining the nullable and length properties explicitly when overriding the name column (further information can be found in section "Annotations" 4.2.3).

In addition to the stuff mentioned above, the entity beans must provide constructor(s), annotated fields and corresponding getters, setters and isSet-methods. **Note:** If you are not extending DBEntry or UpdatableDbEntry, you should not override toString(). If some class needs a more comprehensive String representation of an entity, it should use the getters to create such a String.

4.2.1 Constructors

Each Mobiliser JPA bean must provide a default constructor. This constructor has to be public for JPA beans with generated primary keys. Such beans must not provide any other constructors! For all other JPA beans, the default constructor has to be deprecated and protected. It must be available for the Java persistence framework and should not be used programmatically. These beans must provide a second constructor which expects the ID. If the ID has a corresponding primitive type in Java, use the primitive type and convert it into the wrapper object before passing it to the constructor of the superclass.

4.2.2 Fields

Each none-transient field must be annotated with JPA-annotations (see 4.2.3). All field types must be non-primitive. Default values have to be assigned directly to the field. There should be no default values in the database! All fields must be private. Access should be granted by getters and setters.

Please use the default value for serialVersionUID:

```
1 /** Serial version ID. */
2 private static final long serialVersionUID = 1L;
```

Listing 4.2: “Serial Version ID” Serial Version ID

4.2.3 Annotations

All annotations used in the data model beans must be JPA compatible. We do only annotate fields and classes. Please do not annotate methods.

Some of our common patterns for entity beans can be found below. Please consider the following rules.

- NULLABLE: The fact that a column is nullable or not should be represented with annotations. Use the attribute optional = "[true|false]" of the @Basic, @ManyToOne and @OneToMany annotation. Use the attribute nullable = "[true|false]" of the @Column and @JoinColumn annotation. Take care that optional

and nullable always refer to the same value. The used Java persistance framework will check if a field that is not nullable contains a NULL value and then throws an exception before it connects to the DB.

- When using @JoinColumn, you also have to use the @ManyToOne annotation and set the nullable property of the @JoinColumn annotation and the optional property of the @ManyToOne annotation.
- When using @Column, you also have to use the @Basic annotation and set the nullable property of the @Column annotation and the optional property of the @Basic annotation.
- Strings: @Column annotations that are used on String fields (except for fields that are annotated with @Lob) must set the length property. The Java persistence framework will check the length of the value before a connection to the DB is established and throws an exception if the value is too long. Please use the constants for all Mobiliser standard character type length provided by DbEntry:

```
1 public static final int LENGTH_CHAR_ONE = 1;
2 public static final int LENGTH_CHAR_TWO = 2;
3 public static final int LENGTH_CHAR_THREE = 3;
4 public static final int LENGTH_STRING_SMALL = 6;
5 public static final int LENGTH_STRING_MEDIUM = 80;
6 public static final int LENGTH_STRING_LARGE = 200;
7 public static final int LENGTH_STRING_HUGE = 2048;
```

Listing 4.3: Standard Column Length

4.2.4 Getter and Setter

Implement a getter and setter for each field. Usually, the type of the field should also be used for the getter and setter. Some getters and setters need to implement type conversions, e.g., from Character to Boolean (see "common patterns for entity beans" 4.2.6). If the field that corresponds to the getter/setter is nullable, the return type/parameter type must be none-primitive. If the field is not nullable and the type is a wrapper class of a primitive type, the types used in the getter and setter must be the corresponding primitive type. If a field without a default value is not nullable, it is possible that it could not return a value. In this case, the getter should throw a NullPointerException. Hence, the getters can be implemented straightforward. For example:

```
1 public int getStatus() {
2         return this.status.intValue();
3 }
```

Listing 4.4: Setter

Classes that extend the CompositeIdEntry should provide getters for the PK attributes (see "Composite Primary Key").

4.2.5 isSet-methods

For each getter there should be a corresponding isSet-method. This method is required, because there is no other way to check if a field that is not nullable has already been set. To avoid confusion and maintenance problems we have established the convention to implement an isSet-method for every getter, not only for getters of not nullable fields. Hence, you also have to add isSet-methods for getters which delegate to attributes of a PK class. The isSet-method should return false if the corresponding getter would throw a NullPointerException or its return value would be null.

4.2.6 Common Patterns for entity beans

Mappings

All foreign key relations from the data model should be mapped to the corresponding data model bean (if it exists and is not simply a foreign key to a dumb lookup table w/o any configurational content). A

mapped property will not be required in most cases, prefer defining lazy fetching. For more specific uses where there's a particular set of data required it may make sense to define tailored DAO methods.

```
1     @ManyToOne(fetch = FetchType.LAZY, optional = false)
2     @JoinColumn(name = "ID_CUSTOMER", nullable = false)
3     private Customer customer;
```

Listing 4.5: Many To One Mapping

Use OneToMany mappings sparsely, and only in cases where chances are very high that the joined collections will always be used if the entity itself is loaded. For more occasional relationships, rather define a DAO method to perform the load operation.

```
1     @OneToMany(fetch = FetchType.LAZY, mappedBy = "id.invoiceConfiguration")
2     private Set<InvoiceConfigurationAttribute> attributes;
```

Listing 4.6: One To Many Mapping

Boolean

```
1     @Basic(optional = false)
2     @Column(name = "BOL_IS_ACTIVE", nullable = false)
3     private Character dbActive = Character.valueOf('Y');
4
5     public boolean isActive() {
6     return this.dbActive.equals(Character.valueOf('Y'));
7     }
8
9     public void setActive(boolean active) {
10    this.dbActive = Character.valueOf(active ? 'Y' : 'N');
11    }
```

Listing 4.7: Boolean Field

Note: The occurrence of $\mathrm{this.dbActive}$ and $\mathrm{Character.valueOf('Y')}$ in $\mathrm{isActive()}$ should not be switched, because otherwise it will return false instead of throwing a $\mathrm{NullPointerException}$ when $\mathrm{dbActive}$ has no default value and is NULL.

Date

```
1     @Temporal(TemporalType.TIMESTAMP)
2     @Basic(optional = true)
3     @Column(name = "DAT_LAST_UPDATE", nullable = true)
4     private Date lastUpdate;
5
6     public Date getLastUpdate() {
7     if (this.lastUpdate == null) {
8     return null;
9     }
10    return new Date(this.lastUpdate.getTime());
11    }
12
13    @Override
14    public void setLastUpdate(Date lastUpdate) {
15    this.lastUpdate = new Date(lastUpdate.getTime());
16    }
```

Listing 4.8: Date Field

Large Objects

```
1     @Lob
2     @Basic(optional = true)
3     @Column(name = "BIN_CONTENT", nullable = true)
4     private byte[] content;
```

Listing 4.9: Large Object

Floating point numbers

```
1     @Basic(optional = false)
2     @Column(name = "PRC_SHARE", nullable = false)
3     private BigDecimal percentage;
```

Listing 4.10: Floating Point Number

4.2.7 Composite Primary Key

Composite primary keys should be implemented as nested classes which implement Serializable. The fields, annotations and getters should be implemented the same way as the normal entity beans as described above. Do not implement setters, because primary key values should not be updatable. Each PK class must provide a deprecated and protected default constructor and a public constructor which expects values for all PK attributes. This constructor should be the only way to set the values of the attributes. Furthermore, the class that uses the PK class as primary key should provide getters for the attributes of the PK. Hence, it is easier to access the values of the PK. PK classes should also provide isSet-methods as described in section "isSet-methods".

If, e.g., a PK contains a customer as attribute, the subclass of CompositeIdEntry that uses this PK must provide the following getter:

```
1 public Customer getCustomer() {
2         return this.getId().getCustomer();
3 }
```

Listing 4.11: Get a CompositIdEntry

Note: As in all other implementations of entity beans, there is no NULL check in the getter. If the id is NULL, the getter should throw a NullPointerException.

4.3 Data Access Object API

To abstract the use of a direct database connection a DAO (*Data Access Object*) API was introduced. All database access and object manipulations reaching from simple loading of an entity to complex select or update queries have to be done by using a DAO object. These DAO's hide the direct database connection, session and transaction as well as the concrete implementation (*SQL dialect and database system*) of the queries. This makes handling database objects much more transparent and interchangeable from a business logic point of view.

In order to achieve this, for each JPA bean, a corresponding DAO interface exists to define the DAO's functionality and to hide the actual implementation. All DAO interfaces are defined in the dao-api bundle or the corresponding bundle in your customization project. The naming convention for a DAO interface reflects the JPA bean's name with the "DAO" suffix added to it (*e.g., CustomerDAO*).

Each DAO interface extends one of the following base interfaces, which provide a predefined set of methods which each concrete DAO implementation has to provide. There is a strong relationship between the base class which is extended by the JPA bean and the interface you would have to use as a basis for the DAO interface.

Base Interface	Description
BaseDAO	Has to be used if writing a DAO for a JPA bean extending DbEntry
UpdatableDAO	Has to be used if writing a DAO for a JPA bean extending UpdatableDbEntry
IdDAO	Has to be used if writing a DAO for a JPA bean extending IdEntry or NamedLookupEntry
GeneratedIdDAO	Has to be used if writing a DAO for a JPA bean extending GeneratedIdEntry
NoneUpdatableGeneratedIdDAO	Has to be used if writing a DAO for a JPA bean extending NoneUpdatableGeneratedIdEntry
CompositeIdDAO	Has to be used if writing a DAO for a JPA bean extending CompositeIdEntry

Table 4.4: Base DAO interfaces

To ensure a certain level of type safety the DAO API makes use of generics. All interfaces are parameterized by the entity class and the entity id, these generics must be fixed when defining the DAO interface. When extending the GeneratedIdDAO, the id is already fixed to Long.

```
1 public interface CustomDAO extends GeneratedIdDAO<Custom> {
2
3     // TODO: Expose DAO methods implemented in CustomDaoHbnImpl
4
5 }
```

Listing 4.12: Example of a DAO interface

By extending one of the base DAO interfaces, you automatically get methods defined to create a new instance, save/update/delete an entity, get an entity by id, and a few more. The specific DAO interface must then be enhanced by the specific methods applicable for the covered entity; usually these are specific query methods.

The dao-api bundle also defines a generic DAO factory interface, defining getter methods for each supported DAO interface. This way each DAO factory implementation is forced to implement a getter to access the DAO interface implementation. The naming convention follows the standard Java naming convention for getter methods.

You will usually want to also include some finder methods in you DAO which allow you to retrieve a list of Entity beans from the DAO. An example is shown in 4.13.

```
1     List<Custom> findByStatusAndType(int status, int type);
```

Listing 4.13: Example of a DAO finder method in the interface

4.3.1 Large Result Sets

Note: If there is a chance that the list of returned objects is large (>100) you should *never* implement a finder method like the one shown in the previous example but instead define one with a callback that can be used to iterate through a list. Listing 4.14 shows such an example.

```
1     /**
2      * Returns the number of processed entities
3      *
4      * @param status
5      *            the status to filter for
6      * @param type
7      *            the type to filter for
8      * @param customVisitor
```

```
 9       *            the instance that will be called
10       * @param callerId
11       *            the id of the caller
12       * @return the number of entities processed
13       */
14      int workOnCustomWithStatusAndType(final int status, final int type,
15          final ICustomVisitor customVisitor, long callerId);
16
17      public interface ICustomVisitor {
18
19      /**
20       * This method can work on the Custom entity bean. In case any data in
21       * the bean was changed, the method has to return <code>true</code>
22       * otherwise the bean does not necessarily get stored to the DB.
23       *
24       * @param custom
25       * @return <code>true</code> if the entity bean was changed and needs to
26       *         get updated, <code>false</code> otherwise.
27       */
28      boolean workOnCustom(final Custom custom);
29      }
```

Listing 4.14: Example of a DAO method to go through a large list of entities

4.4 Implementing a DAO

More detailed information which parts of a project might be affected by enhancing the DAO later will be given in the following sub-chapters. This chapter will take a look at the DAO interface and the DAO implementation. To enhance the DDL and implement a new JPA bean see 4.1

4.4.1 Quick Guide

This is just a quick guide on how to implement a new DAO. It is recommended you are already familiar with the DAO layer in the Mobiliser and just want to know the steps to implement a new DAO.

1. Create a new entity bean class using only JPA compatible annotations
2. Create a new DAO interface extending a sub-interface of $\mathrm{BaseDAO}$
3. Make sure to define the encapsulated entity class and its primary key class as generic arguments in the interface definition
4. Add a (public) getter for the new DAO interface to the DAOFactory interface;
5. Create a new DAO implementation for the interface which extends a sub-class of $\mathrm{BaseDAoHbnImpl}$ (for a Hibernate implementation)
   (a) Again make sure to provide the encapsulated entity class and its primary key class as generic arguments
   (b) Implement $\mathrm{getEntityClass}()$ to return the encapsulated entity bean class
   (c) add test cases for any specific DAO method that you implement
6. Enhance $\mathrm{HibernateDaoFactoryImpl}$ and add a new class variable for the DAO interface as well as a set of (public) getter and setter methods
7. Configure the spring context to instantiate the new DAO interface implementation and inject the session factory instance
8. Configure the DAO Factory in the spring context to inject an instance of the new (Hibernate) DAO implementation

4.4.2 DAO Implementation

For each supported database system we need to implement the new DAO interface. Currently, only a Hibernate-based implementation is available, located in the dao-hibernate bundle. The naming convention for implementation classes is that it must suffix the interface with DaoHbnImpl, e.g., CustomerDaoHbnImpl.

The DAO Hibernate bundle consists of three parts:

- The DAO Implementation is the implementation of the DAO interfaces that exist for each Model defined in the model bundle.
- The DAO Factory provides an easy access to all DAO classes and is exported via the OSGi service registry. It implements the interface defined in the DAO-API bundle.
- The PersistenceServiceProvider is responsible for registering the Model beans with the Hibernate SessionFactory, again by exporting via the OSGi service registry.

The Hibernate implementation of the new DAO interface needs to implement the DAO interface (obviously) and extend the class BaseDAOHbnImpl or one of the sub-classes that correspond to the used DAO super interface which was used for defining the actual DAO interface. (see 4.4) This offers a Hibernate specific implementation of the BaseDAO interface, again specifying the encapsulated entity bean class and its primary key class. You must implement the method getEntityClass() derived from BaseDao. This method needs to return the concrete class of the entity bean class you want this DAO to encapsulate. This method is necessary since in Java5/6 the class names of the generic values are not available at runtime due to type erasure. For easier use this method is called for each operation that needs to know the entity bean class (e.g. CRUD operations). This means the encapsulated entity bean class is referenced a total of three times for each new DAO:

1. DAO interface by extending BaseDAO
2. DAO implementation by extending the BaseDAO implementation
3. Implementing getEntityClass()

Note: It is very important that these three definitions of the entity bean class (and its primary key class) are coherent!

This is the sample Hibernate implementation of a DAO interface

```
1  package com.sybase365.mobiliser.money.persistence.dao.hibernate.custom;
2
3  import com.sybase365.mobiliser.money.persistence.dao.api.custom.CustomDAO;
4  import com.sybase365.mobiliser.money.persistence.dao.hibernate.common.
       GeneratedIdDAOHbnImpl;
5  import com.sybase365.mobiliser.money.persistence.model.custom.Custom;
6
7  /**
8   * <p>
9   *
10  * </p>
11  * <p>
12  * &copy; 2011 by Sybase Inc.
13  * </p>
14  *
15  * @author <a href='#'>Mobiliser Plain Endpoint Archetype</a>
16  *
17  */
18 public class CustomDaoHbnImpl extends GeneratedIdDAOHbnImpl<Custom> implements
19  CustomDAO {
20
21     public CustomDaoHbnImpl() {
22     }
23
```

```
24      @Override
25      public Class<Custom> getEntityClass() {
26        return Custom.class;
27      }
28
29      // TODO: Implement CustomDaoHbnImpl methods (exposed in CustomDAO)
30  }
```

Listing 4.15: Example DAO implementation

A simple finder can be implemented like shown in example listing 4.16.

```
1       @SuppressWarnings("unchecked")
2       @Override
3       public List<Custom> findByStatusAndType(int status, int type) {
4       return (List<Custom>) this
5           .getSession()
6           .createQuery(
7               "from Custom where status = :status and type = :type")
8           .setInteger("status", status).setInteger("type", type).list();
9
10      }
```

Listing 4.16: Example DAO finder method implementation

Large Result Sets

A more complex example that uses internally a ScrollableResults to iterate through the results is shown in example 4.17.

```
1       @Override
2       public int workOnCustomWithStatusAndType(int status, int type,
3           ICustomVisitor customVisitor, Long callerId) {
4       int cnt = 0;
5       ScrollableResults results = getSession()
6           .createQuery(
7               "from Custom where status = :status and type = :type")
8           .setInteger("status", status).setInteger("type", type)
9           .setFetchSize(200).scroll(ScrollMode.FORWARD_ONLY);
10      try {
11          while (results.next()) {
12          cnt++;
13          Custom custom = (Custom) results.get(0);
14
15          if (customVisitor.workOnCustom(custom)) {
16              this.update(custom, callerId);
17          }
18          // remove from session to avoid OutOfMemory:
19          this.evict(custom);
20          }
21          return cnt;
22      } finally {
23          results.close();
24      }
25      }
```

Listing 4.17: Example DAO finder method with cursor

This is always the preferred way over returning a large list of objects!

Example 4.18 shows an example of how to work with such a method.

```
1     ...
2     customDao.workOnCustomWithStatusAndType(0, 0, new ICustomVisitor() {
3
4         @Override
5         public boolean workOnCustom(Custom custom) {
6         if (custom.getReference().equals("blabla")) {
7             custom.setStatus(9);
8             return true;
9         }
10        return false;
11        }
12    }, -1L);
```

Listing 4.18: Example of how to use the cursor based DAO method

Sorting Result Sets

Sometime the business logic relies on sorted result sets. The straight forward implementation uses the database build in order by <column> asc|desc query function, but this only returns reproduce able result sets on different database management systems if the result set column does not contain null values. Most DBMS will order null values first (or last on order by ... desc and others allows the user to influence this behavior . Therefore you must not use the DBMS built in order by function when the result set column can include null values and use Javas Collections.sort method with a customized Comperator interface implementation 4.19.

```
1     ...
2  public List<CustomerSession> findOpenByCustomerId(long customerId) {
3     String hql = "select cs from CustomerSession cs "
4         + " where cs.customer.id = :customerid and cs.logoff is null ";
5
6     final List<CustomerSession> result = getSession().createQuery(hql)
7         .setLong("customerid", customerId).list();
8     if (result == null) {
9         return new ArrayList<CustomerSession>();
10    }
11    Comparator<CustomerSession> loginDateComperator = new Comparator<
          CustomerSession>() {
12
13        @Override
14        public int compare(CustomerSession arg0, CustomerSession arg1) {
15        if (arg0.getLogin() == null) {
16            return arg1.getLogin() == null ? 0 : -1;
17        } else {
18            return arg1.getLogin() == null ? 1 : arg0.getLogin()
19                .compareTo(arg1.getLogin());
20        }
21        }
22    };
23    Collections.sort(result, loginDateComperator);
24
25    return result;
26 }
```

Listing 4.19: Example of how to sort nullable result set columns

4.4.3 DAO Factory

The Hibernate DAO factory is also located in the dao-hibernate bundle. It implements the DAOFactory interface, hence you must add the new getter methods for the new DAO interface. To enable the injection of

concrete DAO implementation, add a class variable of the new DAO interface (e.g. private CustomerDAO customerDao;) and add the simple getter and setter methods.

You must update the Spring configuration file to have the new DAO implementation class instantiated and injected into the Hibernate DAO factory:

```
1   <bean id="customerDaoImpl" class="com.sybase365.mobiliser.money.persistence.dao
        .hibernate.custom.CustomDaoHbnImpl">
2     <property name="sessionFactory">
3       <ref bean="sessionFactory" />
4     </property>
5   </bean>
6
7   <bean id="daoFactory" class="com.sybase365.mobiliser.core.framework.data.dao.
        factory.hibernate.impl.HibernateDaoFactoryImpl">
8   <!-- other DAO impls get injected already... -->
9
10  <property name="customerDao">
11    <ref bean="customerDaoImpl" />
12  </property>
13  </bean>
```

Listing 4.20: Example of a DAO Spring configuration

Unless you have added new Java classes to new Java packages there is no need to make changed to the pom.xml.

If you inspect the bundle-context-osgi.xml you will notice that the sessionFactory is imported from the OSGi registry (and added to each DAO implementation). The Hibernate DAO factory itself is exported as an OSGi service and can be consumed by any bundle in the container through the DAO factory interface. This way, any bundle using it is agnostic to the underlying DAO implementation and works purely on the defined DAO interfaces and DAO model entity beans. The Mobiliser framework bundles not only export the session factory that is used by the DAO implementation classes, but they also export the accompanying transaction manager, so if you want to use the DAO factory to actually access data, you will always want to import the transaction manager as well as the DAO factory from the OSGi service registry:

```
1 <osgi:reference id="transactionManager"
2   interface="org.springframework.transaction.PlatformTransactionManager"
3   context-class-loader="service-provider" />
4
5 <osgi:reference id="daoFactory"
6   interface="com.sybase365.mobiliser.money.persistence.dao.factory.api.DaoFactory
        " />
```

Listing 4.21: Importing the DAO factory and transaction manager

4.4.4 Exception Translation

Since we have encapsulated all data access code behind our DAO implementation, you should not use any checked exceptions in these interfaces. Also, since we could be using JDBC, Hibernate, or in future something completely different, you should use Spring's DataAccessException hierarchy. This will allow code using the DAO to catch DataAccessExceptions without knowing the underlying implementation. In most cases the DataAccessException will not be caught and will be transformed into an error code further up the call stack, but should you need to, you do not have to guess what kinds of exception could be thrown by the implementation.

Instead of arduously catching and rethrowing exceptions in your DAO, you can use a crosscutting aspect in the Spring configuration of your DAO bundle to translate exceptions.

If you are using Hibernate, the following XML snippet should get you started:

```
<osgi:reference interface="org.springframework.dao.support.
    PersistenceExceptionTranslator"
  id="exceptionTranslator" sticky="false" />

  <bean id="exceptionAdvice"
    class="org.springframework.dao.support.
       PersistenceExceptionTranslationInterceptor">
    <property name="persistenceExceptionTranslator" ref="exceptionTranslator" />
  </bean>

  <aop:config>
    <aop:pointcut id="daoService"
      expression="execution(* com.sybase365.mobiliser.money.persistence.dao.api
          .*.*.*(..)) || execution(* com.sybase365.mobiliser.framework.
          persistence.dao.api.*.*(..))" />
    <aop:advisor pointcut-ref="daoService" advice-ref="exceptionAdvice" />
  </aop:config>
```

Listing 4.22: Translating Hibernate Exception into DataAccessExceptions

The framework's Hibernate bundle will export an appropriate exception translator which translates Hibernates exceptions into Springs DataAccessExceptions. We simply retrieve the reference from the OSGi registry and then register a Spring Auto-Proxy which will proxy all of our DAO methods to do the translation.

For JDBC code, if you are using a JdbcTemplate which is discussed further on, you automatically get exception translation and require no further configuration.

4.4.5 Persistence Service Provider

As a last step, if a new model bean has been implemented on which the DAO is based, this model bean also needs to be added to the list of model beans returned by the implementation of the PersistenceServiceProvider. This is used to construct one common Hibernate Session Factory that is aware of all Model beans that are deployed into the container.

A PersistenceServiceProvider defines

- The list of Hibernate beans,
- a cache configuration file name
- the required database setup version

The default database setup version check simply validates against the MOB_VERSIONS table. The Hibernate factory needs the list of persistent bean classes and the cache strategy. This can be done by implementing the interface com.sybase365.mobiliser.framework.persistence.hibernate.sessionfactory.api.PersistenceServiceProvider.

The getPersistenceClasses() method returns the list of provided classes and their cache strategy (see Caching 4.4.6 for details on defining the cache strategy) The list can be created the following way:

```
List<PersistenceClass> lpc = new ArrayList<PersistenceClass>();
// Fist add all bean classes that needs cache concurrency mode read/write
lpc.addAll(PersistenceClass.createBeans(new Class[]{
            Attachment.class,
            AuditLog.class,
            BatchRun.class,
            Bully.class,
            Customer.class,
            CustomerAttribute.class,
            CustomerSession.class,
            XmlMessage.class
          },CacheConcurrencyStrategy.READ_WRITE));
```

Listing 4.23: Exampele getPersistenceClasses() implementation

This will set the cache concurrency strategy and for all collection methods in one call. If hibernate can not add a collection you can disable the cache by providing a specific cache strategy for the collection:

```
1 lpc.addAll(PersistenceClass.createBeans(new Class[]{ InvoiceConfiguration.class},
2             CacheConcurrencyStrategy.NONSTRICT_READ_WRITE,
3             CacheConcurrencyStrategy.NONE));
```

Listing 4.24: Special handling for non chached collection

Note: The third parameter is used to configure the collection caching.

4.4.6 Caching

The Mobiliser uses Hibernate with Ehcache provider. The persistence bean cache configuration takes place in different files/classes. If you add a new persistence bean to the session factory implementation please check the following two areas: Concurrency Cache Strategy: The concurrency cache strategy describes how the persistence bean instances are accessed (nature of the associated database rows). Based on the nature you can choose one of the following strategy when using Ehcache (ordered by performance):

CacheConcurrencyStrategy.READ_ONLY Caches data that is never updated. This is the optimal cache setting and allows fast access, because no locking is involved. Use this strategy for all constant values, like lookup tables such as Payment instrument types or identification types. Please note: You cannot invoke save(), update() or saveorupdate() on these beans.

CacheConcurrencyStrategy.NONESTRICT_READ_WRITE Caches data that is sometimes updated (rarely) without ever locking the cache. If concurrent access to an item is possible, this concurrency strategy makes no guarantee that the item returned from the cache is the latest version available in the database. Use this strategy for fast access for tables that may change, such as customer credentials or identifications.

CacheConcurrencyStrategy.READ_WRITE Caches data that is sometimes updated while maintaining the semantics of "read committed" isolation level. If the database is set to "repeatable read", this concurrency strategy almost maintains the semantics. Repeatable read isolation is compromised in the case of concurrent writes. Typical use cases are the transactions and sub transactions

CacheConcurrencyStrategy.NONE Disables the 2nd level cache and forces direct database access. Use this strategy as a last resort or if the bean does not need to be cached because all bean attributes are already cached.

Ehcache specific configuration

Mobiliser comes with a default Ehcache configuration that is based on the Ehcache failsafe configuration. The default disables the disk based cache extension. You can find ehcache-failsafe.xml in the bundle com.sybase365.mobiliser.framework.persistence.hibernate.sessionfactory. You can override this default file location via OSGI ConfigAdmin property ID ehCacheConfiguration. Example:

```
ehCacheConfiguration=/opt/sybase/mobiliser/conf/global-hibernate-ehcache.xml
```

Each persistence factory implementation can add a configuration file with specific configurations for beans collections and named query cache. This configuration is merged to the Mobiliser defaults when the implementation returns the package URL as string.

See this example:

```
1    @Override
2    public String getEhCacheConfigurationFileName() {
3      return ConfigHelper.findAsResource(this.ehCacheConfigurationFileName).
           toExternalForm();
4
5    }
```

Listing 4.25: Configure caching of beans from file

ConfigHelper is part of the Hibernate library and returns the URL of a given library/bundle resource. The advantage of this helper is, that is returns the correct URL for resource in an OSGi bundle.

Persistence Bean Cache Settings & Collections These settings are added during factory and allows to specify none default EhCache setting. You can verify it during startup: ehcache will not log a warning that it will use the default settings.

Here is an example for persistence beans and collections:

```
1  <ehcache>
2  <!-- configuration of the base class -->
3    <cache
4       name="com.sybase365.mobiliser.money.persistence.model.system.
            AuthenticationMethodRule"
5       maxElementsInMemory="10"
6       eternal="false"
7       timeToLiveSeconds="6000"
8       overflowToDisk="false"
9   />
10  <!-- configuration of the collection in a class -->
11  <cache
12      name="com.sybase365.mobiliser.money.persistence.model.system.
            AuthenticationMethodRule.steps"
13      maxElementsInMemory="10"
14      eternal="false"
15      timeToLiveSeconds="6000"
16      overflowToDisk="false"
17  />
18  <!-- configuration for sub class -->
19  <cache
20      name="com.sybase365.mobiliser.money.persistence.model.system.
            AuthenticationMethodRule\$Step"
21      maxElementsInMemory="10"
22      eternal="false"
23      timeToLiveSeconds="6000"
24      overflowToDisk="false"
25  />
26
27 </ehcache>
```

Listing 4.26: Example for persistence beans and collections

Named Query Cache Named query cache allows you to specify special Ehcache configuration for the query cache :

```
1          <cache
2                  name="query.com.sybase365.mobiliser.money.persistence.hibernate.
                       dao.system.UseCaseFeeDaoHbnImpl.searchUseCaseFees"
3                  maxElementsInMemory="100" eternal="false" timeToLiveSeconds="60"
4                  overflowToDisk="false" />
```

Listing 4.27: Example for persistence beans and collections

The name of the cache should start with query. and have the fully qualified method name. In order to use it, invoke setCacheRegion(<name>) along with setCacheable(true) on the SQLQuery or HQLQuery object. **Note:** Currently the query cache is disabled in Mobiliser for performance reasons.

4.4.7 JDBC

You may come across a use case where you need to fall back to JDBC. This is not a problem, but should be best abstracted away just as your hibernate code is. So define normal DAO interfaces and then in your implementation add your JDBC code.

Best Practices

You can safely have Hibernate and JDBC data access code working on the same transaction (JDBC connection), but you must take note of a couple of things.

- When inserting / updating data with Hibernate, you must explicitly flush the session, otherwise you won't be able to see the changes in JDBC.
- Since all our transactions are managed by spring, it is best to use a JdbcOperations reference to your data access code. One is automatically exported to the registry, so you must only define a reference to it in your bundle context. A JdbcOperations object will automatically use the same JDBC connection as the one Hibernate has already opened. Or in the case where you do JDBC first, it will bind the connection in the synchronization manager causing Hibernate to get the same exact connection when it does its data access.
- If you absolutely insist on using plain JDBC, you must wrap the datasource you get from the registry in a org.springframework.jdbc.datasource.TransactionAwareDataSourceProxy. Note this means that a call to getConnection() will not return a new connection but will check if there is a running spring transaction with a connection already bound. If there is no connection, you will get a new one.

Services exported to the registry

- org.springframework.jdbc.datasource.lookup.DataSourceLookup
  - allows access to the configured datasource with getDataSource(null) (actually any string value will return the same datasource in Mobiliser's case)
  - please note the things mentioned above, especially about wrapping if doing raw JDBC
- org.springframework.jdbc.core.JdbcOperations
  - If not using Hibernate, this should be the standard reference to fetch to do your JDBC work.

4.5 Transaction Demarcation

The persistence layer is used by other components, such as services, events and messaging. These components defines also the transaction demarcation logic, like single or multiple transaction contexts. The persistence layer therefore does not take care about transaction demarcation at all. The transaction demarcation principles are documented in section 5.1.1.

4.6 Data Scripts

The actual schema setup is done via a DBMaintain package that contains the schema DDL scripts. DBMaintain is used to create the initial database rows too via SQL scripts. The script location is usually daomodel/src/main/schema/scripts/common/*nnn_name*/*nnn_script*.sql.

nnn_name The first 3 digits defines the sequence in which the scripts of this folder are executed. 000 to 099 are reserved and used by the Mobiliser platform.

***nnn_script*.sql** This is the actual script. Again the first digits defines the sequence in which the scripts are executed.

Note: Since DBMaintain keeps track of already executed scripts, you *must not* insert any new scripts in folders with lower numbers than the current maximum, and you *must not* create new files with an index lower than the maximum index that has already been executed in the folder where you're adding that file to. This means, once a folder 500_FOLDER and a folder 501_FOLDER have been executed, it is illegal to add or modify scripts in the 500_FOLDER. Also, if in a folder the script $100\_\mathrm{DATA.sql}$ has been executed, you must not modify that script or any scripts with a lower index, neither must you add a new script file with a lower index. The only legal option is to add a new file with a higher index, e.g., $101\_\mathrm{MORE\_DATA.sql}$. If you do not follow these rules, DBMaintain will only run successfully by wiping the entire database and reinitializing it from scratch – which might be fine in some development cycles, but be careful and understand the implications if you modify existing files or add files in folders with a lower index etc.

After adding new data scripts make sure that you add them to the $\mathrm{src/main/assembly/*}$ assembly scripts so that they will get included into the DBMaintain script archive build.

- The data scripts are standard SQL scripts.
- Each SQL command is separated by a semicolon - ;[6]
- Identity column values must be defined [7]
- Some DBMS does not support implicit data type transformation, like inserting a VARCHAR value into a NUMERIC column. Therefore do not do it.
- Use the format $\mathrm{'YYYY\text{-}MM\text{-}DD\ HH{:}mm{:}SS'}$ to insert timestamps.

[6] Sybase AS Enterprise does not immediately support ; as command separator. The ASE assembly process must therefore replaces the ; by <newline> go.

[7] Sybase AS Enterprise does not allow direct insert of identity values into a identity column. We do not typically use identity columns in mobiliser, so this should not be a problem.

Chapter 5

Services

Services are used to make business logic publicly available, either a single business logic action or an orchestration of business logic. A service will take the request object, convert the contract beans into Java persistent beans, using the converters provided by the business logic bundles, and invoke the business logic. The outcome from the invocations is converted back into contract beans and returned in the response object.

The Mobiliser follows the *Contract First* approach, so the service contract is defined first and the endpoint uses the contract defined requests, responses and implements the service methods.

5.1 Component Architecture

This section documents the component architecture for services and takes a deeper look into the transaction context/manager and access security for service calls.

The Mobiliser allows you to expose a single services as SOAP and REST (for JSON or XML) web service at once. Please see details documented in the “Mobiliser Gateway” chapter in the “Mobiliser Framework Architecture and Design” guide.

A service is built out of four bundles that work together. The bundle specifics are explained in the following section. The bundles are:

Contract: The service contract defines the requests and responses using XML schema definitions. The service framework uses these request and response beans and translates it into the service definition.

Business Logic: The business logic implements the service logic. It is responsible to access the database layer and also can provides beans for complex request and response parameters. Last, but not least it also may provide converter for contract beans for this service. The business layer is also responsible for the transaction context.

Service Endpoint: The service endpoint implements the web service methods, using contract beans for request and response. The service framework uses the interface to build the web service definition exported by this service. The service implementation uses the business logic bundle services to convert the request and response beans from and to contract and calls the business logic implementation for performing the service. The service interface also defines the access security via annotation.

Service Context: The service context bundle is an aggregation of a set of endpoints and defines how the services are published.

5.1.1 Transaction Manager

The transaction manager is used to run modifying service calls in a transaction manager and a service exception may revert all database changes. Based on the service type, Mobiliser provides two different ways that can

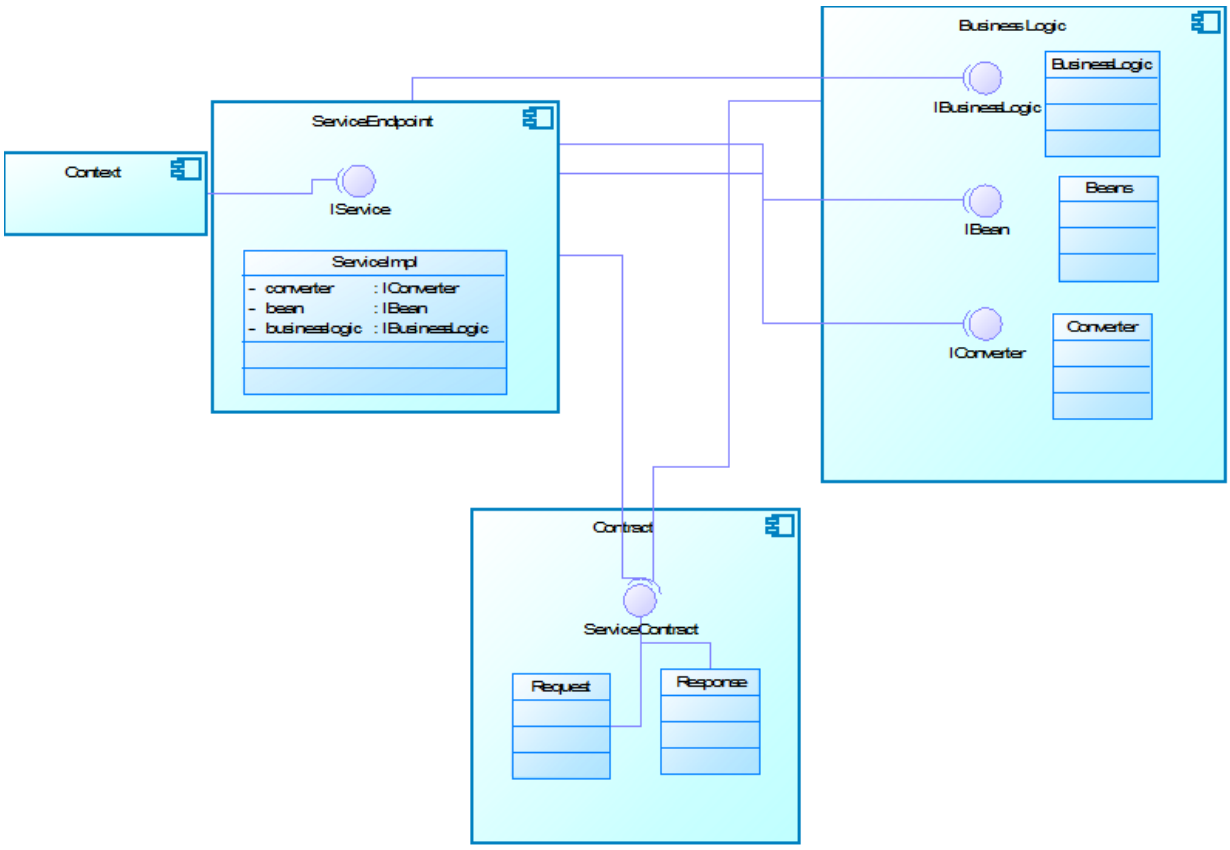

Figure 5.1: The Service Bundle Model

be used. The transaction context is usually defined at the service endpoint level. You might have to change the default “all or nothing” configuration when a data change must be applied even if the service call fails. A good example is the log-in service that must increase the unsuccessful counter when the password does not match and the business logic throws a “password does not match” exception. In this case, the business logic method must declare its own transactional demarcation rule.

Transaction Annotation

Transaction annotations can be used when the call is running in a all or nothing context. The method annotation `@Transactional` will get the existing transaction context or create a new one when currently no exists. Additional attributes of the `@Transactional` annotation are:

propagation Defines the transaction propagation type. Possible types are:

- REQUIRED Support a current transaction, create a new one if none exists.
- SUPPORTS Support a current transaction, execute non-transactionally if none exists.
- MANDATORY Support a current transaction, throw an exception if none exists.
- REQUIRES_NEW Create a new transaction, suspend the current transaction if one exists.
- NOT_SUPPORTED Execute non-transactionally, suspend the current transaction if one exists.
- NEVER Execute non-transactionally, throw an exception if a transaction exists.
- NESTED Execute within a nested transaction if a current transaction exists, behave like PROPAGATION_REQUIRED else.

The default value is REQUIRED.

isolation The transaction isolation level.

DEFAULT Use the default isolation level of the underlying database.

READ_UNCOMMITTED A constant indicating that dirty reads, non-repeatable reads and phantom reads can occur. This level allows a row changed by one transaction to be read by another transaction before any changes in that row have been committed (a "dirty read"). If any of the changes are rolled back, the second transaction will have retrieved an invalid row.

READ_COMMITTED A constant indicating that dirty reads are prevented; non-repeatable reads and phantom reads can occur. This level only prohibits a transaction from reading a row with uncommitted changes in it.

REPEATABLE_READ A constant indicating that dirty reads and non-repeatable reads are prevented; phantom reads can occur. This level prohibits a transaction from reading a row with uncommitted changes in it, and it also prohibits the situation where one transaction reads a row, a second transaction alters the row, and the first transaction rereads the row, getting different values the second time (a "non-repeatable read").

SERIALIZABLE A constant indicating that dirty reads, non-repeatable reads and phantom reads are prevented. This level includes the prohibitions in `ISOLATION_REPEATABLE_READ` and further prohibits the situation where one transaction reads all rows that satisfy a `WHERE` condition, a second transaction inserts a row that satisfies that `WHERE` condition, and the first transaction rereads for the same condition, retrieving the additional "phantom" row in the second read.

The default value is DEFAULT. Please change this only when you are really sure what you want to do. Please also note that some database management system does not support certain level of transaction isolation.

timeout The transaction time out in seconds. The default time out is -1 infinite.

rollbackFor Defines zero (0) or more exception classes, which must be a subclass of Throwable, indicating which exception types must cause a transaction roll back. If no exception class is defined (default) any exception will cause a roll back.

noRollbackFor Defines zero (0) or more exception classes, which must be a subclass of Throwable, indicating which exception types must **not** cause a transaction roll back.

Transaction Template

The Transactional annotation will not work in certain situations when mixed propagation or partly roll backs on errors are required. An example is when a error processing like sending a message is required when the main execution fails. In this situation multiple transaction templates can be used.

```
1  @Override
2     public void generateSubTxnEvent(final long txnId, final long actor) {
3     this.transactionTemplate.execute(new TransactionCallback<Object>() {
4
5         @Override
6         public Object doInTransaction(TransactionStatus status) {
7         \\ do something useful in a separate transaction context
8         return null;
9         }
10    });
11
12    }
```

Listing 5.1: “Transaction Template Example ” Transaction Template Example

Configuration in XML

To be able to use the $@\mathrm{Transactional}$ annotation approach the $<\mathrm{tx{:}annotation\text{-}driven\ transaction\text{-}manager}=\mathrm{"transactionManager"}\ />$ element needs to be configured. As you can see it is also necessary to import the transaction manager from the OSGi service registry.

In this case you shouldn't mix <aop:config /> along with the transaction annotation autoproxy.

Another way of transaction demarcation, usable more control over the demarcation is needed. For example you are doing some long running computation but only need to fetch a couple of things before you start looping (and you won't be needed lazy-init collections or the like while looping), you may opt to use a TransactionOperations instance. This is a programmatic way of handling demarcation by submitting the code to be run inside of the transaction as a callback.

A runtime exception thrown from within the callback will signal a rollback. Without an exception a rollback can only be triggered by calling TransactionStatus.setRollbackOnly();

The third and final option is to use AOP with a transaction advisor.

```
<bean id="foo" class="FooImpl" />
<tx:annotation-driven transaction-manager="transactionManager" />
```

Listing 5.2: Example XML configuration

In this case you shouldn't mix <aop:config /> along with the transaction annotation autoproxy.

If you need more control over the demarcation. For example you are doing some long running computation but only need to fetch a couple of things before you start looping (and you won't be needed lazy-init collections or the like while looping), you may opt to use a TransactionOperations instance. This is a programmatic way of handling demarcation by submitting the code to be run inside of the transaction as a callback.

```
private TransactionOperations txnTemplate;

public void doFoo() {
  // blah blah

  this.txnTemplate.execute(
    new TransactionCallback<Object>() {
      @Override
       public Object doInTransaction(final TransactionStatus status) {
           // do some transactional operation
          return null;
       }
  });
}
```

Listing 5.3: Programmatic handling transaction demarcation

```
<bean id="foo" class="FooImpl">
   <property name="txnTemplate">
      <bean class="org.springframework.transaction.support.TransactionTemplate">
        <constructor-arg ref="txManager" />
      </bean>
   </property>
</bean>
```

Listing 5.4: Spring configuration to use the transaction manager

A runtime exception thrown from within the callback will signal a rollback. You can only trigger a rollback without an exception by calling TransactionStatus .setRollbackOnly();

The third and final option is to use AOP with a transaction advisor.

So for the same example as the @Transaction, minus the annotation:

```
<bean id="foo" class="FooImpl" />

  <tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
      <tx:method name="*" propagation="REQUIRES_NEW" />
```

```
6   </tx:advice>
7
8   <aop:config>
9     <aop:pointcut id="fooOperation"
10      expression="target(FooInterface)" />
11    <aop:advisor advice-ref="txAdvice" pointcut-ref="fooOperation" />
12  </aop:config>
```

Listing 5.5: using the transaction advisor

In most cases the best choice will be using the annotations, but there will be times they are inappropriate.

5.1.2 Access Security

The service endpoint always implements a service endpoint interface. This interface is usually part of the contract bundle and not service interface. The default privilege required for access is MOBILISER_ACCESS. Each interface method can be annotated with a list of allowed roles (privileges in Mobiliser term). Use the @RolesAllowed(String[]) annotation to restrict the method access to a privilege:

```
1  package com.sybase365.mobiliser.custom.services;
2
3  import javax.annotation.security.RolesAllowed;
4
5  (...)
6
7  public interface IMyServicesEndpoint {
8
9      @RolesAllowed(value = { "WS_GET_SOMETHING" })
10     GetSomethingResponse getSomething(GetSomethingRequest request);
11
12 }
```

Listing 5.6: Service Endpoint Interface Example

The annotation in this example restricts the access to all callers with the privilege WS_GET_SOMETHING. The security advisor described in 5.6.1 checks the privileges provided by the customer authorisation framework when the service is called. When using the standard user manager of the Mobiliser Platform you must add the privilege to the table MOB_UMGR_PRIVILEGES. Assign the privilege to a user either direct via entry in MOB_UMGR_USER_PRIVILEGES or add it to a role via entry in MOB_UMGR_ROLE_PRIVILEGES.

The naming convention for service privilege is WS_[GET|CREATE|DELETE|UPDATE]_*entity*.

5.1.3 Error Handling

The service architecture translates all Throwables thrown by the endpoint method into error codes. The Throwable thrown by the endpoint must extends the abstract class MobiliserServiceException in order to provide an error[1] code and additional parameter/information in the response. See D.1 for a full list of predefined error codes.

If you need to throw a MobiliserServiceException you must implement the following abstract method public int getErrorCode(). This method is used by the service framework to fill the error code into the response.

The abstract class MobiliserServiceException comes with a list of constructors with different parameters too. Please take note of the following ones, because they are quite handy and allows you to return a proper exception information:

[1] The service framework returns the error code 9999 when the thrown Throwable does not extend the abstract class MobiliserServiceException. It returns the error code 9935 when an database exception is thrown. Please make sure you always return a specific exception that extends the abstract class MobiliserServiceException.

String message This *message* is displayed in the response directly. Please use this as a general description what went wrong. Example: "*The requested media type is not configured*"

Map<String,String> parameters This map is a key value map and added to the error response. Please make use of this map and add additional information to the response.
Example: map.put("mediaType",request.getMediaType())

Note: Please add as much information as possible to the exception thrown by your customized code, but no security sensitive information See the appendix "Best Practices"A for more details on security and exception handling

Example Please use this code sniplett for a service exception as example how to do it right:

```
1  package com.sybase365.mobiliser.money.businesslogic.transaction.exception;
2
3  import static com.sybase365.mobiliser.money.businesslogic.util.StatusCodes.
       ERROR_AMOUNT_DIFFERS;
4  import static com.sybase365.mobiliser.money.businesslogic.util.StatusCodes.
       ERROR_CAPTURE_AMOUNT;
5  import static com.sybase365.mobiliser.money.businesslogic.util.StatusCodes.
       ERROR_CAPTURE_CANCEL_AMOUNT;
6  import static com.sybase365.mobiliser.money.businesslogic.util.StatusCodes.
       ERROR_TRANSACTION_PERMANENT;
7
8  import java.util.Map;
9
10 import com.sybase365.mobiliser.framework.service.api.MobiliserServiceException;
11
12
13 public class TransactionAmountException extends MobiliserServiceException {
14
15     // TODO add more constructors for ease of use
16
17     /** serial version uid. */
18     private static final long serialVersionUID = 1573064007401098166L;
19
20     /**
21      *
22      * <p>
23      * &copy; 2012 by Sybase Inc., an SAP Company
24      * </p>
25      *
26      */
27     public static enum Reason {
28     /**  */
29     CAPTURE_CANCEL_AMOUNT,
30     /**      */
31     AMOUNT_DIFFERS,
32     /** */
33     CAPTURE_AMOUNT;
34     }
35
36     private final Reason reason;
37
38
39     /**
40      *
41      * @param message
42      * @param reason
43      * @param parameters
44      */
45     public TransactionAmountException(String message, Reason reason,
46         Map<String, String> parameters) {
```

```
47      super(message, parameters);
48      this.reason = reason;
49      }
50
51
52      @Override
53      public int getErrorCode() {
54      switch (this.reason) {
55      case CAPTURE_CANCEL_AMOUNT:
56          return ERROR_CAPTURE_CANCEL_AMOUNT;
57      case CAPTURE_AMOUNT:
58          return ERROR_CAPTURE_AMOUNT;
59      case AMOUNT_DIFFERS:
60          return ERROR_AMOUNT_DIFFERS;
61      }
62
63      return ERROR_TRANSACTION_PERMANENT;
64      }
65
66  }
```

Listing 5.7: Service Exception Example

Custom Error Codes If you need to add custom error codes you can do it the following way:

1. Create a ClassCustomStatusCodes class.
2. Add a new error code constant via the @MobiliserErrorCodeDescription annotation the following way:

```
1  (...)
2      private static final int ERROR_GROUP_CUSTOM = 10000;
3
4      @MobiliserErrorCodeDescription(description = "This is a custom error
           code", infoLevel = InfoMode.INFO)
5      public static final int ERROR_CUSTOM_EXAMPLE = ERROR_GROUP_CUSTOM + 1;
```

Listing 5.8: Custom Error Code Example

   Note: Error code from 0 to 9999 are Mobiliser pre-defined and therefore custom error codes must be greater than 9999

3. Do not forget to add the custom error code to the database table MOB_ERROR_CODES if it is thrown during transaction processing, because the transaction error code field has a foreign key reference to the error code table.

5.2 Context

A Mobiliser Service Context aggregates endpoints (services) into logical groups. When talking about SOAP services, this is the list of operations available in a WSDL and accessible through a certain http context. For the restful services, the operations are dispatched based on suffixes and these are grouped under a common context as well. The service context also defines an HTTP privilege which is required by clients accessing any of the services registered here.

The framework comes with a context configuration which can be reused. Each context only needs to supply the properties needed to defined the context paths and WSDL properties mentioned above as well as the security privileges needed. This configuration is built into your bundle file during compilation.

Our configuration will be a properties file like this:

```
1  #Audit Properties
```

```
2 contextPath=audit
3 wsdlName=Audit
4 portTypeName=AuditSoapPort
5 serviceName=AuditService
6 namespace=http\://mobiliser.sybase365.com/money/audit
7 jsClientName=auditService
```

Listing 5.9: context.properties for “audit” context

```
/
├── pom.xml
└── src
    └── main
        └── resources
            └── META-INF
                └── spring
                    └── context.properties
```

Figure 5.2: Example directory tree for service contexts

For our example “audit”, the default configuration will then register a SOAP context called audit underneath the Mobiliser servlet, so /mobiliser/audit and a dynamic WSDL accessible through /mobiliser/audit/Audit.wsdl. It will register a mapping for restful endpoints under /mobiliser/rest/audit supporting content types application/xml and application/json. A javascript client is also registered and is accessible under /mobiliser/rest/audit?js.

The default privilege required for access is $\mathrm{MOBILISER\_ACCESS}$ which we can override by setting $\mathrm{MAIN\_ACCESS\_PRIVS}$ to another value in the $\mathrm{context.properties}$ file. It is a comma-separated list of “OR” privileges (which means the client must have at lease one of them, but not all).

These pieces are exported to the OSGi registry and picked up by the gateway bundles. Our context then listens for IEndpointInformation object exported to the registry with the path attribute set to our context name (in this case to “audit”). So from the gateway’s point of view, a single mapping/adapter is registered for rest and a soap message dispatcher. Internally these pieces aggregate all of the endpoints we have for that context.

The maven pom.xml for a context is boilerplate code:

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
    /2001/XMLSchema-instance"
2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
      maven-v4_0_0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <parent>
5     <groupId>com.sybase365.mobiliser.money</groupId>
6     <artifactId>com.sybase365.mobiliser.money.context</artifactId>
7     <version>5.0.0.RELEASE</version>
8   </parent>
9   <groupId>com.sybase365.mobiliser.money</groupId>
10  <artifactId>com.sybase365.mobiliser.money.context.audit</artifactId>
11  <name>AIMS Mobiliser :: Money :: Service Context Audit</name>
12  <packaging>bundle</packaging>
13  <build>
14    <resources>
15      <resource>
16        <directory>${basedir}/src/main/resources</directory>
17      </resource>
18      <resource>
19        <directory>${basedir}/target/context-configuration</directory>
20      </resource>
21    </resources>
22    <plugins>
23      <plugin>
24        <groupId>org.apache.maven.plugins</groupId>
25        <artifactId>maven-dependency-plugin</artifactId>
26        <executions>
```

```
27            <execution>
28              <id>unpack</id>
29              <phase>process-resources</phase>
30              <goals>
31                <goal>unpack</goal>
32              </goals>
33              <configuration>
34                <artifactItems>
35                  <artifactItem>
36                    <groupId>com.sybase365.mobiliser.framework</groupId>
37                    <artifactId>com.sybase365.mobiliser.framework.gateway.context</
                          artifactId>
38                    <version>${version.framework}</version>
39                    <type>jar</type>
40                    <overWrite>true</overWrite>
41                    <outputDirectory>${basedir}/target/context-configuration/META-
                          INF/spring</outputDirectory>
42                    <includes>**/bundle*.xml</includes>
43                  </artifactItem>
44                </artifactItems>
45              </configuration>
46            </execution>
47          </executions>
48        </plugin>
49        <plugin>
50          <groupId>org.apache.felix</groupId>
51          <artifactId>maven-bundle-plugin</artifactId>
52          <extensions>true</extensions>
53          <configuration>
54            <instructions>
55              <Bundle-Category>context</Bundle-Category>
56              <Export-Package></Export-Package>
57              <Private-Package></Private-Package>
58              <Import-Package>
59                com.sybase365.mobiliser.framework.gateway.api,
60                com.sybase365.mobiliser.framework.gateway.soap.ws.wsdl,
61                com.sybase365.mobiliser.framework.gateway.security.api,
62                org.springframework.ws.server,
63                org.springframework.ws.soap.server.endpoint,
64                org.springframework.ws.wsdl.wsdl11,
65                org.springframework.ws.transport,
66                org.springframework.security.web.authentication,
67                org.springframework.security.authentication,
68                org.springframework.security.web.authentication.www,
69                org.springframework.security.web.authentication.rememberme,
70                org.springframework.security.web.access.intercept,
71                org.springframework.security.access.vote,
72                org.springframework.security.web.access.expression,
73                org.springframework.security.access.intercept,
74                org.springframework.transaction,
75                org.springframework.web.servlet,
76                <!-- start aop proxy stuff -->
77                org.aspectj.weaver.reflect,
78                org.springframework.aop.framework,
79                org.springframework.aop,
80                org.aopalliance.aop,
81                org.springframework.aop.aspectj.autoproxy,
82                 <!-- add to allow resolving with 3.0.0 servlet api -->
83                javax.servlet;version="[2.4.0, 3.0.0]"
84                ,javax.servlet.http;version="[2.4.0, 3.0.0]"
85                ,javax.servlet.resources;version="[2.4.0, 3.0.0]"
86                ,*
87              </Import-Package>
88            </instructions>
89          </configuration>
```

```
90        </plugin>
91      </plugins>
92    </build>
93    <dependencies>
94      <dependency>
95        <groupId>com.sybase365.mobiliser.framework</groupId>
96        <artifactId>com.sybase365.mobiliser.framework.gateway.soap.ws</artifactId>
97        <version>${version.framework}</version>
98      </dependency>
99      <dependency>
100       <groupId>com.sybase365.mobiliser.framework</groupId>
101       <artifactId>com.sybase365.mobiliser.framework.gateway.rest</artifactId>
102       <version>${version.framework}</version>
103     </dependency>
104     <dependency>
105       <groupId>com.sybase365.mobiliser.framework</groupId>
106       <artifactId>com.sybase365.mobiliser.framework.gateway.security.api</
              artifactId>
107       <version>${version.framework}</version>
108     </dependency>
109     <dependency>
110       <groupId>org.springframework.security</groupId>
111       <artifactId>org.springframework.security.config</artifactId>
112       <scope>runtime</scope>
113     </dependency>
114     <dependency>
115       <groupId>com.sybase365.mobiliser.util.tools</groupId>
116       <artifactId>com.sybase365.mobiliser.util.tools.spring</artifactId>
117       <version>${version.mobiliser-tools}</version>
118     </dependency>
119     <dependency>
120       <groupId>org.aspectj</groupId>
121       <artifactId>com.springsource.org.aspectj.runtime</artifactId>
122       <scope>runtime</scope>
123       <!-- proxies -->
124     </dependency>
125   </dependencies>
126 </project>
```

Listing 5.10: pom.xml for “audit” context

5.3 Contract

Contract bundles hold the XML schema definitions (XSDs) defining both, beans and requests on these beans. During the Maven build, the XSDs get compiled into Java beans and bundled up into an OSGi compatible contract bundle. It also should define the service interface with the methods. This service interface is implemented by the service endpoint. Please also see the service endpoint for access security annotations.

5.3.1 Layout

This is the default layout of a contract bundle as created by the Maven Archetype. The pom.xml is preconfigured to generate a proper OSGi bundle, the beans and request XSDs are placed into the default location, the namespace in the XSDs matches this location.

The convention for XSD namespaces is, that all namespaces must start with:

```
http://mobiliser.sybase365.com/money/contract/v5_0/
```

followed by the contract bundle name, e.g. ’custom’, followed by either ’beans’ or ’requests’, for XSDs defining

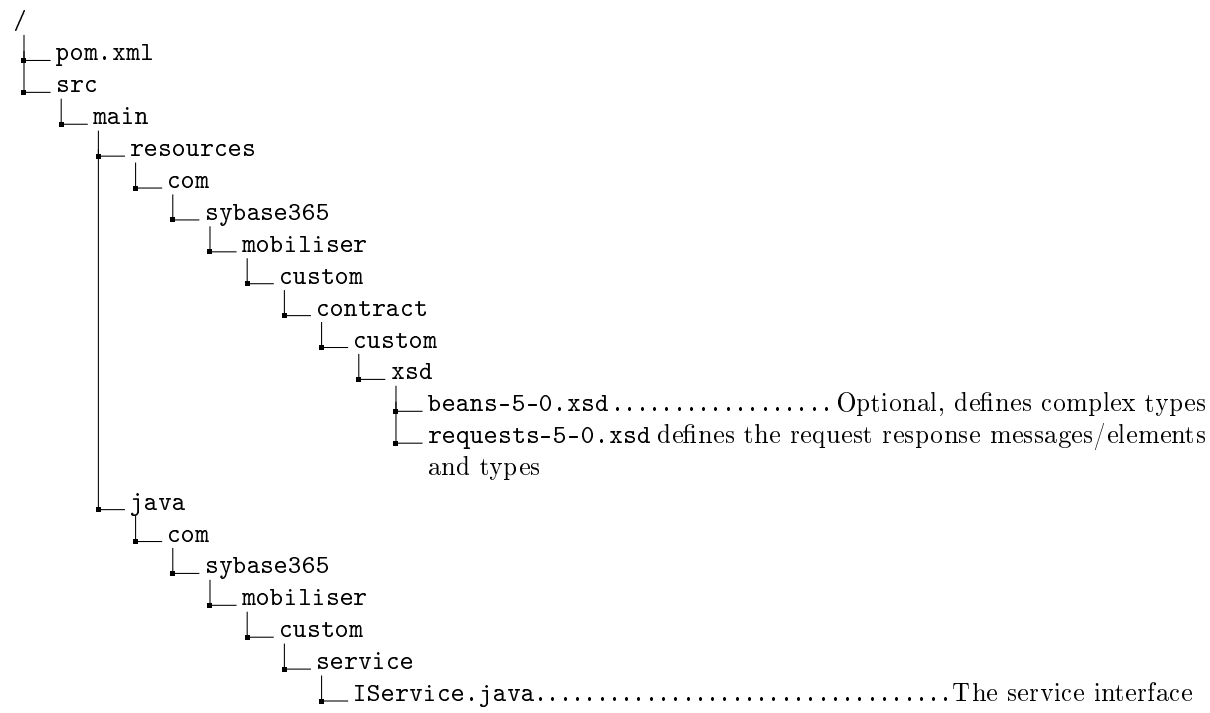

Figure 5.3: Example directory tree for service contracts

general purpose beans or request/response elements, respectively. The folder structure of the XSDs must also match a pattern for easy resource location through the framework. Any XSDs must be located at :

`/com/sybase365/mobiliser/money/contract/<bundle name>/xsd/.`

All XSDs must end with the defined version (that must match the version that is part of the namespace). In order to organize the files, it is suggested to split different request categories into different `*-requests-5-0.xsd` files (but still, all requests must be in the same namespace as imposed by the defined structure).

5.3.2 Beans XSD

The beans XSD defines complex types, which will be used in request and response types and elements. Any data structure that is not a simple type must be defined separately in the beans.xsd and must not be part of an anonymous inner type declaration in the requests.xsd.

The Mobiliser base XSD defines a few common base types, which enforce restrictions on generic datatypes like longs, ints, and strings. These base types map to restrictions we are using in the database and you must make sure that the length restrictions in the bean match the database limitations for the field in the database that will ultimately receive this value (if applicable). Additionally, also be precise on the min and max occurrences of fields; a field must be mandatory if it is mandatory in the database as well and there's no default value for it configured in the database.

The Mobiliser base types include

strSmall 6 character string

strMedium - 80 character string

strLarge 200 character string

strHuge 2048 character string

idShort 5 figure positive number incl. zero

idLong 18 figure positive number incl. zero

Use the documentation tag to add in-line documentation to the beans as well as to their attributes.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0"
  targetNamespace="http://mobiliser.sybase365.com/money/contract/v5_0/custom/
      beans"
  xmlns:base="http://mobiliser.sybase365.com/framework/contract/v5_0/base"
  xmlns="http://mobiliser.sybase365.com/money/contract/v5_0/custom/beans"
  xmlns:jxb="http://java.sun.com/xml/ns/jaxb" xmlns:xs="http://www.w3.org/2001/
      XMLSchema"
  xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
  jxb:extensionBindingPrefixes="xjc" jxb:version="2.0">

  <xs:annotation>
    <xs:appinfo>
      <jxb:schemaBindings>
        <jxb:package
          name="com.sybase365.mobiliser.money.contract.v5_0.custom.beans" />
      </jxb:schemaBindings>
    </xs:appinfo>
    <xs:documentation>
      <![CDATA[The XML Schema for Custom beans. Version:
      $HeadURL: $
      ]]>
    </xs:documentation>
  </xs:annotation>

  <xs:import
    namespace="http://mobiliser.sybase365.com/framework/contract/v5_0/base"
    schemaLocation="../../../../framework/contract/xsd/base-5-0.xsd" />

  <xs:complexType name="CustomBean">
    <xs:annotation>
      <xs:documentation>
        <![CDATA[Holds information on Custom.]]>
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="id" type="base:idLong" minOccurs="0" />
      <xs:element name="customerId" type="base:idLong"
        minOccurs="0" />
      <xs:element name="type" type="base:idShort">
        <xs:annotation>
          <xs:documentation>
            <![CDATA[Blabla.]]>
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="active" type="xs:boolean" />
      <xs:element name="text" type="base:strMedium" />
      <xs:element name="status" type="base:idLong" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

Listing 5.11: “XSD Beans” used to define types used in multiple request/response types

5.3.3 Requests XSD

Each request/response pair is defined by four definitions in the requests.xsd file. There is a RequestType-/ResponseType pair of complex types defining the actual content of request and response. These elements

must (directly or indirectly) extend MobiliserRequestType/MobiliserResponseType. It is important to use the correct type for the request definition, use MobiliserRequestType of all selector services (services that will not change the instances reference by the request) or TraceableRequestType for modifying services. The TraceableRequestType makes the origin and traceNo attribute mandatory and therefore ensures that requests are not submitted multiple times.

Any nested complex types defining richer data structures must be outsourced into the beans.xsd and imported from there. The sequence defined in the ResponseType complex type must have the minOccurs="0" attribute set; in case the service call ends with an exception, the response might not carry any data, but just an error code, which is set to the MobiliserResponseType super type by the Mobilier framework classes. Additionally, there is a Request/Response pair of elements, which do extend the RequestType/ResponseType and must not add any further elements to it. Use the documentation tag to add in-line documentation to the request and response types, as well as to their attributes.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <xs:schema version="1.0"
3    targetNamespace="http://mobiliser.sybase365.com/money/contract/v5_0/custom"
4    xmlns="http://mobiliser.sybase365.com/money/contract/v5_0/custom"
5    xmlns:beans="http://mobiliser.sybase365.com/money/contract/v5_0/custom/beans"
6    xmlns:base="http://mobiliser.sybase365.com/framework/contract/v5_0/base"
7    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:jxb="http://java.sun.com/xml/
       ns/jaxb"
8    xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
9    jxb:extensionBindingPrefixes="xjc" jxb:version="2.0">
10   <xs:annotation>
11     <xs:appinfo>
12       <jxb:schemaBindings>
13         <jxb:package
14           name="com.sybase365.mobiliser.money.contract.v5_0.custom" />
15       </jxb:schemaBindings>
16     </xs:appinfo>
17     <xs:documentation>
18       <![CDATA[The XML Schema for Custom requests. Version:
19       $HeadURL: $
20       ]]>
21     </xs:documentation>
22   </xs:annotation>
23
24   <xs:import
25     namespace="http://mobiliser.sybase365.com/framework/contract/v5_0/base"
26     schemaLocation="../../../../framework/contract/xsd/base-5-0.xsd" />
27   <xs:import
28     namespace="http://mobiliser.sybase365.com/money/contract/v5_0/custom/beans"
29     schemaLocation="beans-5-0.xsd" />
30
31   <xs:complexType name="CustomBeanRequestType">
32     <xs:annotation>
33       <xs:documentation>
34         <![CDATA[This is the Custom request, doing important stuff.]]>
35       </xs:documentation>
36     </xs:annotation>
37     <xs:complexContent>
38       <xs:extension base="base:MobiliserRequestType">
39         <xs:sequence>
40           <xs:element name="CustomElement" type="beans:CustomBean">
41             <xs:annotation>
42               <xs:documentation>
43               <![CDATA[Takes Custom information.]]>
44               </xs:documentation>
45             </xs:annotation>
46           </xs:element>
47         </xs:sequence>
48       </xs:extension>
49     </xs:complexContent>
```

```
50   </xs:complexType>
51
52   <xs:complexType name="CustomBeanResponseType">
53     <xs:annotation>
54       <xs:documentation>
55         <![CDATA[This is the Custom response, returning important stuff.]]>
56       </xs:documentation>
57     </xs:annotation>
58     <xs:complexContent>
59       <xs:extension base="base:MobiliserResponseType">
60         <xs:sequence minOccurs="0">
61           <xs:element name="CustomElement" type="beans:CustomBean">
62             <xs:annotation>
63               <xs:documentation>
64                 <![CDATA[Returns Custom information, some data is updated from
                       the request.]]>
65               </xs:documentation>
66             </xs:annotation>
67           </xs:element>
68         </xs:sequence>
69       </xs:extension>
70     </xs:complexContent>
71   </xs:complexType>
72
73   <xs:element name="CustomBeanRequest">
74     <xs:complexType>
75       <xs:complexContent>
76         <xs:extension base="CustomBeanRequestType" />
77       </xs:complexContent>
78     </xs:complexType>
79   </xs:element>
80
81   <xs:element name="CustomBeanResponse">
82     <xs:complexType>
83       <xs:complexContent>
84         <xs:extension base="CustomBeanResponseType" />
85       </xs:complexContent>
86     </xs:complexType>
87   </xs:element>
88
89 </xs:schema>
```

Listing 5.12: “XSD Request/Response” types and element definition for the service

5.3.4 POM configuration

In case the bundle uses the default layout, the POM does not require any modifications at all. If you add further XSDs in different namespaces (even though this is not recommended - rather create a new contract bundle for this), you want to make sure that you adjust the export-package declaration if required, and also the exclude pattern for the maven-clean-plugin. Also, if you have to import additional dependencies because you are extending existing beans from another namespace, you have to also add it into the maven-dependency-plugin for automatic extraction into the resource folder, and remember to adjust the import-package declaration to add the path for the required XSDs.

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
      /2001/XMLSchema-instance"
2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
      maven-v4_0_0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <parent>
5     <groupId>com.sybase365.mobiliser.money</groupId>
6     <artifactId>com.sybase365.mobiliser.money.contract</artifactId>
```

```
    <version>5.0.0-SNAPSHOT</version>
  </parent>
  <groupId>com.sybase365.mobiliser.money</groupId>
  <artifactId>com.sybase365.mobiliser.money.contract.custom</artifactId>
  <packaging>bundle</packaging>
  <build>
    <resources>
      <resource>
        <directory>${basedir}/src/main/resources</directory>
        <excludes>
          <exclude>com/sybase365/mobiliser/framework/*</exclude>
          <exclude>com/sybase365/mobiliser/framework/contract/xsd/*.xsd</exclude>
        </excludes>
      </resource>
    </resources>
    <plugins>
      <plugin>
        <artifactId>maven-clean-plugin</artifactId>
        <configuration>
          <filesets>
            <fileset>
              <directory>src/main/resources</directory>
              <excludes>
                <exclude>**/.svn/**</exclude>
                <exclude>**/custom/xsd/*</exclude>
              </excludes>
              <followSymlinks>false</followSymlinks>
            </fileset>
          </filesets>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <executions>
          <!-- unpack the referenced schema to a tmp directory. -->
         <execution>
            <id>unpack</id>
            <phase>generate-sources</phase>
            <goals>
              <goal>unpack</goal>
            </goals>
            <configuration>
              <artifactItems>
                <artifactItem>
                  <groupId>com.sybase365.mobiliser.framework</groupId>
                  <artifactId>com.sybase365.mobiliser.framework.contract</
                      artifactId>
                  <version>${project.version}</version>
                  <type>jar</type>
                  <overWrite>true</overWrite>
                  <outputDirectory>${basedir}/src/main/resources</outputDirectory
                      >
                  <includes>**/*.xsd</includes>
                </artifactItem>
              </artifactItems>
            </configuration>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <groupId>com.sun.tools.xjc.maven2</groupId>
        <artifactId>maven-jaxb-plugin</artifactId>
        <executions>
          <!-- generate the files. -->
```

```
70            <execution>
71              <goals>
72                <goal>generate</goal>
73              </goals>
74            </execution>
75          </executions>
76          <configuration>
77            <includeSchemas>
78              <includeSchema>**/*.xsd</includeSchema>
79            </includeSchemas>
80            <verbose>true</verbose>
81            <extension>true</extension>
82          </configuration>
83        </plugin>
84        <plugin>
85          <artifactId>maven-antrun-plugin</artifactId>
86          <executions>
87            <execution>
88              <phase>process-sources</phase>
89              <goals>
90                <goal>run</goal>
91              </goals>
92              <configuration>
93                <tasks>
94                  <delete
95                    dir="${project.build.directory}/generated-sources/xjc/com/
                          sybase365/mobiliser/framework" />
96                </tasks>
97              </configuration>
98            </execution>
99          </executions>
100       </plugin>
101       <plugin>
102         <groupId>org.apache.felix</groupId>
103         <artifactId>maven-bundle-plugin</artifactId>
104         <extensions>true</extensions>
105         <configuration>
106           <instructions>
107             <Export-Package>com.sybase365.mobiliser.money.contract.custom.xsd,
108               com.sybase365.mobiliser.money.contract.v5_0.custom,
109               com.sybase365.mobiliser.money.contract.v5_0.custom.beans</Export-
                    Package>
110             <Private-Package></Private-Package>
111             <Import-Package>com.sybase365.mobiliser.framework.contract.xsd,*</
                  Import-Package>
112           </instructions>
113         </configuration>
114       </plugin>
115     </plugins>
116   </build>
117   <dependencies>
118     <dependency>
119       <groupId>com.sybase365.mobiliser.framework</groupId>
120       <artifactId>com.sybase365.mobiliser.framework.contract</artifactId>
121       <version>${project.version}</version>
122     </dependency>
123   </dependencies>
124 </project>
```

Listing 5.13: “OSGI Configuration” for the contract

5.4 OData

Mobiliser allows automatically exporting CRUD operations through OData if some extra metadata is added to the DTO's generated from the XML schema definitions. The integration layer can only expose operations through OData if a standard Mobiliser service already exists for the operation. Since many entities in Mobiliser already support standard CRUD operations, good coverage of standard OData operations can be achieved by only adding extra configuration.

There are several annotations defined in the package *com.sybase365.mobiliser.framework.contract* which provide extra information for Odata. The following is an overview, but the annotation will be discussed in detail later.

- Entity - marks an class as being an Entity
- Id - defines a field being the key (or part of a composite key) for an entity
- EntityReference - marks a field of a primitive type as being a reference to another entity (like a foreign key)
- CreateOperation - operation for creating instances of an entity
- DeleteOperation - operation for deleting instances of an entity
- UpdateOperation - operation for updating instances of an entity
- GetOperation - operation for fetching an entity using its id
- GetAllOperation - operation for fetching all entities of a type or a subset of that type using search criteria.
- Criterion - marks a field as being a search criteria
- CriteriaField - marks a field being a holder for search criteria
- EntityFieldReference - marks a field in an operation request as referencing an field in an entity, which itself is an entity reference.
- EntityRequestField - marks a field as holding the entity for a request (like the entity to update for an update request)
- IdReponseField -. marks the field as holding the id in a response object (like the newly generated ID after a create request)
- IdRequestField - marks the field as holding the id in a request object (like the ID to fetch in a get request)
- OperationResultField - marks the field as holding the result of an operation (like the search result list of a getall operation)

5.4.1 Basics

Every 'type' (usually those in beans.xsd) needs to be marked with *@Entity* so that the odata integration knows which types are entity types. The id field(s) of that entity should then be marked with *@Id* or one of the following conditions must be fulfilled (searched in this order):

- the entity has a field named 'typeId' - where type is the entity type with a leading lowercase letter. As an example, if my entity is called *Department*, the field for the id would be *departmentId*.
- the entity simply has a field named 'id'

There are currently 5 operation types. These will always reference an entity type you have already defined.

- GetOperation - maps to an HTTP GET for a single entity

- GetAllOperation - maps to an HTTP get for an entitySet (can possibly be combined with query values)
- UpdateOperation - maps to an HTTP PUT for a single entity
- CreateOperation - maps to an HTTP POST for a single entity
- DeleteOperation - maps to an HTTP DELETE for a single entity

All operation type annotations must be placed on the complex type for the request not the element itself. The JAXB plugin we are using will fail to add the annotation to the type created for the element. (So put it on CreateXXXRequestType and not on CreateXXXRequest)

The annotations are typically not included in the XSD itself since we do not want to pollute the standard XSDs for clients using them, but not interested in OData. Instead we use a maven plugin to add the annotations directly to our generated beans at build time.

To achieve this, we change standard build described above for generating the JAXB classes like this:

```
...
      <plugin>
        <groupId>com.sun.tools.xjc.maven2</groupId>
        <artifactId>maven-jaxb-plugin</artifactId>
        <executions>
          <!-- generate the files. -->
          <execution>
            <goals>
              <goal>generate</goal>
            </goals>
          </execution>
        </executions>
        <configuration>
          <includeSchemas>
            <includeSchema>**/*.xsd</includeSchema>
          </includeSchemas>
          <includeBindings>
            <includeBinding>**/*.xjb</includeBinding>
          </includeBindings>
          <verbose>true</verbose>
          <extension>true</extension>
          <args>-Xannotate</args>
        </configuration>
        <dependencies>
          <dependency>
            <groupId>com.sybase365.mobiliser.framework</groupId>
            <artifactId>com.sybase365.mobiliser.framework.contract</artifactId>
            <version>${version.framework}</version>
          </dependency>
        </dependencies>
      </plugin>
...
```

Listing 5.14: “OData POM” for the contract

The bindings are standard binding files for XJC but reference the annox plugin to add the annotations:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<jaxb:bindings version="2.1" xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:annox="http://annox.dev.
    java.net" jaxb:extensionBindingPrefixes="annox">

  <!-- entities -->
  <jaxb:bindings schemaLocation="beans-5-0.xsd" node="/xs:schema">
    <jaxb:bindings node="xs:complexType[@name='FeeSet']">
      <annox:annotate target="class">
```

```
 8          <annox:annotate annox:class="com.sybase365.mobiliser.framework.contract.
                Entity" />
 9        </annox:annotate>
10      </jaxb:bindings>
11    </jaxb:bindings>
12
13    <jaxb:bindings schemaLocation="system-requests-5-0.xsd" node="/xs:schema">
14      <jaxb:bindings node="xs:complexType[@name='CreateFeeSetRequestType']">
15        <annox:annotate target="class">
16          <annox:annotate annox:class="com.sybase365.mobiliser.framework.contract.
                CreateOperation" entityType="com.sybase365.mobiliser.money.contract.
                v5_0.system.beans.FeeSet" />
17        </annox:annotate>
18      </jaxb:bindings>
19      <jaxb:bindings node="xs:complexType[@name='CreateFeeSetResponseType']/
            xs:complexContent/xs:extension/xs:sequence/xs:element[@name='feeSetId']">
20        <annox:annotate target="field">
21          <annox:annotate annox:class="com.sybase365.mobiliser.framework.contract.
                IdReponseField" />
22        </annox:annotate>
23      </jaxb:bindings>
24      <jaxb:bindings node="xs:complexType[@name='GetFeeSetsRequestType']">
25        <annox:annotate target="class">
26          <annox:annotate annox:class="com.sybase365.mobiliser.framework.contract.
                GetAllOperation" entityType="com.sybase365.mobiliser.money.contract.
                v5_0.system.beans.FeeSet"/>
27        </annox:annotate>
28      </jaxb:bindings>
29      <jaxb:bindings node="xs:complexType[@name='DeleteFeeSetRequestType']">
30        <annox:annotate target="class">
31          <annox:annotate annox:class="com.sybase365.mobiliser.framework.contract.
                DeleteOperation" entityType="com.sybase365.mobiliser.money.contract.
                v5_0.system.beans.FeeSet"/>
32        </annox:annotate>
33      </jaxb:bindings>
34      <jaxb:bindings node="xs:complexType[@name='UpdateFeeSetRequestType']">
35        <annox:annotate target="class">
36          <annox:annotate annox:class="com.sybase365.mobiliser.framework.contract.
                UpdateOperation" entityType="com.sybase365.mobiliser.money.contract.
                v5_0.system.beans.FeeSet"/>
37        </annox:annotate>
38      </jaxb:bindings>
39    </jaxb:bindings>
40  </jaxb:bindings>
```

Listing 5.15: “OData Binding FIle” for the contract

5.4.2 GET Operation

The only information we get from the OData integration is the ID and the entity. To make these operations automatically work, mark the following:

- The request type should be marked with *@GetOperation.*
- The field where the ID(s) should be set on the request object should be marked with *@IdRequestField* or the fields must match the names of the id fields from the entity itself, there must be a field called entityId where entity is the entity type in lower case or there must be a field called ’id’. (searched in that order)
- The fetched entity will be set on the reponse object on the field marked with *@OperationResultField* or if the field is called ’type’ where type is the lower case entity name or on the field with the name of the response type after removing the leading Get and stripping the trailing Response and changing the leading character to lowercase (in that order) nothing is required. Example of the first would be the field

department for the type com.sybase365.blibb.Department. An example for the second would be a field departmentSingle for the response type GetDepartmentSingleResponse.

5.4.3 GetAll Operation

The only information we get from the OData integation is the entity. To make these operations automatically work, mark the following:

- The request type should be marked with *@GetAllOperation*
- The fetched entities will be found in the field marked with *@OperationResultField*, or if no annotated field is found, it will look for a field starting with lowercase with the entity named in the plural (so for the Department entity, will look for a field departments) or a field with the name of the response type after removing the leading Get/GetAll and removing the trailing Response (so for the response type GetAllDepartmentsEntityResponse, the field departmentsEntity)

Criteria Support

Currently you can do a search operation an entity set if you have a standard request which searches based on a 'criteria' object. This means you have a request with a single field of a complex type and on this complex type you set your search criteria. Each criteria will map to a field in the entity and to an operation (equals, less-than, etc) This is probably explained best with an example:

```
1 FindDepartmentRequestType
2 searchCriteria
3
4 DepartmentSearchCriteria
5 name
6 dateFrom
7 dateTo
```

Listing 5.16: Search Criteria Example

We have several such services already.

To automatically make this work for OData, you need to do as above and mark this as a *@GetAllOperation* and then additionally mark the criteria field with *@CriteriaField*. That will change the getAll into a search operation. Additionally you need to annotate the fields of the criteria object class with *@Criterion* which will include the information about which entity field this maps to and which operation.

e.g.:

```
1 @Criterion(field = "name", operation = "=")
2 private String name;
3 @Criterion(field = "creationDate", operation = "<=")
4 private XMLGregorianCalendar dateTo;
```

Listing 5.17: Search Criteria Object Example

If a client requests something which the criteria object does not support, he'll get back a 501 not implemented from the server. Unfortunately what is actually allowed does not show up in the CSDL so in this case, the client must know a bit about the traditional mobiliser schemas.

The other option for searching is to not have a criteria object, but annotate the fields of the operation directly with *@Criterion*. This is probably also explained best with an example:

```
1 FindDepartmentRequestType
2 name
3 dateFrom
```

```
4 dateTo
```

Listing 5.18: Search Criteria Example

```
1 Then the fields should have annotations like:
2 e.g.:
3 @Criterion(field = "name", operation = "=")
4 private String name;
5 @Criterion(field = "creationDate", operation = "<=")
6 private XMLGregorianCalendar dateTo;
```

Listing 5.19: Search Criteria Example

Note that it is possible to have two GetAll operations - one which supports some criteria and one which simply returns all entities of the entity type. This will only be the case with small entity sets. For large sets, you will only have one which allows searching.

Also note that if you have a GetAll operation without criteria and the entity set does not have a simple Get operation, the OData interface will emulate one by using the GetAll to fetch all the entities and then use the primary key to find the actual entity in the list returned from the GetAll operation.

The only information we get from the OData integration is the ID and the entity. To make these operations automatically work, mark the following:

- The request type should be marked with *@DeleteOperation*.
- The field where the ID should be set on the request object should be marked with *@IdRequestField* or in the same fashion as the id field for a GET request.

5.4.4 PUT Operation (update)

The only information we get from the OData integration is the entity set and the new entity itself. To make these operations automotically work, mark the following:

- The request type should be marked with *@UpdateOperation*
- The field on the request type holding the new entity should be marked with *@EntityRequestField* or with the name 'entity' for a given Entity (so 'department' for the entity com.sybase365.blibb.Deparment) or the name of the request with the leading Update and the trailing Request removed (so for UpdateDepartmentRequest it would be department.

5.4.5 POST Operation (create)

The only information we get from the OData integration is the entity set and the the new entity itself. To make these operations automatically work, mark the following:

- The request type should be marked with *@CreateOperation*
- The field on the request holding the new entity should be marked in the same manner as PUT operations.
- If the entity type uses autogenerated primary key, you need to mark the field in the response type which will hold the new primary key with *@IdReponseField*. This is REQUIRED.
- The OData interface uses the GET Operation to return the newly created entry. Therefore each POST Operation requires a GET Operation

Special Note: There are some create operations which are not modeled this way. For those, the request itself has the fields of the actual entity instead of filling an entity and setting it on the request object. These can also be handled by the integation layer. It is probably best explained with an example:

Split entity creation:

```
CreateFeeTypeRequestType
  feeType
  beneficiaryId
```

Listing 5.20: Split Entity Creation Example

The fee type itself has an exploded beneficiary normally, so it seems this request was written so that you send along the ID instead of the customer. In OData the create request looks like this:

```
{
    "id": 9876,
    "commission": false,
    "includeVat": false,
    "name": "soapui test fee type",
    "processWithTxn": true,
    "beneficiary": {"__metadata": {"uri": "\${Project#mobiliser.url}/money_
        contract_v5_0_customer_beans_Customer(204L)"}}
}
```

Listing 5.21: FeeType Creation OData Example

The annotations should then be like this in CreateFeeTypeRequestType:

```
    @EntityFieldReference(entityField = "beneficiary")
    protected long beneficiaryId;
```

Listing 5.22: FeeType CreationRequest Annotations

This tells the integration layer that this field in the request actually maps to the entity field beneficiary. So when the request comes in through Odata, it takes the id that you sent along and sets it on this field in the JAXB request so that the normal mobiliser service works as it should.

A separate example is one that doesn't have any entity field at all in the create request, but instead all fields inlined:

```
CreateLimitSetClassRequestType
  limitSetId;
  limitClassId;
  useCaseId
```

Listing 5.23: LimitSetClass Request

And again the create request:

```
{
    "useCaseId": 160,
    "limitSet": {"__metadata": {"uri": "\${Project#mobiliser.url}/money_contract_
        v5_0_system_beans_LimitSet(103L)"}},
    "limitClass": {"__metadata": {"uri": "\${Project#mobiliser.url}/money_contract
        _v5_0_system_beans_LimitClass(105L)"}}
}
```

Listing 5.24: LimitSetClass Creation OData Example

```
And the annotated JAXB bean CreateLimitSetClassRequestType:

    @EntityFieldReference(entityField = "limitSet")
    protected long limitSetId;
    @EntityFieldReference(entityField = "limitClass")
    protected long limitClassId;
```

```
7       @EntityFieldReference(entityField = "useCaseId")
8       protected Integer useCaseId;
```

Listing 5.25: Annotated CreateLimitSetClassRequestType

Hopefully these examples are now clear.

5.4.6 MERGE Operation

Mobiliser does not currently export what you could call a merge operation throught standard services. For the OData integration, a MERGE operation is provided for entity sets which define both a GET and a POST operation. Internally the entity will be fetched with GET, the fields from the MERGE request will be set on the entity and then it will call the update operation.

5.4.7 Links

Now for links between entities. OData deals with relations using <link> in atom+xml. We have this information in our JAX-B entities but it is not so obvious. To help the odata integration understand the data model, we'll need to mark the links between entities as well. This is probably best explained with some examples.

Lets take a complex example from our system schema: *LimitSetClass*. It looks like this in XSD schema:

```
1   <xs:complexType name="LimitSetClass">
2     <xs:sequence>
3       <xs:element name="id" type="base:idLong" minOccurs="0" maxOccurs="1" />
4       <xs:element name="limitSet" type="LimitSet" minOccurs="1" maxOccurs="1" />
5       <xs:element name="limitClass" type="LimitClass" minOccurs="1" maxOccurs="1"
            />
6       <xs:element name="useCaseId" type="base:idShort" minOccurs="0" maxOccurs="1
            " />
7     </xs:sequence>
8   </xs:complexType>
```

Listing 5.26: XSD Definition of LimitSetClass

For odata, we want to model the fields limitSet and limitClass as links. In this case there is nothing to do, since the odata integration will see that the types of these fields are themselves already entities.

The generated metadata looks like this:

```
1       <EntityType Name="money_contract_v5_0_system_beans_LimitSetClass">
2         <Key>
3           <PropertyRef Name="id"></PropertyRef>
4         </Key>
5         <Property Name="id" Type="Edm.Int64" Nullable="true"></Property>
6         <Property Name="useCaseId" Type="Edm.Int32" Nullable="true"></Property>
7         <NavigationProperty Name="limitSet" Relationship="com.sybase365.mobiliser
              .money_contract_v5_0_system_beans_LimitSetClass_money_contract_v5_0_
              system_beans_LimitSet_money_contract_v5_0_system_beans_LimitSet_money
              _contract_v5_0_system_beans_LimitSetClass" FromRole="money_contract_
              v5_0_system_beans_LimitSetClass_money_contract_v5_0_system_beans_
              LimitSet" ToRole="money_contract_v5_0_system_beans_LimitSet_money_
              contract_v5_0_system_beans_LimitSetClass"></NavigationProperty>
8         <NavigationProperty Name="limitClass" Relationship="com.sybase365.
              mobiliser.money_contract_v5_0_system_beans_LimitSetClass_money_
              contract_v5_0_system_beans_LimitClass_money_contract_v5_0_system_
              beans_LimitClass_money_contract_v5_0_system_beans_LimitSetClass"
              FromRole="money_contract_v5_0_system_beans_LimitSetClass_money_
              contract_v5_0_system_beans_LimitClass" ToRole="money_contract_v5_0_
```

```
            system_beans_LimitClass_money_contract_v5_0_system_beans_
            LimitSetClass"></NavigationProperty>
9       </EntityType>
```

Listing 5.27: LimitSetClass OData Metadata

The create request for a limit set class would be something like this (in json):

```
1 {
2    "useCaseId": 160,
3    "limitSet": {"__metadata": {"uri": "${Project#mobiliser.url}/money_contract_v5
        _0_system_beans_LimitSet(103L)"}},
4    "limitClass": {"__metadata": {"uri": "${Project#mobiliser.url}/money_contract_
        v5_0_system_beans_LimitClass(105L)"}}
5 }
```

Listing 5.28: LimitSetClass Create Request JSON Format

The other case is when we in our XSDs only accept the id of the referenced entity. Here another example from the system contract:

```
1   <xs:complexType name="OrgUnit">
2     <xs:sequence>
3       <xs:element name="id" type="base:strSmall" minOccurs="1" maxOccurs="1" />
4       <xs:element name="name" type="base:strMedium" minOccurs="1" maxOccurs="1"
            />
5       <xs:element name="currency" type="base:Currency" minOccurs="1" maxOccurs="1
            " />
6       <xs:element name="language" type="base:Language" minOccurs="1" maxOccurs="1
            " />
7       <xs:element name="feeSetId" type="base:idLong" minOccurs="0" maxOccurs="1"
            />
8       <xs:element name="legalAddressId" type="base:idLong" minOccurs="0"
            maxOccurs="1" />
9       <xs:element name="limitSetId" type="base:idLong" minOccurs="0" maxOccurs="1
            " />
10      <xs:element name="timeZone" type="base:strMedium" minOccurs="0" maxOccurs="
            1" />
11      <xs:element name="autoCancelAfterMinutes" type="base:idShort" minOccurs="0"
             maxOccurs="1" />
12      <xs:element name="vatPrc" type="base:dAmount" minOccurs="0" maxOccurs="1"
            />
13      <xs:element name="country" type="base:Country" minOccurs="0" maxOccurs="1"
            />
14    </xs:sequence>
15  </xs:complexType>
```

Listing 5.29: XSD Defintion of OrgUnit

The feeSetId here is actually an entity reference. We need to tell the odata integration this. This is also quite simple and only requires an additional annotation: *@EntityReference(entityType = FeeSet.class)*

This will cause the following odata metadata to be generated:

```
1       <EntityType Name="money_contract_v5_0_system_beans_OrgUnit">
2         <Key>
3           <PropertyRef Name="id"></PropertyRef>
4         </Key>
5         <Property Name="autoCancelAfterMinutes" Type="Edm.Int32" Nullable="true"
              ></Property>
6         <Property Name="country" Type="Edm.String" Nullable="true"></Property>
7         <Property Name="currency" Type="Edm.String" Nullable="false"></Property>
8         <Property Name="id" Type="Edm.String" Nullable="false"></Property>
9         <Property Name="language" Type="Edm.String" Nullable="false"></Property>
```

```
10          <Property Name="name" Type="Edm.String" Nullable="false"></Property>
11          <Property Name="timeZone" Type="Edm.String" Nullable="true"></Property>
12          <Property Name="vatPrc" Type="Edm.Decimal" Nullable="true"></Property>
13          <NavigationProperty Name="feeSetId" Relationship="com.sybase365.mobiliser
                .money_contract_v5_0_system_beans_OrgUnit_money_contract_v5_0_system_
                beans_FeeSet_money_contract_v5_0_system_beans_FeeSet_money_contract_
                v5_0_system_beans_OrgUnit" FromRole="money_contract_v5_0_system_beans
                _OrgUnit_money_contract_v5_0_system_beans_FeeSet" ToRole="money_
                contract_v5_0_system_beans_FeeSet_money_contract_v5_0_system_beans_
                OrgUnit"></NavigationProperty>
14          <NavigationProperty Name="limitSetId" Relationship="com.sybase365.
                mobiliser.money_contract_v5_0_system_beans_OrgUnit_money_contract_v5_
                0_system_beans_LimitSet_money_contract_v5_0_system_beans_LimitSet_
                money_contract_v5_0_system_beans_OrgUnit" FromRole="money_contract_v5
                _0_system_beans_OrgUnit_money_contract_v5_0_system_beans_LimitSet"
                ToRole="money_contract_v5_0_system_beans_LimitSet_money_contract_v5_0
                _system_beans_OrgUnit"></NavigationProperty>
15          <NavigationProperty Name="legalAddressId" Relationship="com.sybase365.
                mobiliser.money_contract_v5_0_system_beans_OrgUnit_money_contract_v5_
                0_customer_beans_Address_money_contract_v5_0_customer_beans_Address_
                money_contract_v5_0_system_beans_OrgUnit" FromRole="money_contract_v5
                _0_system_beans_OrgUnit_money_contract_v5_0_customer_beans_Address"
                ToRole="money_contract_v5_0_customer_beans_Address_money_contract_v5_
                0_system_beans_OrgUnit"></NavigationProperty>
16      </EntityType>
```

Listing 5.30: OrgUnit OData Metadata

Unfortunately we have some circular references of entities which are not in the same project, so these must be left as raw ids. Take for example the PaymentInstrument in the wallet contract. It references a customer id but we can't import the customer contract because it itself references the wallet contract. We'll just have to live with these.

5.4.8 General

So much for how to make our services work automatically. A good intro to OData can be found at http://www.odata.org/documentation.

OData, as the name implies, is very data centric. Basically your entities are sorted into sets and you will use standard REST HTTP verbs to manipulate and query the data with some extra additions from odata. OData can return the payloads in atom+xml or json format.

With its current implementation, we will only support PUT POST GET DELETE MERGE operations with a subset of the query operations (and only those already implemented by a standard mobiliser service). This requires no extra coding, only configuration of the annotations at build time.

OData does not use XML schema for defining its data contract but rather additional information included in the service metadata. Requests passing through Mobiliser's OData integration will still be verified against the standard XSD schemas, so the checks performed on incoming data are the same.

An OData service should be completely self explanatory. You should be able to get all known entity sets by doing a GET on the service root and you can get the service metadata descriptor by doing a GET on the service root/$metadata. The data contracts will be explained there.

```
1 curl -s -X GET http://localhost:8080/mobiliser/odata/odata.svc/ | tidy -q -xml -i
      -w 1000
2 curl -s -X GET http://localhost:8080/mobiliser/odata/odata.svc/\$metadata | tidy
     -q -xml -i -w 1000
```

Listing 5.31: cUrl Commands for Metadata

Since many parameters are passed as URI parameters, they will also be url-encoded. This makes the URLs

pretty much unreadable unless you type them in somewhere in plain text and then URL encode them. If you are using SOAP-UI, it will do it for you which makes things a bit easier.

5.5 Endpoint

An endpoint defines the services provided through the Mobiliser gateway, by way of the context configuration previously described. The Mobiliser gateway maps incoming SOAP requests by root element of the payload to the method having exactly this request element as a single parameter; so, it is critical to use each request element only once per context. Restful calls are mapped based on method name, so when registering multiple endpoints for a context, method names are not permitted to overlap. The registration with the Mobiliser gateway happens through the OSGi registry by publish the endpoint as part of an IEndpointInformation object with contains some additional information on XML marshalling and request splitting.

The Mobiliser provides aspects around service implementation to facilitate

- transaction handling,
- user authorisation,
- request tracing and repeating,
- auditing, and
- exception handling and mapping.
- session validation

The service implementation class itself is not polluted with these infrastructure tasks, but contains pure code to do the conversion between JPA and XML beans, and business logic invocations. In most cases, service endpoint implementation classes have dependencies on Business logic classes, the security context (to access privileges of the user executing the service, and the dao factory (for very basic DAO operations - more complex tasks should be outsourced into BL classes); these dependencies are injected from the OSGi service registry.

The service endpoint also creates the transaction context for the business logic.

5.5.1 Configuration

The configuration of service endpoints will import a few standard aspects from the Mobiliser framework to enable these cross-cutting concerns, inject BL, converters, DAO factory, security interceptor, and publish the endpoint into the service registry so that the Mobiliser gateway can make the service available.

First of all, the endpoint implementation itself gets initialised:

```
1  <osgi:reference id="daoFactory"
2    interface="com.sybase365.mobiliser.core.framework.data.dao.factory.api.
         DaoFactory" />
3  <osgi:reference id="customLogic"
4    interface="com.sybase365.mobiliser.money.businesslogic.custom.ICustomLogic" /
         >
5  <osgi:reference id="customConverter"
6    interface="com.sybase365.mobiliser.money.converter.custom.ICustomConverter" /
         >
7  <osgi:reference id="callerUtils"
8    interface="com.sybase365.mobiliser.security.ICallerUtils" />
9   <bean id="customImplDelegate"
10   class="com.sybase365.mobiliser.money.services.custom.CustomEndpoint">
11   <property name="callerUtils" ref="callerUtils" />
12   <property name="daoFactory" ref="daoFactory" />
13   <property name="customLogic" ref="customLogic" />
```

```
14      <property name="customConverter" ref="customConverter" />
15    </bean>
```

Listing 5.32: “OSGI Configuration” OSGI Context Configuration

This is itself has nothing to do with the gateway other than accepting the types we have defined as part of our contract.

In a next step, the aspects provided by the Mobiliser framework are configured around this bean. This is simplified with using a custom “mobiliser-service” Spring tag.

```
1   <mobconfig:mobiliser-service id="customImplComplete"
2     endpoint-ref="customImplDelegate" security-advisor="securityAdvisor"
3     txn-advice="txnAdvice"
4     endpoint-information-objects="custom1Endpoint,custom2Endpoint"
5     proxy-interfaces="com.sybase365.mobiliser.money.services.custom.
          ICustom1Endpoint, com.sybase365.mobiliser.money.services.custom.
          ICustom2Endpoint" />
```

Listing 5.33: “OSGI Configuration” OSGI Context Configuration

Now we wrap each of the prepared mobiliser-service beans into an implementation of IEndpointInformation to be ready for publishing it into the OSGi service registry. Other than the endpoint implementation, we must provide the list of XSD files defining the XML elements and types, which are used by this service methods, a pattern filtering the requests that are handled by this endpoint, and the list of packages holding the JAX-B beans.

By default, the customImplComplete will be exported to the OSGi registry for use by other code running in the container. The service is proxied to add a validator which checks internal calls against the XSD schema. The endpoint-information-objects is used when creating this proxy since the information for creating the schema is configured there.

In case the request elements (in the same namespace) are split across multiple XSD files, you must define multiple EndpointInformation objects and the regular expression pattern must do a unique filtering on request elements so that the gateway component can map incoming requests using the correct unmarshaller. If this is not accurate, it may happen that the gateway passes a request into the wrong endpoint, and the unmarshaller will then not be able to do its job since it is missing the appropriate XSD files.

This is also true if you have multiple XSDs with requests in different namespaces but all implemented as part of a single target object.

```
1   <bean id="custom1Endpoint"
2     class="com.sybase365.mobiliser.ws.api.impl.EndpointInformationImpl">
3     <property name="endpoint" ref="customImplComplete"/>
4     <property name="schemaPaths">
5       <list>
6         <value>/com/sybase365/mobiliser/framework/contract/xsd/base-5-0.xsd</
              value>
7         <value>/com/sybase365/mobiliser/money/contract/custom/xsd/beans-5-0.xsd</
              value>
8         <value>/com/sybase365/mobiliser/money/contract/custom/xsd/custom1-
              requests-5-0.xsd</value>
9       </list>
10    </property>
11    <property name="allowedMessageElements">
12      <list>
13        <value>^.*Custom1.*$</value>
14      </list>
15    </property>
16    <property name="contextPaths">
17      <list>
18        <value>com.sybase365.mobiliser.framework.contract.v5_0.base</value>
19        <value>com.sybase365.mobiliser.money.contract.v5_0.custom</value>
20        <value>com.sybase365.mobiliser.money.contract.v5_0.custom.beans</value>
```

```
21        </list>
22      </property>
23    </bean>
24
25    <bean id="custom2Endpoint"
26      class="com.sybase365.mobiliser.ws.api.impl.EndpointInformationImpl">
27      <property name="endpoint" ref="customImplComplete"/>
28      <property name="schemaPaths">
29        <list>
30          <value>/com/sybase365/mobiliser/framework/contract/xsd/base-5-0.xsd</
                value>
31          <value>/com/sybase365/mobiliser/money/contract/custom/xsd/beans-5-0.xsd</
                value>
32          <value>/com/sybase365/mobiliser/money/contract/custom/xsd/custom2-
                requests-5-0.xsd</value>
33        </list>
34      </property>
35      <property name="allowedMessageElements">
36        <list>
37          <value>^.*Custom2.*$</value>
38        </list>
39      </property>
40      <property name="contextPaths">
41        <list>
42          <value>com.sybase365.mobiliser.framework.contract.v5_0.base</value>
43          <value>com.sybase365.mobiliser.money.contract.v5_0.custom</value>
44          <value>com.sybase365.mobiliser.money.contract.v5_0.custom.beans</value>
45        </list>
46      </property>
47    </bean>
```

Listing 5.34: “OSGI Configuration” OSGI Context Configuration

These endpoint information objects are then published into the OSGi service registry, which will automatically make them available through the Mobiliser gateway.

```
1   <osgi:service ref="custom1Endpoint"
2     interface="com.sybase365.mobiliser.ws.api.IEndpointInformation">
3     <osgi:service-properties>
4       <beans:entry key="path" value="/custom" />
5     </osgi:service-properties>
6   </osgi:service>
7
8
9   <osgi:service ref="custom2Endpoint"
10    interface="com.sybase365.mobiliser.ws.api.IEndpointInformation">
11    <osgi:service-properties>
12      <beans:entry key="path" value="/custom" />
13    </osgi:service-properties>
14  </osgi:service>
```

Listing 5.35: “OSGI Configuration” OSGI Context Configuration

Then, there is some boilerplate configuration to make sure that all the different aspects provided by the framework are available for injection.

```
1   <osgi:reference id="transactionManager"
2     interface="org.springframework.transaction.PlatformTransactionManager"
3     context-class-loader="service-provider" />
4
5   <osgi:reference id="securityAdvisor" filter="(path=/custom)">
6     <osgi:interfaces>
7       <value>org.springframework.aop.Advisor</value
8     </osgi:interfaces>
```

```
  </osgi:reference>


  <tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
      <tx:method name="*" />
    </tx:attributes>
  </tx:advice>
</beans>
```

Listing 5.36: “OSGI Configuration” OSGI Context Configuration

As alternative to this setup, we could have had two endpoint implementations, each only processing the requests on each request xsd. Then the regex could simply be ^.*$ since the mappings would be implicit based on the endpoint implementation itself. (The gateway will not try to map something for which there is no method) This would mean we have an additional <mobconfig:mobiliser-service> element, so one for each implementation.

Looking at endpoints in the other direction, we could imagine we have a single requests XSD but have many many requests which we split into multiple interfaces. We could then have a single implementation for each interface (so X implementations) and would then have X EndpointInformation objects. In this case we also do not need to filter the message elements since we have no overlapping methods. Also in this case we have additional <mobconfig:mobiliser-service> elements, so one for each implementation.

5.6 Service Aspects

As mentioned previously, the endpoints are proxied and a list of advice objects ist added to the advisor chain to address cross-cutting concernts.

5.6.1 Security Advisor

This is the standard security advisor from Spring Security. In most standard contexts we use the following configuration:

```
  <security:global-method-security
    secured-annotations="disabled" jsr250-annotations="enabled"
    pre-post-annotations="enabled" />
```

Listing 5.37: Method Security Interceptor Configuration

This will create an Advisor named org.springframework.security.methodSecurityMetadataSourceAdvisor. Two MetadataSources are created - one fed from the JSR-250 annotations on our endpoint and one from the special Spring Security @PreAuthorize @PostAuthorize annotations. This will enforce our role configuration based on the roles of the caller of the incoming message.

5.6.2 ResponseCodeAspect

This method interceptor ensures that the Status field of outgoing responses is always filled and copies the conversation id from the request into the response. It also catches any exception thrown and creates a response object and fills it with an error code. It appears twice in the advisor list, once to catch exceptions coming from the endpoint itself, and once to catch them coming form the traceable request interceptor.

5.6.3 TraceableRequestAspect

This method interceptor handles TraceableRequestType checking; if the exact same call has already been processed, it returns the old response found in the database. Otherwise the reponse is stored to the database for future processing. Usually, any request generating or changinge data should be of that type. Through this mechanism we can make sure that these requests are not processed twice, accidentally. Double-sending a request could happen because of issues on the network layer, but also due to misbehaving clients. Traceable requests must always provide a trace number (we recommend to use a UUID for that). This trace number must be unique for the last X hours (which is configurable, default 72h). If the trace number is not unique, an error is returned. Additionally, this mechanism can also be used to query for a request result a second time without actually executing the request. To achieve that, the caller must send the same request with the exact same trace number, but explicitly mark the request as a repeat request. In this case, if the caller is identical, Mobiliser will return the stored response from the previous call.

The following error codes may be returned

- 51, trace number is not unique, but repeat flag missing
- 52, cannot return cached message, because original request is still in progress
- 53, wrong user for repeat, the original request came from another client
- 54, repeat flag set, but no message found for given trace number
- 55, no response cached for trace number (this should only happen through severe system crashes)

5.6.4 AuditAspect

This method interceptor stores audit information about incoming web service calls by sending relevant information to a configured instance of IAuditManager. Auditing is discussed in detail in another chapter.

5.6.5 SessionInterceptor

All MobiliserRequestTypes have a field sessionId which clients can use to mark the request as belonging to a previously initiated session. Endpoints must check the validity of this session value and the SessionInterceptor does just that. It also updates the last used date of the session.

5.6.6 TransactionInterceptor

In most cases, it is simplest to define the transaction demarcation for endpoint methods at the method level. You can do that by configuring the standard standard Spring transation interceptor. If you don't need any special transactional handling, you can configure your transaction attributes using Spring's XML support like this:

```
1  <tx:advice id="txAdvice" transaction-manager="transactionManager">
2    <tx:attributes>
3      <tx:method name="*" />
4    </tx:attributes>
5  </tx:advice>
```

Listing 5.38: Endpoint Transaction Interceptor

This simply configures the aspect to apply the default transaction semantics of the underlying transaction manager to any methods intercepted by the pointcut defined by the framework.
Note: This is $\mathrm{args(com.sybase365.mobiliser.framework.contract.v5\_0.base.MobiliserRequestType)}$ in the default configuration

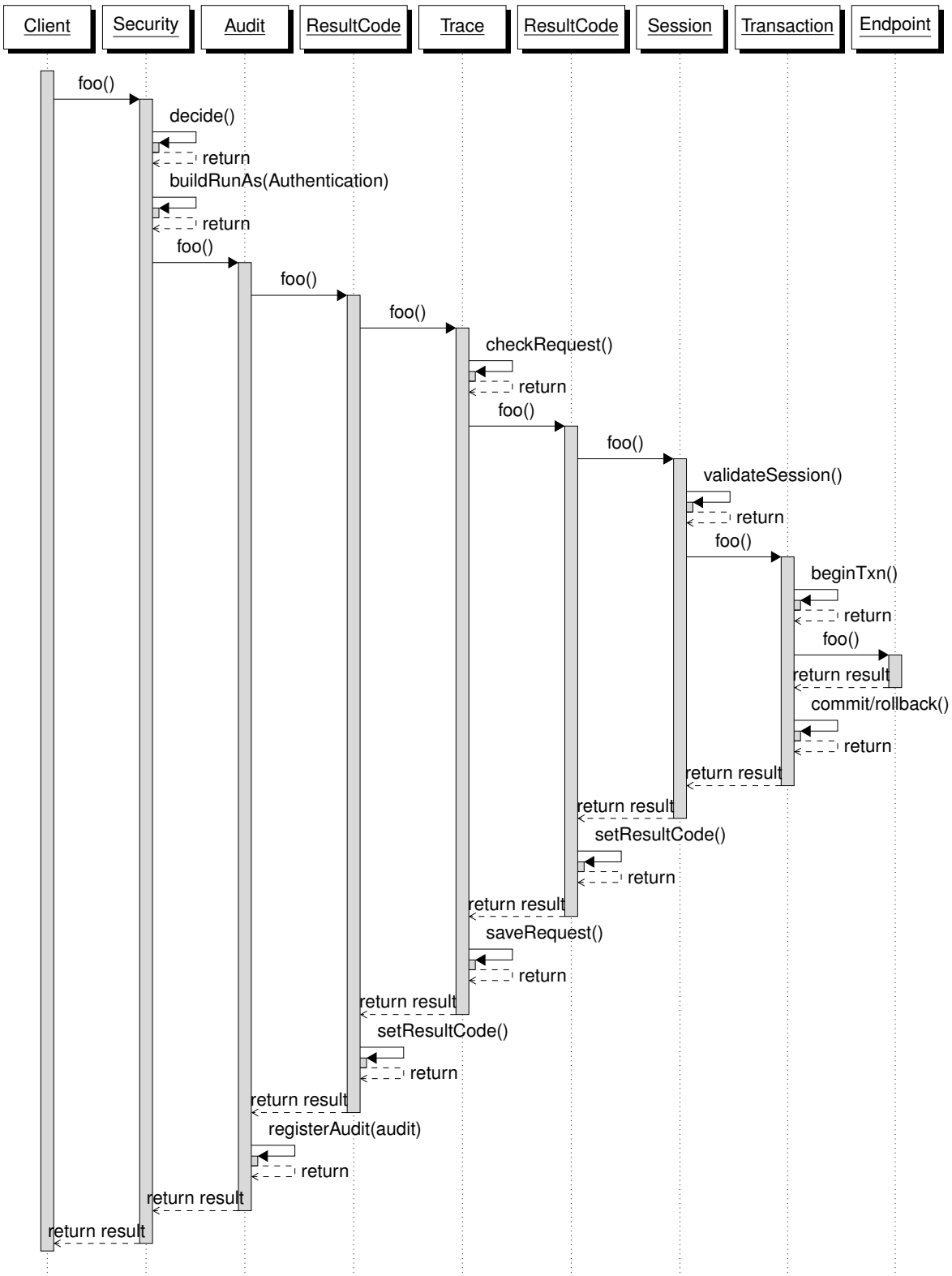

Figure 5.4: Endpoint Advisor Chain

5.6.7 Service Filters

In addition to the pre-configured service aspects, you can also easily register own service filters by implementing the IMobiliserServiceFilter interface and registering it into the OSGi service registry. This provides an easy way of injecting custom behavior into standard Mobiliser service calls. Implementations have full access to to

the current request and response objects and can modify these if required. These do not necessarily need to be the original request and response objects. They could have been modified by previously invoked service filters.

```
1  public interface IMobiliserServiceFilter {
2
3      /**
4       * This method is invoked before the Mobiliser endpoint is executed. It is
5       * not executed inside an existing database transaction. If this method
6       * terminates by throwing an exception, the remainder of the service filter
7       * chain as well as the method invocation itself is skipped. Instead, all
8       * previously called filters will have
9       * {@link IMobiliserServiceFilter#afterMobiliserServiceCall(String, String,
            MobiliserRequestType, Throwable)}
10      * invoked and the exception will be re-thrown.
11      *
12      * @param endpoint
13      *             The fully qualified class name of the interface that declared
14      *             the endpoint method.
15      * @param method
16      *             The name of the endpoint method that will be invoked.
17      * @param request
18      *             The current request object. This object can be modified.
19      */
20     void beforeMobiliserServiceCall(String endpoint, String method,
21         MobiliserRequestType request);
22
23     /**
24      * This method is invoked after the Mobiliser endpoint has executed without
25      * an exception. This does not necessarily mean that the endpoint has
26      * executed without an error, the status element of the response can be
27      * different from zero. This method is not executed inside an existing
28      * database transaction. Any exception that is thrown by this method is
29      * logged only. The processing will continue with all other service filters
30      * and the response object will be returned to the caller.
31      *
32      * @param endpoint
33      *             The fully qualified class name of the interface that declared
34      *             the endpoint method.
35      * @param method
36      *             The name of the endpoint method that has been invoked.
37      * @param request
38      *             The current request object. This object can be modified.
39      * @param response
40      *             The current response object. This object can be modified.
41      */
42     void afterMobiliserServiceCall(String endpoint, String method,
43         MobiliserRequestType request, MobiliserResponseType response);
44
45     /**
46      * This method is invoked after the Mobiliser endpoint has thrown a
47      * throwable. This method is not executed inside an existing database
48      * transaction. Any exception that is thrown by this method is logged only.
49      * The processing will continue with all other service filters and the
50      * original throwable is re-thrown then.
51      *
52      * @param endpoint
53      *             The fully qualified class name of the interface that declared
54      *             the endpoint method.
55      * @param method
56      *             The name of the endpoint method that has been invoked.
57      * @param request
58      *             The current request object. This object can be modified.
59      * @param t
```

```
60       *            The occurred throwable.
61       */
62      void afterMobiliserServiceCall(String endpoint, String method,
63          MobiliserRequestType request, Throwable t);
64
65      /**
66       * Defines the priority of this service filters. Lower numbers will be
67       * invoked earlier on the <i>before</i> method and later on the <i>after</i>
68       * method than service filters with higher numbers. The order on service
69       * filters with equal priority is random (but it is ensured that the order
70       * of corresponding <i>before</i> and <i>after</i> methods is consistent)
71       *
72       * @return the filters priority.
73       */
74      int getPriority();
75
76      /**
77       * This flag indicates if this filter must be applied to external service
78       * calls, i.e. calls from outside of the container.
79       *
80       * @return <code>true</code> if applicable to external service calls.
81       */
82      boolean applyToExternalServiceCalls();
83
84      /**
85       * This flag indicates if this filter must be applied to internal service
86       * calls, i.e. calls within the container.
87       *
88       * @return <code>true</code> if applicable to internal service calls.
89       */
90      boolean applyToInternalServiceCalls();
```

Listing 5.39: IMobiliserServiceFilter interface

This XML snippet shows the registration of a custom myServiceFilter bean so that it is picked up and applied with each service invocation.

```
1   <osgi:service ref="myServiceFilter"
2     interface="com.sybase365.mobiliser.framework.service.filter.api.
          IMobiliserServiceFilter"
3     ranking="10">
4   </osgi:service>
```

Listing 5.40: IMobiliserServiceFilter OSGi registration

5.6.8 Custom Service Aspects

In addition to implementing the IMobiliserServiceFilter interface and get custom actions included conveniently, you can also implement plain method interceptors and have them applied with service invocations as well. Implement the org.aopalliance.intercept.MethodInterceptor interface with any logic you want to apply to Mobiliser service invocations and register it with the OSGi service registry.

```
1   <osgi:service ref="myCustomServiceAspect"
2     interface="org.aopalliance.intercept.MethodInterceptor" ranking="10"
3     context-class-loader="service-provider">
4     <osgi:service-properties>
5       <beans:entry key="type" value="myAspect" />
6       <beans:entry key="origin" value="mobiliser" />
7     </osgi:service-properties>
8   </osgi:service>
```

Listing 5.41: Custom Service Aspects - OSGi registration

To actually get the aspect part of the filter chain for Mobiliser service invocations, you have to list it by the name you used in the type service attribute in the container's ./conf/system.properties file. Mobiliser maintains two different lists for external and internal service calls. The aspects are invoked in the order in which they are listed.

```
1 # these two configurations define the default adviser chain for all mobiliser
      services and internal service calls
2 # the values defined here must be defined as interceptors in the OSGi service
      registry
3 com.sybase365.mobiliser.framework.service.config.
      MobiliserServiceBeanDefinitionParser.serviceAdviceConfig=audit,security,...,
      myAspect
4 com.sybase365.mobiliser.framework.service.config.
      MobiliserServiceBeanDefinitionParser.internalServiceAdviceConfig=security,
      responseCode,...,myAspect
```

Listing 5.42: Custom Service Aspects - System Properties

5.7 Service Business Logic

The service business logic is the logical service implementation. It provides interfaces with simple Java types or DAO beans. You can also use beans to simplify the method request and response parameters. It also provides the converter that converts contract and DAO bean types between each other. The standard layout of the project is: Please note that the converter and bean packages are optional.

```
/
├── pom.xml
├── src
│   └── main
│       ├── java
│       │   └── com
│       │       └── sybase365
│       │           └── mobiliser
│       │               └── custom
│       │                   └── sample
│       │                       ├── businesslogic
│       │                       │   └── aservice
│       │                       │       ├── IAServiceInterface.java
│       │                       │       ├── bean
│       │                       │       │   ├── IAnyRequestBean.java
│       │                       │       │   ├── IAnyResponseBean.java
│       │                       │       │   └── impl
│       │                       │       │       ├── AnyRequestBeanImpl.java
│       │                       │       │       └── AnyResponseBeanImpl.java
│       │                       │       └── impl
│       │                       │           └── AServiceInterfaceImpl.java
│       │                       └── converter
│       │                           └── aservice
│       │                               ├── IAServiceConverter.java
│       │                               └── impl
│       │                                   └── AServiceConverterImpl.java
│       └── resources
│           └── META-INF
│               └── spring
│                   ├── bundle-context-osgi.xml
│                   └── bundle-context.xml
```

Figure 5.5: Example directory tree for service business logic bundle sources

This bundle must export the OSGI services for the converter, beans and business logic interface.

The business logic may also be responsible for defining the transaction handling (see 5.1.1) when an JPA bean needs to be created or updated even if an exception/failure occured.

All interface packages are part of the OSGi bundle export. All impl sub packages are OSGi private bundles.

5.7.1 Converter

The converter interface defines fromContract() and toContract() methods to convert between JPA and XML beans. Usually there is a very strong correlation between these to beans and the conversion is straight forward.

When converting from contract to JPA beans, the converter either creates a new JPA instance if the PK of the JPA bean is not defined in the XML bean, or it loads the JPA through the DAO API and updates all fields of the loaded JPA bean with values coming from the XML bean. Hence, the converter implementation will require the DAO factory injected in all cases. The converter must be prepared for the situation that the DAO will not return an object for the given PK from the contract (since the entry does not exist in the database). This is not an error situation for the converter, the converter should rather return the null object, if the bean does not exist in the database, or set properties to null in case nested beans cannot be retrieved from the database. It is in the responsibility of business logic to fail gracefully if getting null values is not expected.

Like mentioned before, all OSGi service are published by interfaces and implemented in the impl sub package. This is an example OSGi service interface for the converter service:

```
1  package com.sybase365.mobiliser.money.converter.audit;
2
3  import com.sybase365.mobiliser.money.contract.v5_0.audit.beans.AuditLog;
4
5  public interface IAuditConverter {
6
7      com.sybase365.mobiliser.money.persistence.model.customer.AuditLog
           fromContract(
8          AuditLog contract);
9
10     AuditLog toContract(
11         com.sybase365.mobiliser.money.persistence.model.customer.AuditLog model);
12
13 }
```

Listing 5.43: “Audit” Converter Interface Example

The implementation is done in the impl sub package. This is an example implementation for the interface above:

```
1  package com.sybase365.mobiliser.money.converter.audit.impl;
2
3  import ...;
4
5  public class AuditConverter implements IAuditConverter {
6
7      private DaoFactory daoFactory;
8
9      public final void setDaoFactory(DaoFactory daoFactory) {
10     this.daoFactory = daoFactory;
11     }
12
13     @Override
14     public final com.sybase365.mobiliser.money.persistence.model.customer.
           AuditLog fromContract(
15         AuditLog contract) {
16     com.sybase365.mobiliser.money.persistence.model.customer.AuditLog auditLog;
17
```

```
18      if (contract.getId() != null) {
19          auditLog = daoFactory.getAuditLogDao().getById(contract.getId());
20      } else {
21          auditLog = daoFactory.getAuditLogDao().newInstance();
22      }
23
24      if (auditLog == null) {
25          return null;
26      }
27
28      auditLog.setAction(contract.getAction());
29      auditLog.setActionResultCode(contract.getActionResultCode());
30      auditLog.setApplication(contract.getApplication());
31      auditLog.setCaller(daoFactory.getCustomerDao().getById(
32          Long.valueOf(contract.getCaller())));
33      if (contract.getCustomerId() != null) {
34          auditLog.setCustomer(daoFactory.getCustomerDao().getById(
35              contract.getCustomerId()));
36      }
37      auditLog.setDetails(contract.getDetails());
38      auditLog.setDevice(contract.getDevice());
39      auditLog.setDeviceId(contract.getDeviceId());
40      auditLog.setDurationMilliseconds(contract.getDuration());
41      auditLog.setOrigin(contract.getOrigin());
42      auditLog.setOtherDeviceId(contract.getOtherDeviceId());
43      auditLog.setParameter1(contract.getParam1());
44      auditLog.setParameter2(contract.getParam2());
45      auditLog.setTraceNo(contract.getTraceNumber());
46      if (contract.getTxnId() != null) {
47          auditLog.setTransaction(daoFactory.getTransactionDao().getById(
48              contract.getTxnId()));
49      }
50
51      return auditLog;
52      }
53
54      @Override
55      public final AuditLog toContract(
56          com.sybase365.mobiliser.money.persistence.model.customer.AuditLog model)
                {
57      AuditLog auditLog = new AuditLog();
58      auditLog.setAction(model.getAction());
59      auditLog.setActionResultCode(model.getActionResultCode());
60      auditLog.setApplication(model.getApplication());
61      auditLog.setCaller(model.getCaller() == null ? 0 : model.getCaller()
62          .getId().longValue());
63      auditLog.setCreated(FormatUtils.getSaveXMLGregorianCalendar(model
64          .getCreationDate()));
65      auditLog.setCreatedBy(model.getCreator());
66      auditLog.setCustomerId(model.getCustomer() == null ? null : model
67          .getCustomer().getId());
68      auditLog.setDetails(model.getDetails());
69      auditLog.setDevice(model.getDevice());
70      auditLog.setDeviceId(model.getDeviceId());
71      auditLog.setDuration(model.getDurationMilliseconds());
72      auditLog.setId(model.getId());
73      auditLog.setOrigin(model.getOrigin());
74      auditLog.setOtherDeviceId(model.getOtherDeviceId());
75      auditLog.setParam1(model.getParameter1());
76      auditLog.setParam2(model.getParameter2());
77      auditLog.setTraceNumber(model.getTraceNo());
78      auditLog.setTxnId(model.getTransaction() == null ? null : model
79          .getTransaction().getId());
80
81      return auditLog;
```

```
82      }
83  }
```

Listing 5.44: “Audit” Converter Implementation Example

5.7.2 Service Business Logic

The business logic interfaces define the particular business logic API for the bundle. To keep things clearly arranged, business logic may be split up into various interfaces. All business logic APIs are defined on JPA beans for both, input and return parameters. Any class invoking business logic must use the provided converters in case the input is only available as XML beans to map XML into JPA beans.

Business logic implementation classes will usually have the DAO factory injected so that they can access, update, create data through this API. In case the execution of business logic fails (due to controlled reasons), the implementation must throw a subclass of MobiliserServiceException. The Mobiliser response will be filled with the error code linked to this exception by a framework aspect that is wrapped around the service endpoint implementation.

This is an example business logic interface, exported as OSGi service:

```
1  package com.sybase365.mobiliser.money.businesslogic.audit;
2
3  import java.util.List; ...
4
5  public interface IAuditLogLogic {
6
7      long createAuditLog(final AuditLog auditLog, final long callerId)
8          throws EntityMandatoryException, IllegalDataException;
9
10     void updateAuditLog(final AuditLog auditLog, final long callerId)
11         throws EntityNotFoundException, EntityMandatoryException;
12
13     void deleteAuditLog(final long auditLogId, final long callerId)
14         throws EntityNotFoundException;
15
16     AuditLog getAuditLog(final long auditLogId, final long callerId)
17         throws EntityNotFoundException;
18
19     List<AuditLog> getAuditLogsByCustomer(final long customerId,
20         final long callerId) throws EntityNotFoundException;
21
22     List<AuditLog> getAuditLogsByTransaction(final long txnId,
23         final long callerId) throws EntityNotFoundException;
24 }
```

Listing 5.45: “Audit” Service Interface Example

Again, the actual implementation is located in the impl sub package:

```
1  package com.sybase365.mobiliser.money.businesslogic.audit.impl;
2
3  import java.util.List;...
4
5  public final class AuditLogLogic implements IAuditLogLogic, InitializingBean {
6      /** The <code>Log</code> instance to use. */
7      private static final org.slf4j.Logger LOG = org.slf4j.LoggerFactory
8          .getLogger(AuditLogLogic.class);
9
10     private DaoFactory daoFactory;
11
12     private DaoFactory getDaoFactory() {
13     return daoFactory;
```

```
14      }
15
16      public void setDaoFactory(DaoFactory daoFactory) {
17      this.daoFactory = daoFactory;
18      }
19
20      @Override
21      public void afterPropertiesSet() throws Exception {
22      if (daoFactory == null)
23          throw new IllegalStateException("daoFactory is required");
24      }
25
26      @Override
27      public long createAuditLog(AuditLog auditLog, long callerId)
28          throws EntityMandatoryException, IllegalDataException {
29      if (LOG.isTraceEnabled()) {
30          LOG.trace("# createAuditLog({},{})",
31              new Object[] { auditLog, Long.valueOf(callerId) });
32      }
33
34      // check if an auditLog object has been given
35      if (auditLog == null) {
36          throw new EntityMandatoryException("No auditLog information set");
37      }
38
39      // id must not be set on create operation
40      if (auditLog.getId() != null) {
41          throw new IllegalDataException(
42              "Id must not be set on create operation", auditLog.getId()
43                  .toString());
44      }
45
46      // all we have to do now is persist the already converted AuditLog
47      // bean
48      if (LOG.isDebugEnabled()) {
49          LOG.debug(
50              "# Creating auditLog of type [{}] for Customer = {}.",
51              new Object[] {
52                  auditLog.getAction(),
53                  (auditLog.getCustomer() == null ? "null" : auditLog
54                      .getCustomer().getId()) });
55      }
56
57      getDaoFactory().getAuditLogDao().save(auditLog, Long.valueOf(callerId));
58
59      return auditLog.getId().longValue();
60      }
61
62      @Override
63      public void updateAuditLog(AuditLog auditLog, long callerId)
64          throws EntityNotFoundException, EntityMandatoryException {
65      if (LOG.isTraceEnabled()) {
66          LOG.trace("# updateAuditLog({},{})",
67              new Object[] { auditLog, Long.valueOf(callerId) });
68      }
69
70      // check if an identity object has been given
71      if (auditLog == null) {
72          throw new EntityMandatoryException("No auditLog information set");
73      }
74
75      // check if the auditLog's Id is not null
76      if (auditLog.getId() == null) {
77          throw new EntityMandatoryException(
78              "In order to update an auditLog, an auditLog ID has to be set");
```

```
79      }
80
81      // otherwise the auditLog has a valid ID and customer
82      // information set and thus can be updated
83      getDaoFactory().getAuditLogDao().update(auditLog,
84          Long.valueOf(callerId));
85      }
86
87      @Override
88      public void deleteAuditLog(long auditLogId, long callerId)
89          throws EntityNotFoundException {
90      LOG.trace("# deleteAuditLog({},{})", Long.valueOf(auditLogId),
91          Long.valueOf(callerId));
92
93      AuditLogDAO auditLogDao = getDaoFactory().getAuditLogDao();
94
95      // first of all load the auditLog that should be deleted
96      AuditLog auditLog = auditLogDao.getById(Long.valueOf(auditLogId));
97      if (auditLog == null) {
98          throw new EntityNotFoundException(
99              "No auditLog entry found with ID #" + auditLogId,
100             Long.toString(auditLogId));
101     }
102
103     if (LOG.isDebugEnabled()) {
104         LOG.debug("# Removing auditLog #{} from database",
105             new Object[] { Long.valueOf(auditLogId) });
106     }
107
108     auditLogDao.delete(auditLog);
109     }
110
111     @Override
112     public AuditLog getAuditLog(long auditLogId, long callerId)
113         throws EntityNotFoundException {
114     if (LOG.isTraceEnabled()) {
115         LOG.trace("# getAuditLog({})",
116             new Object[] { Long.valueOf(auditLogId) });
117     }
118
119     // try to load the auditLog for the given Id
120     final AuditLog auditLog = getDaoFactory().getAuditLogDao().getById(
121         Long.valueOf(auditLogId));
122
123     if (auditLog == null) {
124         throw new EntityNotFoundException("AuditLog #" + auditLogId
125             + " not found", Long.toString(auditLogId));
126     }
127
128     return auditLog;
129     }
130
131     @Override
132     public List<AuditLog> getAuditLogsByCustomer(long customerId, long callerId)
133         throws EntityNotFoundException {
134     LOG.trace("# getAuditLogsByCustomer({})", Long.valueOf(customerId));
135
136     CustomerDAO customerDao = getDaoFactory().getCustomerDao();
137
138     // load the customer
139     Customer customer = customerDao.getById(Long.valueOf(customerId));
140     if (customer == null) {
141         throw new EntityNotFoundException("No customer found with ID #"
142             + customerId, Long.toString(customerId));
143     }
```

```
144
145     // try to load the auditLog for the given Id
146     return getDaoFactory().getAuditLogDao().getAuditLogsByCustomer(
147         customerId);
148     }
149
150     @Override
151     public List<AuditLog> getAuditLogsByTransaction(long txnId, long callerId)
152         throws EntityNotFoundException {
153     LOG.trace("# getAuditLogsByTransaction({})", Long.valueOf(txnId));
154
155     TransactionDAO transactionDao = getDaoFactory().getTransactionDao();
156
157     // load the transaction
158     Transaction transaction = transactionDao.getById(Long.valueOf(txnId));
159     if (transaction == null) {
160         throw new EntityNotFoundException("No transaction found with ID #"
161             + txnId, Long.toString(txnId));
162     }
163
164     // try to load the auditLog for the given Id
165     return getDaoFactory().getAuditLogDao()
166         .getAuditLogsByTransaction(txnId);
167     }
168 }
```

Listing 5.46: “Audit” Service Implementation Example

Please take particular note of the structure of the implementation example:

- The JPA/DAO factory service is injected via spring. Therefore the implementation should implement the InitializingBean interface and therefore the afterPropertiesSet() method. This method checks if the daoFactory bean configuration is valid and the service available after bundle start up. The bean setter is public.
- All methods runs in the transaction context created by the service endpoint.
- On errors the business logic throws a sub class of the MobiliserServiceException, like the EntityNotFoundException().

5.7.3 Configuration

First of all the OSGi Service import and export is configured in the `bundle-context-osgi.xml`:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4  xmlns:osgi="http://www.eclipse.org/gemini/blueprint/schema/blueprint"
5  xmlns:beans="http://www.springframework.org/schema/beans"
6  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
       springframework.org/schema/beans/spring-beans-3.0.xsd http://www.eclipse.org/
       gemini/blueprint/schema/blueprint http://www.eclipse.org/gemini/blueprint/
       schema/blueprint/gemini-blueprint-1.0.xsd">
7
8    <osgi:reference id="transactionManager"
9    interface="org.springframework.transaction.PlatformTransactionManager"
10   context-class-loader="service-provider" sticky="false" />
11   <osgi:reference id="daoFactory"
12   interface="com.sybase365.mobiliser.money.persistence.dao.factory.api.DaoFactory
         "
13   sticky="false" />
14   <!-- new services -->
```

```
15   <osgi:service ref="auditLogLogic"
16   interface="com.sybase365.mobiliser.money.businesslogic.audit.IAuditLogLogic"
17   ranking="10">
18     <osgi:service-properties>
19       <beans:entry key="origin" value="mobiliser" />
20     </osgi:service-properties>
21   </osgi:service>
22   <osgi:service ref="auditConverter"
23   interface="com.sybase365.mobiliser.money.converter.audit.IAuditConverter"
24   ranking="10">
25     <osgi:service-properties>
26       <beans:entry key="origin" value="mobiliser" />
27     </osgi:service-properties>
28   </osgi:service>
29 </beans>
```

Listing 5.47: “Audit” OSGI Service Configuration Example

The standard spring configuration in business logic bundles imports the DAO factory from the OSGi service registry and injects it into converters and business logic implementations. It also injects the transaction manager by the `tx:annotation-driven` tag and the @Transactional annotation may be used.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:beans="http://www.
       springframework.org/schema/beans"
4    xmlns:tx="http://www.springframework.org/schema/tx" xmlns:aop="http://www.
       springframework.org/schema/aop"
5    xmlns:mobconfig="http://www.sybase.com/mobiliser/money/service/config"
6    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
       springframework.org/schema/beans/spring-beans-3.0.xsd
7    http://www.springframework.org/schema/aop http://www.springframework.org/schema
       /aop/spring-aop-3.0.xsd
8    http://www.springframework.org/schema/tx http://www.springframework.org/schema/
       tx/spring-tx-3.0.xsd
9    http://www.springframework.org/schema/context http://www.springframework.org/
       schema/context/spring-context-3.0.xsd
10   http://www.sybase.com/mobiliser/money/service/config http://www.sybase.com/
       mobiliser/money/service/config/sybase-service-config-5.0.xsd">
11
12   <tx:annotation-driven />
13
14   <bean id="auditLogLogic"
15     class="com.sybase365.mobiliser.money.businesslogic.audit.impl.AuditLogLogic">
16     <property name="daoFactory" ref="daoFactory" />
17   </bean>
18
19   <bean id="auditConverter"
20     class="com.sybase365.mobiliser.money.converter.audit.impl.AuditConverter">
21     <property name="daoFactory" ref="daoFactory" />
22   </bean>
23
24 </beans>
```

Listing 5.48: “Audit” OSGI Service Configuration Example

5.8 Mobiliser Virus Scan Framework

The Mobiliser Framework includes a virus scan tool too, using the SAP NetWeaver Virus Scan Interface and its plug-ins for a variety of commercial Virus Scanner. This chapter describes the usage of the Mobiliser Virus Scan Framework Bundle in service implementation.

Note: The Mobiliser Virus Scan Framework should be used to parse any binary/file[2] data that is part of the service request.

The configuration of the Mobiliser Virus Scan Framework see the Money Mobiliser Installation Guide and http://help.sap.com/saphelp_nw04/helpdata/EN/ca/7cb340be761b07e10000000a155106/frameset.htm

5.8.1 Usage

First of all you must add the Mobiliser Virus Scan Framework dependency to your project configuration pom.xml

```
<dependency>
  <groupId>com.sybase365.mobiliser.framework</groupId>
  <artifactId>com.sybase365.mobiliser.framework.vscan</artifactId>
  <version>${version.framework}</version>
</dependency>
```

Listing 5.49: "Project Configuration pom.xml"

This will import the OSGi service interface com.sybase365.mobiliser.framework.vscan.IVScan to your project. Now you need to inject this interface to your implementation. In your implementation you can directly call the scanner the following way:

```
private void checkBinaryData(byte[] binaryCode) {
    try {
          if (binaryCode != null) {
          final InputStream inStream = new ByteArrayInputStream(
            binaryCode);
          this.ivScan.scan(inStream);
        }
    } catch (Exception ex) {
        throw new Exception(ex, "Virus Scanner Error!");
    }
  }
```

Listing 5.50: Example usage of the Mobiliser Virus Scan Interface

The scan method throws three different exceptions:

- com.sybase365.mobiliser.framework.vscan.exception.ScanException The scanner can not scan the binary data.
- com.sybase365.mobiliser.framework.vscan.exception.ServiceException An Service exception in the Virus Scan Interface has occurred.
- com.sybase365.mobiliser.framework.vscan.exception.VirusException A Virus was found in the binary data.

Note: The Mobiliser Virus Scan Framework is initialized on bundle start up. If the scanner can not start due to configuration or installation issues, the scan is disabled and no error or warning occurs during scan service call. Please make sure that the Mobiliser Virus Scan Framework is configured and starts by examine the Mobiliser log file.

[2]Binary Data Fields are usually modeled as xs:base64Binary type fields

5.9 Using Services

5.9.1 Within the Container

Mobiliser services are not only published through the Framework Gateway, but also available for internal consumption through the OSGi service registry. This makes it easy for tasks, event handlers, and other components, to call Mobiliser services in-memory without going over HTTP.

Internal Service Publication

In order to make an endpoint available for internal use, the service must be exported not only as an IEndpointInformation to the Framework Gateway, but also using the actual service interface. The service for internal consumption differs slightly from the service that is publicly made available; internal service calls will not be traced and audited. This is done automatically by our custom <mobconfig:mobiliser-service> bean configuration, having only the response code and security aspect configured. If you wish to disable the automatic export for any reason add this to the configuration.

```
disable-automatic-internal-export="true"
```

Internal Service Consumption

To consume an internal service it is not always sufficient to simply import the service from the OSGi service registry and use it. Internal service calls are advised by the security aspect, just as any external service call as well. Hence, the call must be authenticated, i.e., the Spring Security context must hold an authentication object.

In case the internal service call is triggered as part of a currently running external service call invocation, the security context is available and the service may directly be consumed. If the internal service, however, is called by other Mobiliser components like event handlers or tasks, which are not part of an external service call, the authentication information must be provided explicitly.

To support internal service call authentication, we provide an InternalServiceCallAuthenticatorFactory; this factory returns a method interceptor which an be used to create a proxy service which creates an authentication object from a provided username through Spring Security (we do not require a credential here, since internal service calls are issued from within the trusted environment of the container and a credential would not add security in this case). The username is provided through the IInternalServiceCallAuthenticationData interface.

The framework provides an implementation of the factory which can be fetched from the OSGi registry:

```
1  <osgi:reference id="internalAuthenticatorFactory"
2    filter="(instance=default)"
3    interface="com.sybase365.mobiliser.framework.gateway.security.api.
         InternalServiceCallAuthenticatorFactory"
4    sticky="false" />
```

Listing 5.51: InternalServiceCallAuthenticatorFactory from OSGi

The credential interface is a simple value object:

```
1 public interface IInternalServiceCallAuthenticationData {
2
3     String getUsername();
4
5 }
```

Listing 5.52: IInternalServiceCallAuthenticationData API

The framework doesn't provide a default implementation of this interface, though Mobiliser does have one based on preferences values.

Get the service you want to use through the OSGi service registry (has to be available per the description from above).

```
<osgi:reference id="customEndpoint"
  interface="com.sybase365.mobiliser.money.services.custom.ICustom1Endpoint" />
```

Listing 5.53: Internal Service from OSGi

This service must now be wrapped with the authentication aspect, which can then be injected into your classes as the ICustom1Endpoint to use for service invocation. We are also assuming you have some how created an IInternalServiceCallAuthenticationData instance with the bean name internalAuthData.

```
<mobconfig:mobiliser-internal-service id="authenticatingcustomEndpoint"
  endpoint-ref="customEndpoint"
  proxy-interfaces="com.sybase365.mobiliser.money.services.custom.
      ICustom1Endpoint" />
```

Listing 5.54: Wrapped Internal Service from OSGi

Since we used **internalAuthenticatorFactory** and **internalAuthData** as bean names, some of the configuration was automatic. Otherwise you'll need two more attributes.

```
<mobconfig:mobiliser-internal-service id="authenticatingcustomEndpoint"
  endpoint-ref="customEndpoint"
  authenticator-factory-ref="authFactory"
  authdata-ref="authConfiguration"
  proxy-interfaces="com.sybase365.mobiliser.money.services.custom.
      ICustom1Endpoint" />
```

Listing 5.55: Wrapped Internal Service from OSGi with custom bean names

5.9.2 Outside the Container

To ease remote access to the services provided by Mobiliser, we have chosen to make all endpoints implement an interface. This interface contains the service methods the endpoint can process (pairs of MobiliserRequestTypes / MobiliserResponseType). These are packaged in a separate bundle and can be used by clients to call into the services remotely

IServiceClientFactory IServiceClientFactory instances can create dynamic clients based on a given interface and endpoint url. The respective factory types determine how the clients talk to the backend (SOAP, POST XML, POST JSON, etc).

Rest To create RESTful clients, use com.sybase365.mobiliser.util.tools.clientutils.rest.RestClientFactory. It needs a clienttype during construction which is one of:

- **XML**
- **JSON**

The url provided to the factory should be the base URL without any operation name. The operation names will be calculated dynamically from the invoked method.

SOAP To create a SOAP client, use com.sybase365.mobiliser.util.tools.clientutils.soap.SoapClientFactory. You can specify a SoapMessageFactory at construction otherwise a SaajSoapMessageFactory will be created for you.

Usage After construction of the factory, you can retrieve the actual client like this:

```
1 final IAuthorisationEndpoint soapClient = soapFactory.createClient(
      IAuthorisationEndpoint.class, "http://localhost:8080/mobiliser/transaction");
2 final IAuthorisationEndpoint restClient = restFactory.createClient(
      IAuthorisationEndpoint.class, "http://localhost:8080/mobiliser/rest/
      transaction");
```

Listing 5.56: Get Client From Factory

As you can see, the configured url must match the client factory type you are using.

Refreshing If you would like your client to be dynamically reconfigured based on changes pulled from the preferences you can do that with standard Mobiliser classes. An example of a dynamically refreshable soap client:

```
1   <bean id="soapClientFactory"
2     class="com.sybase365.mobiliser.util.tools.clientutils.soap.SoapClientFactory"
          />
3
4   <bean id="serviceConfiguration" class="MyConfigurationClass">
5     <property name="preferences" ref="prefsNode" />
6   </bean>
7
8   <bean id="transactionClientSource"
9     class="com.sybase365.mobiliser.util.tools.clientutils.api.
          RefreshableClientTargetSource">
10    <property name="clientFactory" ref="soapClientFactory" />
11    <property name="configuration" ref="serviceConfiguration" />
12    <property name="clientInterface"
13      value="com.sybase365.mobiliser.api.TransactionEndpoint" />
14  </bean>
15
16  <bean
17    class="com.sybase365.mobiliser.util.prefs.util.
          RegisterChangeListenerFactoryBean">
18    <constructor-arg ref="prefsNode" />
19    <constructor-arg ref="transactionClientSource" />
20  </bean>
21
22  <bean id="transactionClient" class="org.springframework.aop.framework.
        ProxyFactoryBean">
23    <property name="targetSource" ref="transactionClientSource" />
24    <property name="interfaces">
25      <list>
26        <value>com.sybase365.mobiliser.api.TransactionEndpoint</value>
27      </list>
28    </property>
29  </bean>
```

Listing 5.57: Refreshable Mobiliser Clients

You need to implement the interface com.sybase365.mobiliser.util.tools.clientutils.api.IClientConfiguration so that we have a standard way to access the client data. Your implementation will most likely pull the information out of a preferences node.

```
1 public interface IClientConfiguration {
2
3     String getMobiliserEndpointUrl();
4
5     String getWsPassword();
6
7     String getWsUserName();
8 }
```

Listing 5.58: IClientConfiguration API

For the SOAP case mentioned above, the implementation should return:

`http://localhost:8080/mobiliser/transaction`

Optionally, we could use the same configuration for multiple clients. Then the URL would be:

`http://localhost:8080/mobiliser`

We would set one more property on our RefreshableClientTargetSource:

```
<property name="endpointSuffix" value="/transaction" />
```

Chapter 6

Audit

The Mobiliser gateway comes preconfigured with a pluggable audit framework.

Auditing happens at the message dispatch level and is implemented as a MethodInterceptor, wrapping your implemented endpoint.

The method interceptor collects IAuditManager implementations from the OSGi registry and dispatches the results of a service call to each registered audit manager asynchronously. This means each audit manager has its own dedicated ExecutorService.

com.sybase365.mobiliser.framework.service.audit.api.IAuditManager

The auditing aspect for endpoints leverages the OSGi service registry to track audit managers. Each audit manager will be called asynchronously with the results of the service call. Managers must not alter the input data, though currently this is not explicitly enforced. Doing so would however be a grave mistake.

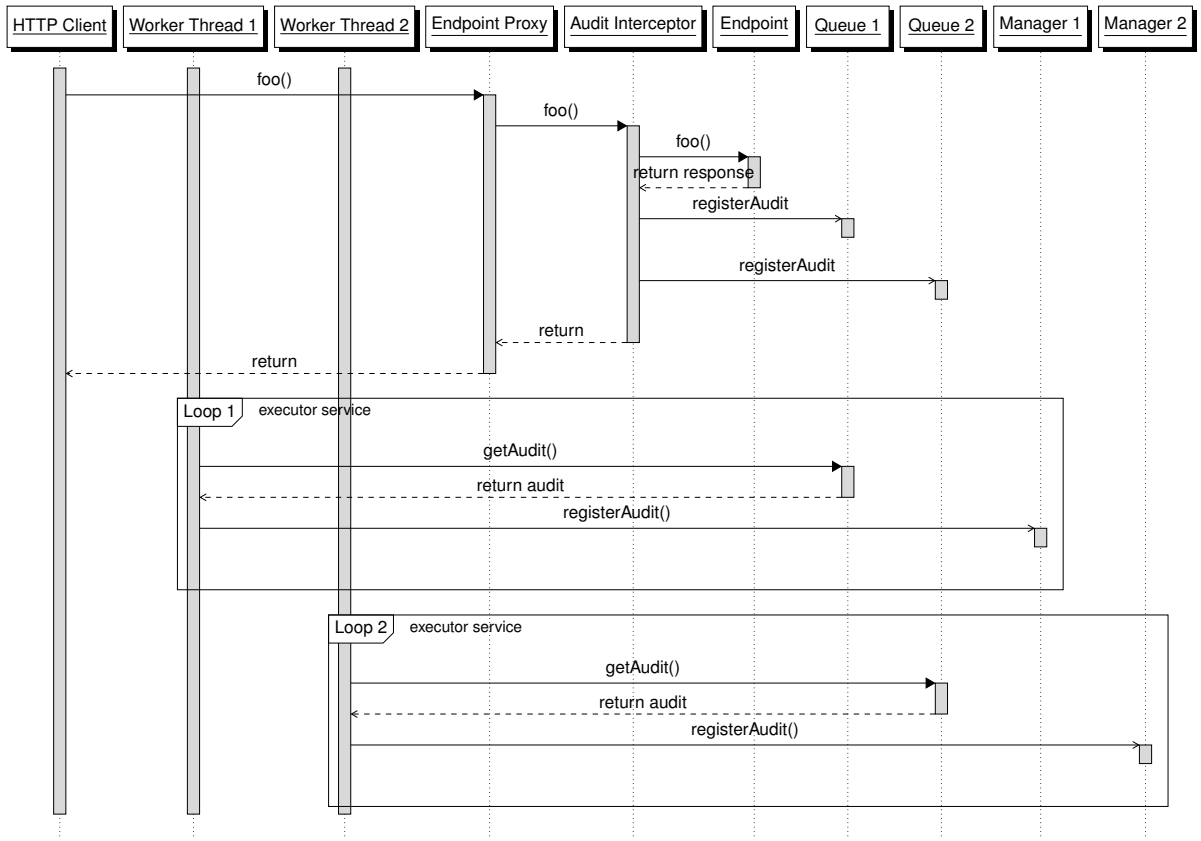

Figure 6.1: Audit Sequence Diagram

```
    boolean isAuditEnabled();

    void registerAudit(final MobiliserRequestType request,
            final MobiliserResponseType response, final long actor,
            final long executionMs, final int resultCode,
            final Map<String, Object> additionalParameters);

    void registerAudit(final MobiliserRequestType request,
            final Throwable exception, final long actor,
            final long executionMs,
            final Map<String, Object> additionalParameters);
```

Listing 6.1: “Audit Context Java Interface” Audit Interface

The audit aspect will call one of the two methods after the method has completed successfully or resulted in an exception. The aspect will also not dispatch any results for managers which respond to isAuditEnabled with false.

During a call to a service method, a thread local is bound through which client code is able to set additional parameters. These then get passed to the audit manager as seen in the interface above. This is a rarely needed feature and also ties the service implementation to the audit manager code, so use with care. If using this feature, care must be taken to properly handle the thread-local reference to the IAuditContext. If your service implementation does asynchronous work, you'll need to pull out the reference to the context and somehow make it available to the running threads. Also note that the default implementation is not thread safe since it is based on a thread local and assumes it will only be used from that one thread.

```
public interface IAuditContext {

    /**
     * Returns all the audit parameters held by the context. The map returned by
     * this method is live, but may not be changed other than through
     * {@link #setAuditParameter(String, Object)}.
     *
     * @return the audit parameters
     */
    Map<String, Object> getAuditParameters();

    /**
     * Sets the given audit parameter
     *
     * @param key
     * @param value
     */
    void setAuditParameter(final String key, final Object value);
}
```

Listing 6.2: “Audit Context Interface” Audit Interface

```
public interface IAuditContextHolderStrategy {
    /**
     * Clears the current context.
     */
    void clearContext();

    /**
     * Obtains the current context.
     *
     * @return a context (never <code>null</code> - create a default
     *         implementation if necessary)
     */
    IAuditContext getContext();

    /**
```

```
16      * Sets the current context.
17      *
18      * @param context
19      *             to the new argument (should never be <code>null</code>,
20      *             although implementations must check if <code>null</code> has
21      *             been passed and throw an <code>IllegalArgumentException</code>
22      *             in such cases)
23      */
24     void setContext(IAuditContext context);
25 }
```

Listing 6.3: “Audit Context Holder Interface” Audit Context Holder Interface

Chapter 7

Security

One of the major components in the Mobiliser architecture is the security layer. The security layer is responsible for authenticating and authorising access to the various Mobiliser services.

7.1 Filters

If you need to implement a custom security filter, you'll implement a **javax.servlet.Filter** (probably using **GenericFilterBean**) and export it in a special way through the registry so it gets added to this filter chain. For this purpose, the security API has a simple interface ISecurityFilter:

```
1 public interface ISecurityFilter {
2
3     /**
4      * Returns the filter
5      *
6      * @return the filter
7      */
8     Filter getFilter();
9 }
```

Listing 7.1: ISecurityFilter API

The tricky part now is to correctly place your filter in the correct order in the list. For this to work, your filter implementation should implement **org.springframework.core.Ordered** and provided a numeric value which is used to sort the filter into the list. Please note that you should follow the recommendations noted in the Spring Security documentation about adding custom filters.[1]

7.2 Authentication

To implement a custom authentication provider, you will need to hook in to Spring Security with your custom UserDetailsService. The authentication filters will delegate authentication to a list of **AuthenticationProviders** which in many cases is a DaoAuthenticationProvider. Your custom **UserDetailsService** will allow the provider to fetch the required customer information to allow authentication to continue.

```
1 public interface UserDetailsService {
2     UserDetails loadUserByUsername(String username)
3         throws UsernameNotFoundException, DataAccessException;
4 }
```

[1] Custom Filters http://static.springsource.org/spring-security/site/docs/3.0.x/reference/springsecurity-single.html#ns-custom-filters

Listing 7.2: UserDetailsService API

This is standard spring security with the filters pulling out the user to be authenticated and you providing his roles, status (expired, locked, etc), and password. Your customisation could retrieve this data from a webservice, a database, flat files etc.

This implementation must be exported through the OSGi registry:

```
1   <osgi:service ref="userDetailsService"
2     interface="org.springframework.security.core.userdetails.UserDetailsService"
3     ranking="10">
4     <osgi:service-properties>
5       <beans:entry key="origin" value="mobiliser" />
6     </osgi:service-properties>
7   </osgi:service>
```

Listing 7.3: UserDetailsService OSGi Export

The authentication provider will also need a SaltSource and a PasswordEncoder so the provider can correctly verify authentication data coming from your UserDetailsService. Again this is all standard Spring Security, but for completeness, the interfaces are as follows:

```
1 public interface SaltSource {
2     Object getSalt(UserDetails user);
3 }
```

Listing 7.4: SaltSource API

```
1 public interface PasswordEncoder {
2
3     String encodePassword(String rawPass, Object salt)
4         throws DataAccessException;
5
6     boolean isPasswordValid(String encPass, String rawPass, Object salt)
7         throws DataAccessException;
8 }
```

Listing 7.5: PasswordEncoder API

Your implementation must be exported through the registry as well:

```
1   <osgi:reference id="passwordEncoder"
2     interface="org.springframework.security.authentication.encoding.
          PasswordEncoder"
3     sticky="false" />
4   <osgi:reference id="saltSource"
5     interface="org.springframework.security.authentication.dao.SaltSource"
6     sticky="false" />
```

Listing 7.6: PasswordEncoder and SaltSource Export

So your UserDetailsService will give the authentication provider the hashed password and then your implementation will return the salt for that user which is used by the password encoder to create a hash of the raw password the user supplied. Then it is just a matter of a simple match.

You could also implement a complete authentication provider and export it through the registry. This should not be required, but you could bundle one of the existing ones from Spring Security and export it like this:

```
1   <osgi:service ref="customAuthenticationProvider"
2     interface="org.springframework.security.authentication.AuthenticationProvider
          "
```

```
3      ranking="10">
4      <osgi:service-properties>
5        <beans:entry key="origin" value="mobiliser" />
6      </osgi:service-properties>
7    </osgi:service>
```

Listing 7.7: Custom Authentication Provider Export

You will still have to implement and export a UserDetailsService since it is needed in other places in the security framework, notably in the RememberMeAuthenticationProvider.

Chapter 8

Preferences

Preferences are the standard mechanism for application configuration in Mobiliser. They are mainly used to manage operating level configuration data such as URLs, user name, timeouts, retries when communicating with other systems, or thread/object pool size and the like. In other situations, they are used for business logic configuration like names of message templates or the customer type id used for creating new customers. They replace the use of properties files seen in many other projects or components. See “Mobiliser Framework Architecture and Design” guide for

8.1 Using Preferences

There are various possibilities to make use of Preferences. We will only focus on the most common and easiest ways.

First you need to make sure that the namespaces for the prefs schema extension is correctly installed in your bundle-context.xml:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:beans="http://www.springframework.org/schema/beans"
5   xmlns:util="http://www.springframework.org/schema/util"
6   xmlns:tx="http://www.springframework.org/schema/tx"
7   xmlns:prefs="http://www.sybase.com/mobiliser/util/prefs/config"
8   xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
        springframework.org/schema/beans/spring-beans-3.0.xsd
9     http://www.springframework.org/schema/util http://www.springframework.org/
          schema/util/spring-util-3.0.xsd
10    http://www.springframework.org/schema/tx http://www.springframework.org/
          schema/tx/spring-tx-3.0.xsd
11    http://www.sybase.com/mobiliser/util/prefs/config http://www.sybase.com/
          mobiliser/util/prefs/config/sybase-prefs-config-5.0.xsd">
```

Listing 8.1: Preferences Namespace Import

Once the schema extension is in place you can retrieve Preference nodes like this:

```
1 <!-- PREFERENCES CONFIGURATION - simple example -->
2 <prefs:node id="customAuthorisationPreferences" auto-decrypt="true"
3   class="com.sybase365.mobiliser.custom.project.businesslogic.impl.
        CustomAuthorisation" />
```

Listing 8.2: Preferences Retrieve Node

The class specified in there is only used for retrieving the correct path from Preferences. The created prefs:node

always implements the IPreferences interface and represents one node in the Preferences tree.

This node gets injected into the actual bean that is used to access the Preferences:

```
<bean id="customAuthService" parent="abstractService"
  class="com.sybase365.mobiliser.custom.project.businesslogic.impl.
      CustomAuthorisation">
  <property name="blacklistAction" ref="blacklistAction" />
  <property name="preferences" ref="customAuthorisationPreferences" />
</bean>
```

Listing 8.3: Preferences Use Retrieved Node

From there you can work with the preferences object:

```
public class CustomAuthorisation {
  private IPreferences preferences;

  // omitted

  if (this.preferences.getBoolean("disableBlacklistCheck", false)) {
    // omitted
  } else {
    LOG.debug("Blacklist check disabled as per preferences configuration.");
  }
}
```

Listing 8.4: Preferences Use Retrieved Node in Java

Another nice way to configure a POJO which knows nothing of our preferences is to use a listener. Assuming we have the node as configured above, we can change the example like this:

```
public class CustomAuthorisation {
  private volatile boolean disableBlacklistCheck;

  // omitted

  if (disableBlacklistCheck) {
    // omitted
  } else {
    LOG.debug("Blacklist check disabled as per preferences configuration.");
  }
}
```

Listing 8.5: Preference Value Injected

```
  <bean id="customAuthorisation"
    class="com.sybase365.mobiliser.custom.project.businesslogic.impl.
        CustomAuthorisation" depends-on="
        customAuthorisationPrefChangeListenerRegistrar">
    <property name="disableBlacklistCheck">
      <prefs:value node="prefsNode" key="disableBlacklistCheck"
        default="false" depends-on="
            customAuthorisationPrefChangeListenerRegistrar" />
    </property>
  </bean>

  <prefs:node id="prefsNode"
    class="com.sybase365.mobiliser.custom.project.businesslogic.impl.
        CustomAuthorisation"
    change-listeners="customAuthorisationPrefChangeListener"
    change-listeners-factory-names="
        customAuthorisationPrefChangeListenerRegistrar" />
```

```
13
14   <bean id="customAuthorisationPrefChangeListener"
15     class="com.sybase365.mobiliser.util.prefs.util.
           GenericPropertyPreferenceChangeListener">
16     <property name="targetBeanName" value="customAuthorisation" />
17   </bean>
```

Listing 8.6: Using A Preferences Listener

So now our code has no dependency on the preferences and is a simple POJO, making testing simpler since we no longer need to mock the preferences node. The registrar bean is only there to enforce the ordering. That way we register the listener, then read the value. The GenericPropertyPreferenceChangeListener will inject preferences values mapping preferences keys onto Java bean properties. Notice that the member field is marked volatile. This was on purpose since our runtime updates will happen concurrently. If you expect values to change and configure your POJOs as such, then you must also implement them properly to handle those runtime updates.

For implementing a custom listener, you have to implement this simple interface:

```
1 public interface IPreferenceChangeListener {
2     void preferenceChanged(final IPreferenceChangeEvent evt);
3 }
```

Listing 8.7: Preference Lister API

8.2 Accessing Preferences from within the container

In case you want to use Preferences from within the container, you just need to import the preferences service from the OSGi registry:

```
1 <osgi:reference id="preferencesService"
2   interface="com.sybase365.mobiliser.util.prefs.api.IPreferencesService" />
3 <osgi:reference id="encryptionManager"
4   interface="com.sybase365.mobiliser.util.prefs.encryption.api.
       IPreferencesEncryptionManager" />
```

Listing 8.8: OSGi Config for Preferences

We also imported the encryption manager that is responsible for decrypting encrypted values when reading from the preferences. This is switched on by setting the attribute auto-decrypt="true" on the prefs:node config. Preferences values may also use embedded system properties as described previously. You can disable system property replacement with resolve-system-properties="false".

There is a container-wide configuration that is used:
The file com.sybase365.mobiliser.util.prefs.store.local.properties is located in the conf/cfgbackup directory. It has the following configuration options:

Key	Default	Description
preferencesAppName	businesslayer	The name of the preferences application to retrieve.
refreshInterval	300000	Time in milliseconds between preferences refresh runs.
refreshIntervalFailed	300000	If refreshing the preferences fails for whatever reason, the time in milliseconds to wait before the next try.

Table 8.1: Local Preferences Configuration Options

That will set the application the whole osgi container uses to “businesslayer” and client code needs to have no knowledge that even such a thing exists.

Encryption is also configured container-wide with another configuration file based on the encryption scheme configured in the container. Currently, preferences may be encrypted using AES-128, AES-256 or 3DES. AES-256 is only available if the unlimited policy files have been installed in the JDK. This will be automatically detected. The configuration files are named:

- com.sybase365.mobiliser.util.prefs.encryption.aes.properties and
- com.sybase365.mobiliser.util.prefs.encryption.tripledes.properties

respectively.

They both have the following configuration option:

Key	Default	Description
preferencesEncryptionKey	secret	The password used to decrypt encrypted preferences.

Table 8.2: Local Preferences Encryption Configuration Options

8.3 Accessing Preferences outside the container

As we said in the beginning, we can use the same interfaces and implementations inside or outside of the OSGi container. This would be the case when using preferences in the web applications. We just need a properly configured backing store. While in the container, the backing store reads the preferences data directly out of the database by accessing a business logic class, outside of the container we can use the PreferencesService published through the Mobiliser gateway. To do this, we use the bundle: com.sybase365.mobiliser.util.prefs:com.sybase365.mobiliser.util.prefs.store.remote.

```
<bean id="backingStore"
  class="com.sybase365.mobiliser.util.prefs.store.remote.
      MagicRemoteBackingStoreFactoryBean" />

<!-- the backing store is simultaneously the refresh config and the encryption
    strategy -->
<alias name="backingStore" alias="prefsConfiguration" />
<alias name="backingStore" alias="encryptionStrategy" />

<bean id="prefsService"
  class="com.sybase365.mobiliser.util.prefs.impl.PreferencesServiceFactoryBean">
  <constructor-arg ref="backingStore" />
  <constructor-arg ref="prefsConfiguration" />
</bean>
```

Listing 8.9: Config for Preferences running outside the Container

This is really automagic, but works in a similar fashion to the old 2.6.x prefs.

8.3.1 Configuration

MagicRemoteBackingStoreFactoryBean requires that you already have configured the preferences configuration data somewhere. The configuration options are:

Key	Example	Description
serverUrl	http://localhost:8080/mobiliser/rest/prefs	URL of the preferences service.
applicationIdentifier	presentationlayer	The preferences application (tree) to access.
userName	mobiliser	HTTP user name to access the service if needed.
password	secret	HTTP password to access the service if needed.
clientType	json	Type of client (ie. json/xml/soap).
pollInterval	60000	refresh interval in milliseconds.
secret	paybox	The password to decrypt encrypted preferences.

Table 8.3: Remote Preferences Configuration Options

The client type is optional, but defaults to JSON if Jackson[1] is found on the classpath.

Client Type	Description
json	send HTTP posts with JSON content to backend
xml	send HTTP posts with XML content to backend
soap	send SOAP requests to backend

Table 8.4: Remote Preferences Client Type Options

The bundle itself has optional dependencies on the client types, and will only try to load those it finds on the classpath (soap, rest, with rest dependent upon Jackson and Jaxb2 being on the classpath).

Currently you can configure the options in one of three ways. The following list contains them in decreasing priority.

- System Properties
- JNDI
- Properties Files

System Properties

The first place the MagicRemoteBackingStoreFactoryBean looks for configuration data is in the system properties (like by setting them in bin/setenv.sh in Tomcat).

```
1 com.sybase365.mobiliser.money.prefs.remote.url=prefs://
    username:password@localhost:8080/mobiliser/rest/prefs?pollInterval=60000&
    clientType=json&applicationIdentifier=presentationlayer
2 com.sybase365.mobiliser.money.prefs.secret=paybox
```

Listing 8.10: System Property Preferences Configuration

JNDI

The second place MagicRemoteBackingStoreFactoryBean looks for configuration data is in JNDI entries (like by setting them in conf/context.xml in Tomcat).

```
1 <Environment description="The prefs2 configuration for web ui"
```

[1] Jackson Homepage http://jackson.codehaus.org/

```
  name="prefs2/config" type="java.lang.String"
  value="prefs://username:password@localhost:8080/mobiliser/rest/prefs?
      pollInterval=60000&amp;clientType=json&amp;applicationIdentifier=
      presentationlayer"
/>
<Environment description="The prefs secret key" name="prefs/secret"
  type="java.lang.String" value="paybox" />
```

Listing 8.11: JNDI Preferences Configuration

Properties File

The third place MagicRemoteBackingStoreFactoryBean looks for configuration data is in a properties file name sybase-preferences.properties in the root of the classpath (like in WEB-INF/classes in a tomcat context).

```
serverUrl=http\://localhost\:8080/mobiliser/rest/prefs
applicationIdentifier=presentationlayer
userName=username
password=password
clientType=json
encryption-secret=paybox
```

Listing 8.12: JNDI Preferences Configuration

8.4 Extending Preferences

8.4.1 Encryption

You can implement additional encryption types for the preferences.

To add a new encryption type implement these two interfaces:

```
public interface IEncryptionStrategyFactory {

    String getPrefix();

    boolean supportsType(final String type);

    IPreferencesEncryptionStrategy createEncryptionStrategy(
            final Properties properties);
}

public interface IPreferencesEncryptionStrategy {

    String encryptValue(final String preference);

    String decryptValue(final String encryptedPreference);

}
```

Listing 8.13: Preferences Encryption APIs

Then export them through the OSGi registry:

```
  <osgi:service ref="encryptionStrategyFactory"
    interface="com.sybase365.mobiliser.util.prefs.encryption.api.
        IEncryptionStrategyFactory"
    ranking="10">
```

```
4     <osgi:service-properties>
5       <beans:entry key="origin" value="mobiliser" />
6     </osgi:service-properties>
7   </osgi:service>
8   <osgi:service ref="encryptionStrategy"
9     interface="com.sybase365.mobiliser.util.prefs.encryption.api.
          IPreferencesEncryptionStrategy"
10    ranking="10">
11    <osgi:service-properties>
12      <beans:entry key="type" value="BLOWFISH" />
13      <beans:entry key="origin" value="mobiliser" />
14    </osgi:service-properties>
15  </osgi:service>
```

Listing 8.14: OSGi Export of Encryption Impl

The decryption interceptor inspects preferences values and if they have an encryption prefix, they are decrypted on the fly using the correct strategy retrieved from the encryption manager before being returned.

8.4.2 Interceptor

As mentioned in the design document, additional functionality is added by implementing method interceptors. If you find you need to add something, you will need to implement the method interceptor and extend the standard factory bean to add your interceptor to the advisor list.

```
1  public final class CustomInterceptor implements MethodInterceptor {
2      @Override
3      public Object invoke(final MethodInvocation invocation) throws Throwable {
4
5          final String key = (String) invocation.getArguments()[0];
6          final String result = (String) invocation.proceed();
7
8          if (!StringUtils.hasLength(result)) {
9              return result;
10         }
11
12         // do something with the value or transform it
13         return result;
14     }
15 }
```

Listing 8.15: Example No-Op Interceptor

```
1  public final class CustomNodeFactoryBean extends
2          PreferencesNodeFactoryBean {
3
4
5      /**
6       * @param preferencesService
7       * @param clazz
8       */
9      public CustomNodeFactoryBean(
10             final IPreferencesService preferencesService, final Class<?> clazz) {
11         super(preferencesService, clazz);
12     }
13
14     /**
15      * @param path
16      * @param preferencesService
17      */
18     public CustomNodeFactoryBean(final String path,
19             final IPreferencesService preferencesService) {
```

```
20          super(path, preferencesService);
21      }
22
23      @Override
24      protected List<Advisor> getAdvisorList(final IPreferences instance) {
25
26          final CustomInterceptor interceptor = new CustomInterceptor();
27          interceptor.afterPropertiesSet();
28
29          final AspectJExpressionPointcutAdvisor advisor = new
                AspectJExpressionPointcutAdvisor();
30          advisor.setExpression("execution(* " + IPreferences.class.getName()
31                  + ".get(..))");
32
33          // start aspectj init hack
34          // swap the classloader for aspectj's init so it can see the target
35          // class and then swap it back after the expression has been compiled
36          // once
37          final Thread currentThread = Thread.currentThread();
38
39          final ClassLoader current = currentThread.getContextClassLoader();
40          try {
41              currentThread.setContextClassLoader(instance.getClass()
42                      .getClassLoader());
43
44              // force init
45              advisor.getPointcut().getMethodMatcher();
46
47          } finally {
48              currentThread.setContextClassLoader(current);
49          }
50
51          // end aspectj init hack
52
53          advisor.setAdvice(interceptor);
54
55          return Collections.singletonList((Advisor) advisor);
56      }
57  }
```

Listing 8.16: Custom Factory Bean

Unfortunately you cannot use the spring schema extension with your new factory bean and will have to do the full configuration manually.

Chapter 9

Events and Alerts

9.1 Event Generation API

The figure below shows the basic class hierarchy from which events are created and processed.

The following code blocks show the important methods from the class hierarchy.

9.1.1 Header And Body

Events are defined through the base abstract class:

`com.sybase365.mobiliser.framework.event.model.Event`

This class maintains data associated with an event and includes header, body and (optional) delay information. This class exposes only one abstract method that must be implemented by different sub-classes;

```
1 /**
2  * Abstracted initialisation code for your events called
3  * when any Event is instantiated.
4  */
5 public abstract void init();
```

Listing 9.1: com.sybase365.mobiliser.framework.event.model.Event

Header

The event header information managed by the Event class mainly contains internal control information that is not required to be managed by the event generator or handler. The only header value that needs be set is the **event name**.

Note: Each event must provide a name that uniquely identifies it to the event system. For Event sub-classes, the default value for the event name will be the Event sub-class name (as provided by Class.getSimpleName()). When using the event factory, the name is required on the event creation method call.

Note: The event name must not exceed 100 characters in length, as limited by the database storage.

Body

The event body manages a set of data values, each identified by a key.

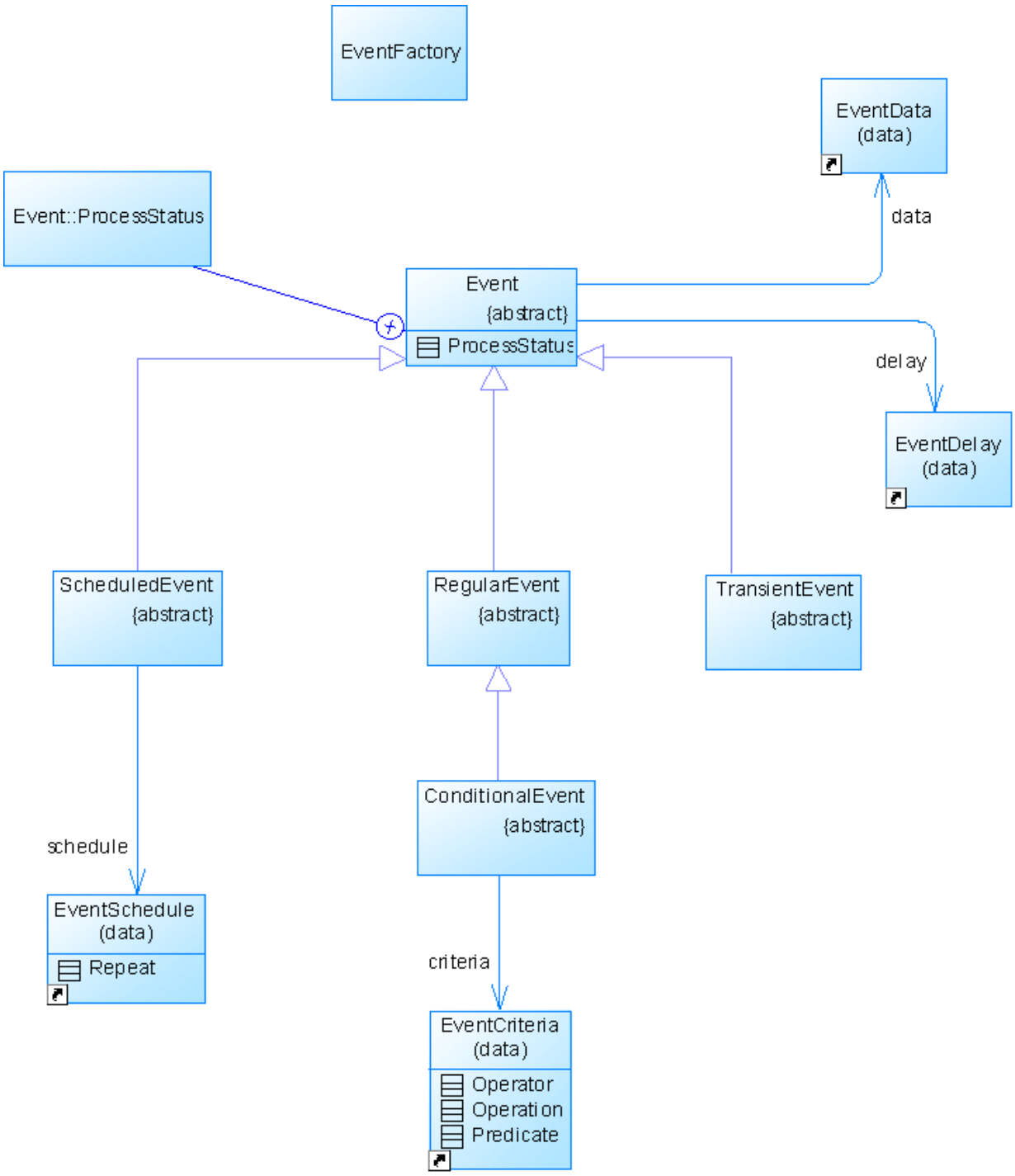

Figure 9.1: Components For Event Generation

This class maintains a;

`java.util.HashMap`

for storage of those entries.

The event body is managed by the class;

```
1 ...
2 /**
```

```
 * Standard constructor - just wrap a HashMap
 */
public EventData(HashMap<String, Object> value) { ... }

/**
 * Empty constructor - create a new HashMap
 */
public EventData() { ... }

/**
 * Set the wrapped HashMap for this event data
 */
public void setData(HashMap<String, Object> value) { ... }

/**
 * Get the wrapped HashMap for this event data
 */
public HashMap<String, Object> getData() { ... }

/**
 * Check if event data contains the key value
 */
public boolean containsKey(String key) { ... }

/**
 * Get the event data value for the specified key
 */
public Object get(String key) { ... }

/**
 * Put the event data value for the specified key
 */
public void put(String key, Object value) { ... }
...
```

Listing 9.2: com.sybase365.mobiliser.framework.event.model.data.EventData

Note: The data values are expected to be of a String type when entered into the event data and will be stored as string values in the database. Use of other datatypes is allowed, but is determined by the generator and handler mapping those datatype values to and from Strings.

Note: The data key must not exceed 100 characters in length and the data value must not exceed 1000 characters in length, as limited by the database storage.

Delay (Optional)

Events may be assigned a short delay that indicates a short period of time after which the event should then be attempted to be processed.

An event delay is different from a scheduled event; a scheduled event is triggered at a known fixed point in time in the future, maybe hours, days or weeks, and it may optionally repeat at a know interval. An event delay is a short period of time from the current system time and is expected to be specified in milliseconds, seconds or minutes (although periods of time represent by hours and days can be used too).

The EventDelay class allows a easily understood DSL-like syntax for creating delays. For example;

```
...
EventDelay oneSecDelay = new EventDelay().delayFor(1L).seconds())
EventDelay().delayFor(1L).seconds());
myEvent.setDelay(oneSecDelay);
myEvent.create();
...
```

Listing 9.3: Delay For 1 Second

or

```
1 ...
2 EventDelay oneMinDelay = new EventDelay().minutes(1L))
3 EventDelay().minutes(1L));
4 myEvent.setDelay(oneMinDelay);
5 myEvent.create();
6 ...
```

Listing 9.4: Delay For 1 Minute

The event delay os manage by the class

```
1 ...
2 /**
3  * Specify delay value that will be described by following method.
4  * E.g. new EventDelay().delayFor(1L).minute();
5  */
6 public EventDelay delayFor(long value) { ... }
7
8 /**
9 * Assign value provided to delayFor(long) method as value in milliseconds
10 */
11 public EventDelay milliseconds() { ... }
12
13 /**
14  * Assign value provided directly as a millisecond delay value
15  */
16 public EventDelay milliseconds(long value) { ... }
17
18 /**
19  * Assign value provided to delayFor(long) method as value in seconds
20  */
21 public EventDelay seconds() { ... }
22
23 /**
24  * Assign value provided directly as a second delay value
25  */
26 public EventDelay seconds(long value) { ... }
27
28 /**
29  * Assign value provided to delayFor(long) method as value in minutes
30  */
31  public EventDelay minutes() { ... }
32
33 /**
34  * Assign value provided directly as a minute delay value
35  */
36  public EventDelay minutes(long value) { ... }
37
38 /**
39  * Assign value provided to delayFor(long) method as value in hours
40  */
41 public EventDelay hours() { ... }
42
43 /**
44  * Assign value provided directly as a hour delay value
45  */
46 public EventDelay hours(long value) { ... }
47
48 /**
```

```
49  * Assign value provided to delayFor(long) method as value in days
50  */
51 public EventDelay days() { ... }
52
53 /**
54  * Assign value provided directly as a day delay value
55  */
56 public EventDelay days(long value) { ... }
57 ...
```

Listing 9.5: com.sybase365.mobiliser.framework.event.model.data.EventDelay

9.1.2 Event Factory

The event factory class works as a generalized way of creating new anonymously named sub-classes of the different event types, using static methods. It also provides for an interface to cancel existing scheduled events, using the scheduled event it or name.

For example;

```
1 ...
2 HashMap<String, Object> mapData = new HashMap<String,Object>();
3 mapData.put("customertype","BB");
4
5 RegularEvent event = EventFactory.generateRegularEvent("AnonymousRegularEvent",
      mapData);
6
7 event.create();
8 ...
```

Listing 9.6: Event Factory Example 1

or

```
1 ...
2 EventFactory.create(event);
3 ...
```

Listing 9.7: Event Factory Example 2

```
1 ...
2 /**
3  * Generate a regular event object.
4  */
5 public static RegularEvent generateRegularEvent(final String nameValue, final
      HashMap<String,Object> mapData) { ... }
6
7 /**
8  * Generate a conditional event object with the specified criteria.
9  */
10 public static ConditionalEvent generateConditionalEvent(final String nameValue,
      final HashMap<String,Object> mapData, final EventCriteria value) { ... }
11
12 /**
13  * Generate a scheduled event object with the specified schedule.
14  */
15 public static ScheduledEvent( final String nameValue, final HashMap<String,Object
      > mapData, final EventSchedule value) { ... }
16
17 /**
18  * Generate a transient event object.
```

```
19  */
20 public static TransientEvent generateTransientEvent(final String nameValue,final
       HashMap<String,Object> mapData) { ... }
21
22 /**
23  * Main entry point for new event processing.
24  */
25 public static boolean create(Event event) { ... }
26
27 /**
28  * Cancel a scheduled event using it's unique scheduled id
29  */
30 public static boolean cancelScheduledEvent(final long scheduledId) { ... }
31
32 /**
33  * Cancel scheduled event(s) by event name
34  */
35 public static boolean cancelScheduledEvent(final String eventName) { ... }
36 ...
```

Listing 9.8: com.sybase365.mobiliser.framework.event.model.EventFactory

9.1.3 Regular Events

This class of event has the following features:

- Maintains header, data and optional delay information, as described in the section above.
- Event and event data is persisted in the database.
- Handler processing status is persisted in the database and updated after processing is complete.
- Depending on the handler expiry settings (see below), it is subject to handler catchup processing.

It is recommended to the extend the abstract class:

`com.sybase365.mobiliser.framework.event.model.RegularEvent`

when it is useful to provide helper methods around the event data, for example:

```
1  ...
2  public class MyRegularEvent extends RegularEvent {
3
4  public MyRegularEvent() {
5      super();
6  }
7
8  public MyRegularEvent(EventData data) {
9      super(data);
10 }
11
12 @Override
13 public void init() {
14     this.setName("CustomerTxnEvent");
15 }
16
17 public void setCustomerType(String customerType) {
18     this.getData().put("customertype", customerType);
19 }
20
```

```
21 public String getCustomerType() {
22     return this.getData().get("customertype");
23 }
24
25 public void setTxnType(String txnType) {
26     this.getData().put("txntype", customerType);
27 }
28
29 public String getTxnType() {
30     return this.getData().get("txntype");
31 }
32 }
```

Listing 9.9: Regular Event Example

An example of generation of this example event is:

```
1 ...
2     MyRegularEvent event =new MyRegularEvent();
3
4     event.setCustomerType(``BB'');
5     event.setTxnType(``01'');
6
7     event.create();
8 ...
```

Listing 9.10: Regular Event Generation Example

9.1.4 Conditional Events

This class of event has the same features as the RegularEvent, plus:

- Maintains a set of criteria, against which the event data will be evaluated when passed into the event generator.
- If the criteria evaluation passes (by the generator running the validate() method on the event), the event is processed as a RegularEvent.
- If the criteria evaluation fails the event is ignored and not processed.

The condition criteria for a conditional event could be managed in different ways through sub-classes of abstract ConditionalEvent class. Currently supported conditional events include:

- Criteria Conditional Events
- Delegator Conditional Events
- Expression Conditional Events

Criteria Conditional Events

The condition evaluation for Criteria Conditional Events is managed by the EventCriteria class, which is associated with the abstract CriteriaConditionalEvent class.

The event evaluation criteria is a list of predicates that must all evaluate to true before the event will be validated. A predicate is specified in the form;

<event data key value> <operation> <comparison value>

The EventCriteria class allows a easily understood DSL-like syntax for creating criteria. For example:

```
...
    EventCriteria criteria = new EventCriteria().where("customertype").eq("BB").
        and("txntype!).eq("01");
...
```

Listing 9.11: Event Criteria Example

Note: Typically, use of criteria conditional events is *recommended* by extending from the abstract class:

`com.sybase365.mobiliser.framework.event.model.CriteriaConditionalEvent`

This allows for the specification of the event criteria in the definition of event and separates the criteria specification (in the event class) from the criteria evaluation (when the event data is known; at the point of event object instantiation).

Although it is possible to use the event factory to create an anonymous ConditionalEvent, it is less useful because the point of criteria specification and criteria evaluation are both specified at the point of event object instantiation.

The following class shows an example of extending the ConditionalEvent abstract class:

```
...
public class MyCriteriaConditionalEvent extends CriteriaConditionalEvent {
    public MyCriteriaConditionalEvent}() {
        super();
    }

    public MyCriteriaConditionalEvent(EventData data) {
        super(data);
    }

    @Override
    public void init() {
        this.setName("CustomerTxnEvent-BB/01"); // abitrary name
        EventCriteria criteria = new EventCriteria().where("customertype").eq("BB
            ").and("txntype").eq("01");
        this.setCriteria(criteria);
    }

    public void setCustomerType(String customerType) {
        this.getData().put("customertype", customerType);
    }

    public String getCustomerType() {
        return this.getData().get("customertype");
    }

    public void setTxnType(String txnType) {
        this.getData().put("txntype", customerType);
    }

    public String getTxnType() {
        return this.getData().get("txntype");
    }
}
```

Listing 9.12: Conditional Event Example

Note: Only the "and" conjunction and the "eq" operator as supported at present.

Future additions to the conditional logic functionality will allow for increased criteria flexibility.

Delegator Conditional Events

The condition evaluation for Delegator Conditional Events is managed by a delegate class, which is associated with the IConditionalEventDelegate interface. The delegate does the function of returning true or false for the validation of the event.

Note: Typically, use of Delegator conditional events is *recommended* by extending from the abstract class:

`com.sybase365.mobiliser.framework.event.model.DelegatorConditionalEvent`

Expression Conditional Events

A conditional event that is evaluated through a Java Unified Expression Language (JUEL) context and expression. The context does the function of returning true or false for the validation of the event. The expression set to the event must evaluate to either a boolean true or false value.

Note: Typically, use of Expression conditional events is *recommended* by extending from the abstract class:

`com.sybase365.mobiliser.framework.event.model.ExpressionConditionalEvent`

Note: Usage of this event requires setting of a JUEL ExpressionFactory implementation.

Note: The data associated with the event is automatically made available to the expression through the prefix "eventData.".

For example, here is an expression referencing an event data field named "test":

$eventData.test == "A"

Additional variables or Java Beans can be assigned to the evaluation context using the helper class Context:

```
1 ...
2  myExprEvent.getContextHelper().setVariable("myVariable", myExprEvent.getFactory
       ().createValueExpression("value", String.class));
3
4  myExprEvent.setExpression("\${eventData.testValue == myVariable}");
5 ...
```

Listing 9.13: Add Variables to Expression Condition Evaluation Context Example

Methods can also be added to the evaluation context;

```
1 ...
2   myExprEvent.getContextHelper().setFunction("math", "sin", Math.class.getMethod(
       "sin", double.class));
3
4   myExprEvent = setExpression("\${math:sin(2) > 1}");
5 ...
```

Listing 9.14: Add Methods to Expression Condition Evaluation Context Example

9.1.5 Scheduled Events

This class of event has the following features:

- Maintains header and data information, as described in the section above.
- Any associated Delay information is ignored for processing.

- The event will trigger at a fixed date and time in the future.
- Event and event data is persisted in the database.
- Handler processing status is persisted in the database and updated after processing is complete.
- Depending on the handler expiry settings (see below), it is subject to handler catchup processing.
- The event may optionally repeat at a defined interval until a specified end date.

The schedule date and time specification is managed by the EventSchedule class, which is associated with the abstract ScheduledEvent class.

The EventSchedule class allows a easily understood DSL-like syntax for creating schedules. For example:

```
1 ...
2 // Create a triggered once event, to be triggered in one ho
3 hours time
4 EventSchedule scheduleOnce = new EventSchedule().at new Date(System.
      currentTimeMillis())+(1000*60*60));
5
6 // Or, create a repeat event, to trigger in one hour, then repeat every hour, for
       a day
7 EventSchedule scheduleOnce = new EventSchedule().at new Date(System.
      currentTimeMillis())+(1000*60*60)).repeatsEvery(EventSchedule.Repeat.HOUR).
      ends(new Date(System.currentTimeMillis())+(1000*60*60*24);
8 ...
```

Listing 9.15: Scheduled Event Example

Note: Typically, use of scheduled events is recommended by extending from the abstract class:

`com.sybase365.mobiliser.framework.event.model.ScheduledEvent`

This allows for the specification of the event schedule in the definition of event allows for Spring initiated scheduled event generation.

The following class shows an example of extending the ScheduledEvent abstract class:

```
1 ...
2 public class MyScheduledEvent extends ScheduledEvent {
3
4     public MyScheduledEvent() {
5         super();
6     }
7
8     public MyScheduledEvent(EventData data) {
9         super(data);
10    }
11
12    @Override
13    public void init() {
14        this.setName("CustomerTxnEvent-BB/01"); // abitrary name
15        EventCriteria criteria = new EventCriteria().where("customertype").eq("BB
              ").and("txntype").eq("01");
16        this.setCriteria(criteria);
17    }
18
19    public void setCustomerType(String customerType) {
20        this.getData().put("customertype", customerType);
21    }
22
23    public String getCustomerType() {
```

```
24          return this.getData().get("customertype");
25      }
26
27      public void setTxnType(String txnType) {
28          this.getData().put("txntype", customerType);
29      }
30
31      public String getTxnType() {
32          return this.getData().get("txntype");
33      }
34  }
```

Listing 9.16: Conditional Event Implementation Example

Spring Scheduled Events

It is possible to create scheduled events through Spring beans using the class:

`com.sybase365.mobiliser.framework.event.model.SpringScheduledEvent`

The following is an example Spring bean configuration for a scheduled event:

```
1  ...
2  <bean id="hourlyRepeat"
3      class="com.sybase365.mobiliser.framework.event.model.data.EventSchedule.
           Repeat" factory-method="valueOf">
4      <constructor-arg>
5          <value>HOUR</value>
6      </constructor-arg>
7  </bean>
8
9  <bean id="dateFormat" class="java.text.SimpleDateFormat">
10     <constructor-arg value="yyyy-MM-dd HH:mm" />
11 </bean>
12
13 <bean id="HourlySystemHeartbeat"
14     class="com.sybase365.mobiliser.framework.event.model.SpringScheduledEvent"
15     init-method="init"
16     destroy-method="destroy" >
17
18 <!-- Event Name -->
19 <constructor-arg index="0"
20     type="java.lang.String" value="HourlySystemHeartbeatEvent"/>
21
22 <!-- Event Data Map<String,Object> -->
23 <constructor-arg index="1">
24     <util:map>
25         <entry key="emailTo1" value="admin@system"/>
26         <entry key="emailTo2" value="me@system"/>
27     </util:map>
28 </constructor-arg>
29
30 <!-- Scheduled start Date -->
31 <constructor-arg index="2">
32     <bean factory-bean="dateFormat" factory-method="parse">
33         <constructor-arg value="2010-01-01 00:00" />
34     </bean>
35 </constructor-arg>
36
37 <!-- Repeats interval -->
38 <constructor-arg index="3" ref="hourlyRepeat" />
```

```
39
40 <!-- Scheduled end Date -->
41 <constructor-arg index="4">
42     <bean factory-bean="dateFormat" factory-method="parse">
43         <constructor-arg value="-12-31 23:59" />
44     </bean>
45 </constructor-arg>
46 </bean>
47 ...
```

Listing 9.17: Spring Scheduled Event Example Configuration

Note: The Scheduled Event created by the Spring registration is expected to be uniquely identifiable by it's event name, as assigned in the Spring configuration. Any existing event(s) with this name are canceled prior to the new scheduled event being registered.

Cancelling Scheduled Events

Once a scheduled event is created, it is registered with both the scheduling system and the event data store. Cancelling a scheduled event will remove it from the scheduling system and the event data store, so that it will never trigger.

Note: When a ScheduledEvent triggers it creates a RegularEvent with the same original data, in order to process the event. Cancelling a ScheduledEvent will not remove any RegularEvents that have already been created from the ScheduledEvent.

To cancel existing ScheduledEvents, use the following static methods from the EventFactory class:

`com.sybase365.mobiliser.framework.event.model.EventFactory`

```
1 ...
2 /**
3  * Cancel a scheduled event using it's unique scheduled id
4  *
5  * @param scheduledId The unique if for the scheduled event
6  */
7 public static boolean cancelScheduledEvent(final long scheduledId) { ... }
8
9 /**
10  * Cancel scheduled event(s) by event name
11  *
12  * @param eventName The name of the scheduled event (may not be unique)
13  */
14 public static boolean cancelScheduledEvent(final String eventName) { ... }
15 ...
```

Listing 9.18: Cancel Event Example

9.1.6 Transient Events

This class of event has the following features:

- Maintains header, data and optional delay information, as described in the section above.
- Event and event data is **NOT** persisted in the database.
- Handler processing status is **NOT** persisted in the database.
- Events of this type are **NOT** subject to handler catchup processing.

Note: Transient events should be used with caution as they rely on the assumption that there are no event handlers that require regeneration of historical events of this type. The principle of loosely coupled event generation from event handling is maintained, a transient event generator does not know how or by what it is going to be handled. However, it has made the assumption that no event handler will be able to regenerate historical events, through the event handler catchup processing.

To identify a transient event, you may either extend the abstract class:

`com.sybase365.mobiliser.framework.event.model.TransientEvent`

Or, extend any other class of event and override the base method isTransient, for example;

```
1 ...
2 public class MyTransientEvent extends RegularEvent {
3
4     public MyTransientEvent () {
5         super();
6     }
7
8     public MyTransientEvent (EventData data) {
9         super(data);
10    }
11
12    @Override
13    public void init() {
14        this.setName("CustomerTxnEvent");
15    }
16
17    @Override
18    public boolean isTransient() {
19        return true;
20    }
21
22    public void setCustomerType(String customerType) {
23        this.getData().put("customertype", customerType);
24    }
25
26    public String getCustomerType() {
27        return this.getData().get("customertype");
28    }
29
30    public void setTxnType(String txnType) {
31        this.getData().put("txntype", customerType);
32    }
33
34    public String getTxnType() {
35        return this.getData().get("txntype");
36    }
37 }
```

Listing 9.19: Transient Event Implementation Example

9.2 Event Handling API

This section describes the class hierarchy associated with creating event handlers.

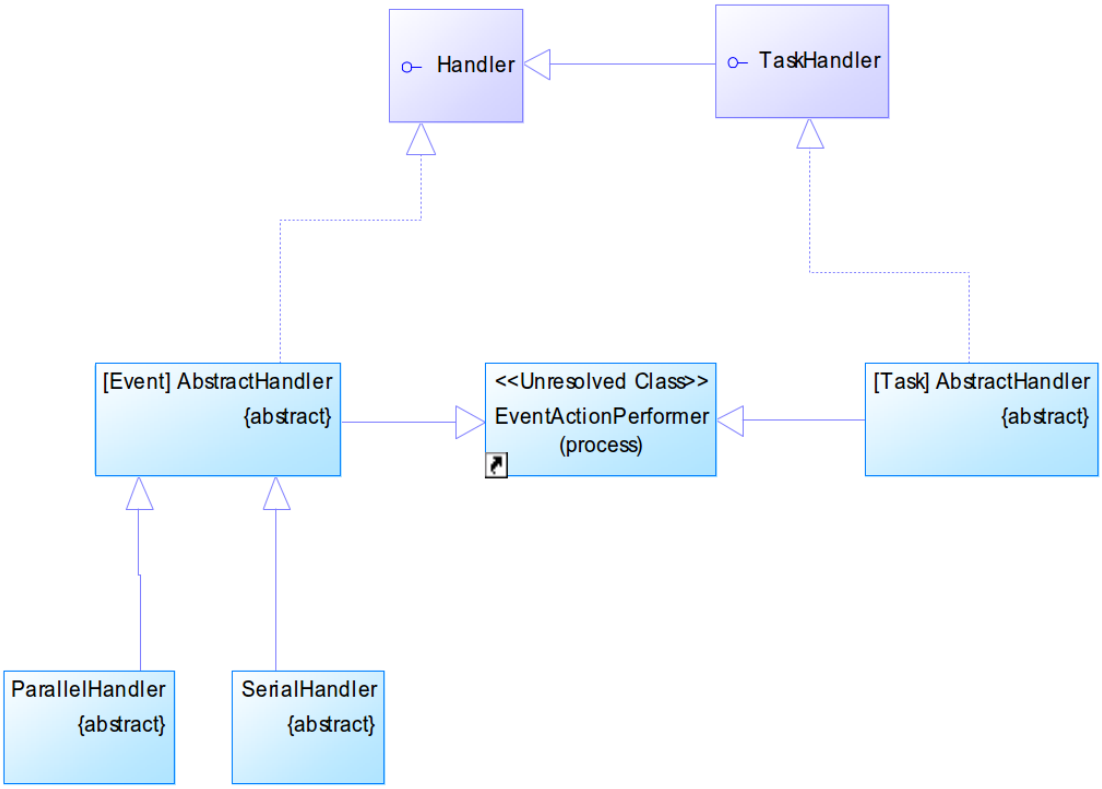

Figure 9.2: Event Handling API Class Diagram

9.2.1 Abstract Event Handler

The abstract handler for events defines the base handler processing requirements. The abstract handler should not be extended directly, instead refer to the next sections describing SerialEventHandler and ParallelEventHandler.

```
1  package com.sybase365.mobiliser.framework.event.model;
2
3  import com.sybase365.mobiliser.framework.event.process.EventActionPerformer;
4
5  /**
6   *
7   * <p>
8   * &copy 2011 Sybase Inc., an SAP Company
9   *
10  * @author msw
11  */
12 public abstract class AbstractHandler extends EventActionPerformer implements
        Handler {
13
14      /**
15       * time in seconds after event creation that this handler will ignore events
16       *  0 means old unprocessed events will be ignored (no catchup mode))
17       * -1 means all unprocessed events will be processed (in catchup mode)
18       * >0 means some unprocessed events will be processed (in catchup mode) - if
             event's
19       *  creation date is greated than now minus this expiry period, then event is
              processed
20       */
21      public int DEFAULT_EXPIRY_PERIOD = 0;
```

```
22
23      /**
24       * Return the expiry time period (in seconds) that this handler uses. Events
             created
25       * past this expiry time will be ignored by this handler.
26       *
27       * @return int Seconds value
28       */
29      @Override
30      public int getExpiryPeriod() {
31      return DEFAULT_EXPIRY_PERIOD;
32      }
33
34      /**
35       * Return a unique name for the event handler - this doesn't have to be the
             same as
36       * the event name.
37       *
38       * @return String value
39       */
40      @Override
41      public abstract String getHandlerName();
42
43      /**
44       * Return the string name of the event that this handler will process.
45       *
46       * @return String value
47       */
48      @Override
49      public abstract String getEventName();
50
51      /**
52       * Process the event.
53       *
54       * @param e
55       * @return boolean true or false if event was processed successfully or not.
56       */
57      @Override
58      public abstract boolean process(Event e);
59
60      /**
61       * Max Active controls the maximum number of objects that can be allocated by
             the pool
62       * (checked out to clients, or idle awaiting checkout) at a given time. When
             non-positive,
63       * there is no limit to the number of objects that can be managed by the pool
             at one time.
64       * When maxActive is reached, the pool is said to be exhausted.
65       *
66       * @return integer value
67       */
68      @Override
69      public abstract int getMaxActive();
70
71      /**
72       * Max Idle controls the maximum number of objects that can sit idle in the
             pool at any
73       * time. When negative, there is no limit to the number of objects that may
             be idle at
74       * one time.
75       *
76       * @return integer value
77       */
78      @Override
79      public abstract int getMaxIdle();
```

```
80 }
```

Listing 9.20: com.sybase365.mobiliser.framework.event.model.AbstractHandler

Note: Overide the getExpiryPeriod() method if your event handler requires to handle un-processed events generated in the past.

Note: The process(Event e) method is the point of interface to the handler. If the event that is being handled is actually a concrete sub-class of one of the abstract Event classes, then this will not be guaranteed to be reflected in the class hierarchy of the process() method parameter.

9.2.2 Serial Event Handler

For your own event handler, extend from the serial event handler when you want to guarantee that only one event handler thread will be processing at any one time.

```
1  package com.sybase365.mobiliser.framework.event.model;
2
3  /**
4   * An event handler that will run only one thread of processing.
5   * <p>
6   * &copy 2011 Sybase Inc., an SAP Company
7   *
8   * @author msw
9   */
10 public abstract class SerialHandler extends AbstractHandler {
11
12     /** Fixed value for serialised handler - 1 active pool thread only */
13     public static int DEFAULT_MAX_ACTIVE = 1;
14
15     /** Fixed value for serialised handler - 1 max idle pool thread only */
16     public static int DEFAULT_MAX_IDLE = 1;
17
18     /**
19      * Max Active controls the maximum number of objects that can be allocated by
            the pool
20      * (checked out to clients, or idle awaiting checkout) at a given time. When
            non-positive,
21      * there is no limit to the number of objects that can be managed by the pool
            at one time.
22      * When maxActive is reached, the pool is said to be exhausted. The default
            setting for
23      * this parameter is 1
24      *
25      * @return integer value
26      */
27     @Override
28     public int getMaxActive() {
29     return DEFAULT_MAX_ACTIVE;
30     }
31
32     /**
33      * Max Idle controls the maximum number of objects that can sit idle in the
            pool at any
34      * time. When negative, there is no limit to the number of objects that may
            be idle at
35      * one time. The default setting for this parameter is 1.
36      *
37      * @return integer value
38      */
39     @Override
40     public int getMaxIdle() {
```

```
41      return DEFAULT_MAX_IDLE;
42      }
43  }
```

Listing 9.21: com.sybase365.mobiliser.framework.event.model.SerialHandler

Note: Serial event handling does NOT guarantee *sequential* event handling. Events are not guaranteed to be processed in the same order in which they were raised.

9.2.3 Parallel Event Handler

For your own event handler, extend from the parallel event handler when you to use a sizeable thread pool for multiple instances of event handler processing.

```
1  package com.sybase365.mobiliser.framework.event.model;
2
3  /**
4   * An event handler that can run handle multiple threads of processing.
5   * <p>
6   * &copy 2011 Sybase Inc., an SAP Company
7   *
8   * @author msw
9   */
10 public abstract class ParallelHandler extends AbstractHandler {
11
12      /** Default value for getMaxActive(): 4 */
13      public static int DEFAULT_MAX_ACTIVE = 4;
14
15      protected int maxActive = DEFAULT_MAX_ACTIVE;
16
17      /** Default value for getMaxIdle(): 2 */
18      public static int DEFAULT_MAX_IDLE = 2;
19
20      protected int maxIdle = DEFAULT_MAX_IDLE;
21
22      /**
23       * Max Active controls the maximum number of objects that can be allocated by
             the pool
24       * (checked out to clients, or idle awaiting checkout) at a given time. When
             non-positive,
25       * there is no limit to the number of objects that can be managed by the pool
             at one time.
26       * When maxActive is reached, the pool is said to be exhausted. The default
             setting for
27       * this parameter is 4
28       *
29       * @return integer value
30       */
31      @Override
32      public int getMaxActive() {
33      return this.maxActive;
34      }
35
36      public void setMaxActive(int value) {
37      this.maxActive = value;
38      }
39
40      /**
41       * Max Idle controls the maximum number of objects that can sit idle in the
             pool at any
42       * time. When negative, there is no limit to the number of objects that may
             be idle at
```

```
43       * one time. The default setting for this parameter is 2.
44       *
45       * @return integer value
46       */
47      @Override
48      public int getMaxIdle() {
49      return this.maxIdle;
50      }
51
52      public void setMaxIdle(int value) {
53      this.maxIdle = value;
54      }
55  }
```

Listing 9.22: com.sybase365.mobiliser.framework.event.model.ParallelHandler

Note: Parallel event handling default thread pool settings may be overridden using Spring bean properties 'MaxActive' and 'MaxIdle'.

9.2.4 Event Handler Registration

To register an event handler into the AIMS/OSGi event system environment requires:

1. Creation of an event handler processing class that extends from one of and implements all abstract methods.

   `com.sybase365.mobiliser.framework.event.model.SerialHandler`

   `com.sybase365.mobiliser.framework.event.model.ParallelHandler`

2. Creation of a Spring bean defining the event handler bean.
3. Creation of an Spring OSGi service referencing the bean using the interface

`com.sybase365.mobiliser.framework.event.model.Handler`

The following is an example Spring bean configuration for a parallel event handler, with non-standard thread pool sizes;

```
1  ...
2  <!--
3  ***********************************
4  Beans Configuration
5  ***********************************
6  -->
7  <bean id="Type0EventHandler"
8        class="com.sybase365.mobiliser.framework.event.test.Type0EventHandler">
9      <property name="maxActive" value="2" />
10     <property name="maxIdle" value="0" />
11 </bean>
12 <!--
13
14 ***********************************
15 OSGi Services Configuration
16 ***********************************
17 -->
18 <osgi:service id="Type0EventHandlerService"
19       ref="Type0EventHandler"
20       interface="com.sybase365.mobiliser.framework.event.model.Handler" />
21 ...
```

Listing 9.23: OSGI Configuration For Event Handler

9.3 Task Handling API

This section describes the class hierarchy associated with creating task handlers.

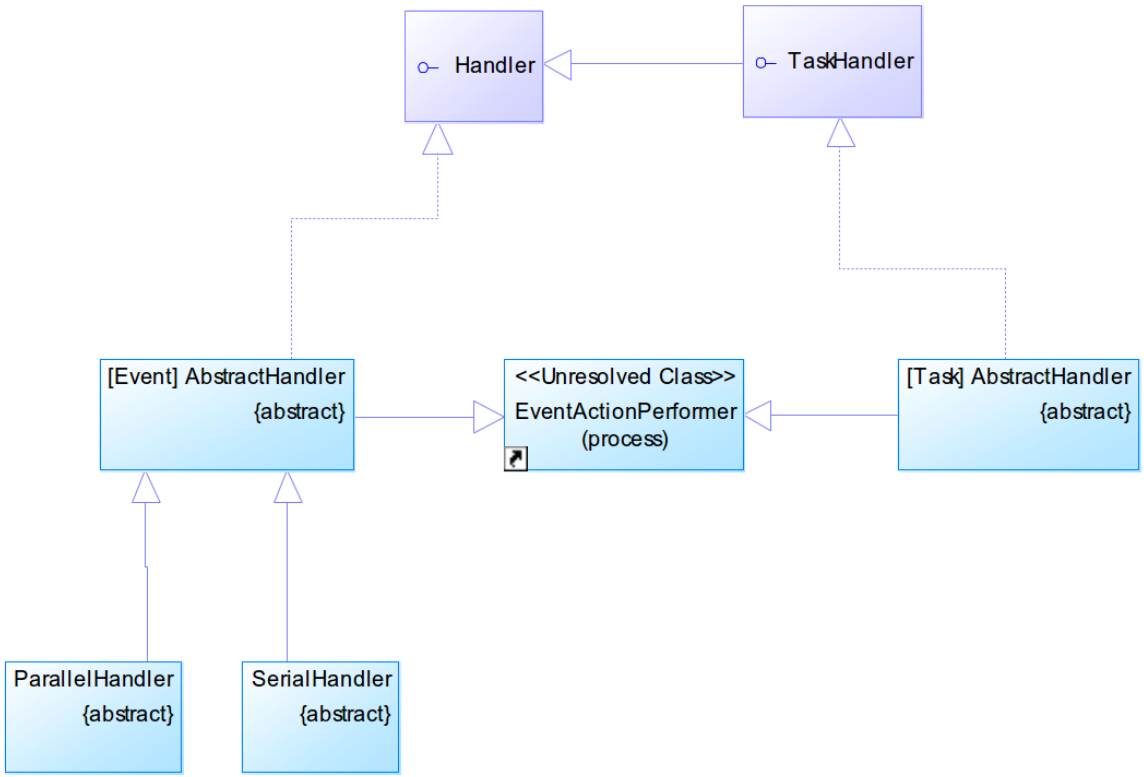

Figure 9.3: The Task Handling API Class Diagram

9.3.1 Abstract Task Handler

The abstract handler for tasks defines the base handler processing requirements.

```
1  package com.sybase365.mobiliser.framework.task.model;
2
3  import com.sybase365.mobiliser.framework.event.model.Event;
4  import com.sybase365.mobiliser.framework.event.process.EventActionPerformer;
5
6  /**
7   *
8   * <p>
9   * &copy 2011 Sybase Inc., an SAP Company
10  *
11  * @author msw
12  */
13 public abstract class AbstractHandler extends EventActionPerformer implements
        TaskHandler {
14
15      /**
16       * time in seconds after event creation that this handler will ignore events
17       *  0 means old unprocessed events will be ignored (no catchup mode))
18       * -1 means all unprocessed events will be processed (in catchup mode)
19       * >0 means some unprocessed events will be processed (in catchup mode) - if
             event's
```

```
20       *  creation date is greated than now minus this expiry period, then event is
             processed
21       */
22      public int DEFAULT_EXPIRY_PERIOD = 0;
23
24      /**
25       * Return the expiry time period (in seconds) that this handler uses. This is
             fixed
26       * for task handlers as they have no need to process prior events for task
             triggers.
27       *
28       * @return int Seconds value
29       */
30      @Override
31      public final int getExpiryPeriod() {
32      return DEFAULT_EXPIRY_PERIOD;
33      }
34
35      /**
36       * Return a unique name for the event handler - this doesn't have to be the
             same as
37       * the event name.
38       *
39       * @return String value
40       */
41      @Override
42      public abstract String getHandlerName();
43
44      /**
45       * Return the string name of the event that this handler will process.
46       *
47       * @return String value
48       */
49      @Override
50      public final String getEventName() {
51      return this.getTaskName();
52      }
53
54      /**
55       * Process the event.
56       *
57       * @param e
58       * @return boolean true or false if event was processed successfully or not.
59       */
60      @Override
61      public abstract boolean process(Event e);
62      /** Default value for task handler - 1 active pool thread only */
63      public static int DEFAULT_MAX_ACTIVE = 1;
64      protected int maxActive = DEFAULT_MAX_ACTIVE;
65      /** Default value for task handler - 1 max idle pool thread only */
66      public static int DEFAULT_MAX_IDLE = 1;
67      protected int maxIdle = DEFAULT_MAX_IDLE;
68
69      /**
70       * Max Active controls the maximum number of objects that can be allocated by
             the pool
71       * (checked out to clients, or idle awaiting checkout) at a given time. When
             non-positive,
72       * there is no limit to the number of objects that can be managed by the pool
             at one time.
73       * When maxActive is reached, the pool is said to be exhausted.
74       *
75       * @return integer value
76       */
77      @Override
```

```
78      public int getMaxActive() {
79      return this.maxIdle;
80      }
81
82      public void setMaxActive(int value) {
83      this.maxActive = value;
84      }
85
86      /**
87       * Max Idle controls the maximum number of objects that can sit idle in the
            pool at any
88       * time. When negative, there is no limit to the number of objects that may
            be idle at
89       * one time.
90       *
91       * @return integer value
92       */
93      @Override
94      public int getMaxIdle() {
95      return this.maxIdle;
96      }
97
98      public void setMaxIdle(int value) {
99      this.maxIdle = value;
100     }
101
102     /**
103      *
104      * @return String value
105      */
106     @Override
107     public abstract String getTaskName();
108
109     /**
110      *
111      * @return String value
112      */
113     @Override
114     public abstract String getCronExpr();
115
116     /**
117      *
118      * @return String value
119      */
120     @Override
121     public String getTimeZone() {
122     return null;
123     }
124 }
```

Listing 9.24: OSGI Configuration For Event Handler

Note: The implemented getExpiryPeriod() method [of the Handler interface] is declared as final because task handlers do not process historical events.

Note: The implemented getEventName() method[of the Handler interface] is declared as final and directly maps the event name to the getTaskName() method [of the TaskHandler interface].

Note: Thread pool settings are available to the task handler, in the same way as event handlers. Normally, and the default setting, is to define a single thread of execution for the task handler. In this case, if a task handler is executing based on a scheduled trigger and another trigger of the task occurs (because the task handler is long-running or if the trigger schedule is frequent), then the 2^{nd} action will be delayed until the task handler completes because the single thread pool for this task is already in use. If the task thread pool settings are increased the 2^{nd} action may continue with a new thread for this task handler.

9.3.2 Task Handler Registration

To register a task handler into the AIMS/OSGi event system environment requires;

1. Creation of a task handler processing class that extends from the base task abstract class and implements all abstract methods:
   `com.sybase365.mobiliser.framework.task.model.TaskHandler`
2. Creation of a Spring bean defining the task handler bean.
3. Creation of an Spring OSGi service referencing the bean using the interface
   com.sybase365.mobiliser.framework.event.model.TaskHandler

The following is an example Spring bean configuration for a task handler:

```
1  ...
2  <!--
3  ***********************************
4  Beans Configuration
5  ***********************************
6  -->
7  <bean id="SimpleTaskHandler"
8        class="com.sybase365.mobiliser.framework.event.test.SimpleTaskHandler">
9  </bean>
10 <!--
11
12 ***********************************
13 OSGi Services Configuration
14 ***********************************
15 -->
16 <osgi:service id="SimpleTaskHandlerService"
17       ref="SimpleTaskHandler"
18       interface="com.sybase365.mobiliser.framework.task.model.TaskHandler" />
19 ...
```

Listing 9.25: OSGI Configuration For Task Handler

9.4 System Bundles and Services

This section described the services configuration of the event system. The event system is comprised of;

Spring Beans These beans are wired automatically by the event-core bundle and are not described further:

- EventDispatcher
  c.s.m.framework.event.process.Dispatch
- EventGenerator
  c.s.m.framework.event.process.Generator
- EventListenerRegistrar
  c.s.m.framework.event.registration.ListenerRegistration
- EventManagementPlugin
  c.s.m.framework.event.managementconsole.EventManagementPlugin

OSGi Configuration Admin Properties

OSGi Services

9.4.1 OSGi Configuration Admin Properties

The following settings affect the processing of the event core and can be managed through the event-core configuration admin properties file
`service.event.core.properties`:

```
1  ################################################################################
2  #
3  # AIMS - Mobiliser Event Processing Configuration
4  #
5  ################################################################################
6  # number of events to be retrieved during each regeneration select
7  # during handler catchup processing
8  regeneration.batch.size=10
9  # virtual capacity of the delayq (events that will be processed after
10 # a short delay)  if capacity reached a warning message is output in the logs
11 delayedq.capacity=1000
12 # virtual capacity of the processq (events that will be processed asap)
13 # if capacity reached a warning message is output in the logs
14 processq.capacity=1000
15 # virtual capacity of the catchupq (events that will be processed by handlers
16 # in catchup mode) if capacity reached a warning message is output in the logs
17 reached
18 catchupq.capacity=1000
```

Listing 9.26: service.event.core.properties

9.4.2 OSGi Service

The following diagram outlines the dependencies and links between bundles providing services to the event core. Each service is described in a separate section below.

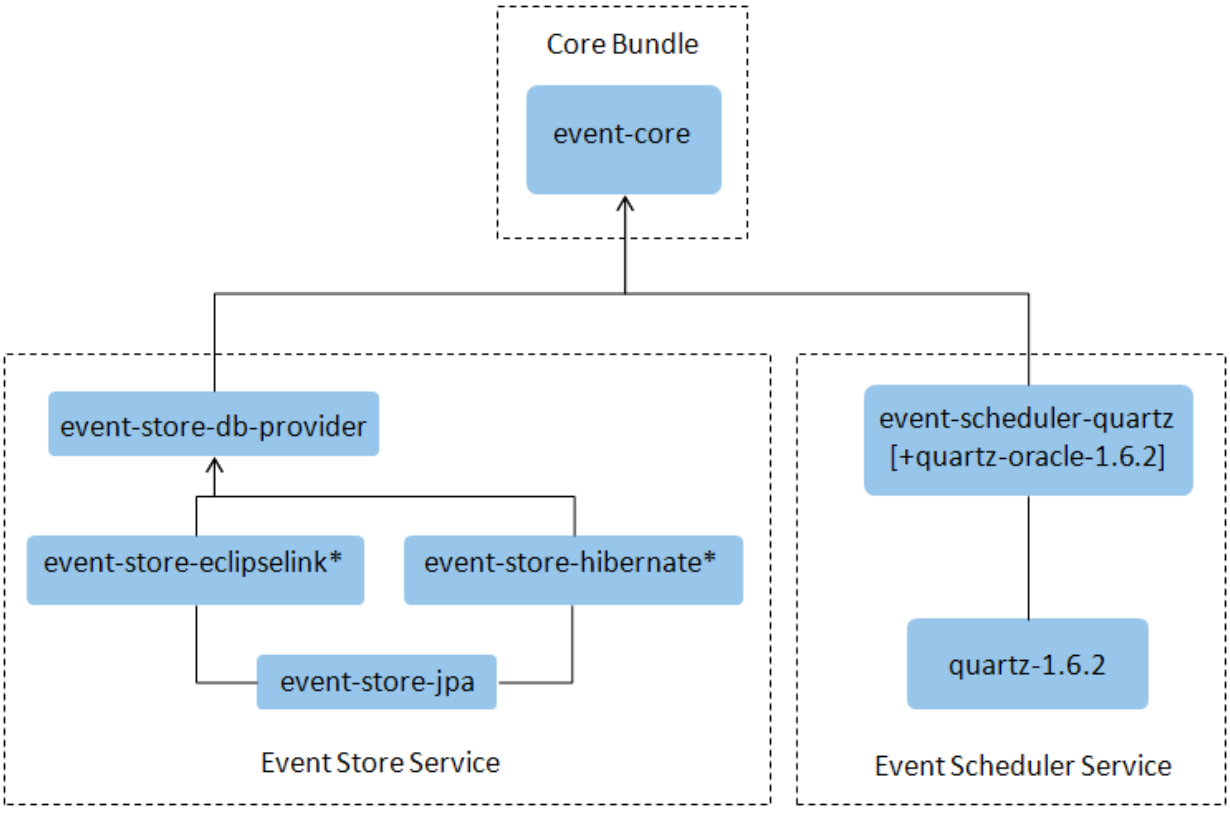

Figure 9.4: Event System Service Bundles

Event Store Service

The event store service provides event persistence, retrieval and modification through the service interface:

`com.sybase365.mobiliser.framework.event.persistence.StoreProvider`

The bundles that comprise this service are:

event-store-db-provider This bundle provides the store logic to map event objects from the core, to event entity objects known to the Java Persistence API (JPA) implementation used. It also defines the required operations for the Data Access Objects (DAOs) of the JPA objects. Where other bundles require a DataSource object for database access, it also defines a wrapper object for access to a DataSource implementation.

event-store-jpa This bundle describes the database JPA entities that are used to persist the event database data back to the database. This bundle is independent of the JPA implementation used. It also defines the standard NamedQuery's that are used to access the entities.

event-store-eclipselink This bundle is used as the JPA implementation to provide access to the Brand Mobiliser 1.2 database. It implements the set of DAO interfaces that provide access to the data from the Brand Mobiliser JPA mechanism.

event-store-hibernate This bundle is used as the JPA implementation to provide access to the Money Mobiliser database. It implements the set of DAO interfaces that provide access to the data from the Money Mobiliser JPA mechanism.

Note: Only one of event-store-eclipselink and event-store-hibernate is required depending on the deployment. If other application databases are required to be interfaced to, it is possible to write a newevent-store-xyz physical database provider.

Scheduler Service

The scheduler service provides event scheduled triggering of events and tasks for the event system through triggering the service interface:

`com.sybase365.mobiliser.framework.event.scheduler.SchedulerProvider`

The bundles that comprise this service are:

event-schedule-quartz This bundle generates and starts an instance of the scheduler provider of service (Quartz) and providers the service implementation which takes events and task configurations and translates those event schedules into Quartz Jobs and Triggers.

Note: The Quartz scheduler is created through Spring config configuration using the class:

`org.springframework.scheduling.quartz.SchedulerFactoryBean`

Certain properties of the Quartz configuration are exposed as Configuration Admin settings under the persistent-0id service.event.quartz.

quartz-1.6.2 This is the bundle providing the Quartz scheduler. This version of the scheduler is used because it is the version that the Spring scheduler capability integrates.

Quartz Scheduler Configuration In most instances it will not be necessary to alter the Quartz scheduler properties.

```
#########################################################
#
# AIMS - Mobiliser Event Processing Configuration
#
# Quartz scheduler properties
#
#########################################################

# Main settings
scheduler.instanceName=MobiliserEventScheduler
scheduler.instanceId=AUTO

# Threadpool settings
threadPool.class=org.quartz.simpl.SimpleThreadPool
threadPool.threadCount=5
threadPool.threadPriority=5

# JobStore settings
jobStore.tablePrefix = QTZ_
jobStore.misfireThreshold=60000
jobStore.class=org.quartz.impl.jdbcjobstore.JobStoreTX
jobStore.driverDelegateClass=org.quartz.impl.jdbcjobstore.oracle.OracleDelegate
jobStore.useProperties=false
jobStore.selectWithLockSQL=SELECT * FROM {0}LOCKS UPDLOCK WHERE LOCK_NAME = ?

# Logging settings
triggHistory.class=org.quartz.plugins.history.LoggingTriggerHistoryPlugin
triggHistory.triggerFiredMessage=Trigger {1}.{0} fired job {6}.{5} at: {4, date,
    HH:mm:ss
MM/dd/yyyy}
triggHistory.triggerCompleteMessage=Trigger {1}.{0} completed firing job {6}.{5}
    at {4,
date, HH:mm:ss MM/dd/yyyy} with resulting trigger instruction code: {9}
```

Listing 9.27: service.event.core.properties

9.5 Web Console Mobiliser Events Plugin

The event system can be monitored and managed through the AIMS System Web Console using the Mobiliser Events Plugin that is provided in the event-core bundle.

The Mobiliser Events Plugin extends the AIMS System Web Console by providing a tab to summarise the processing status of the events system.

Using a Web Browser, connect to the address;

`http://localhost:8080/system/console`

When prompted for the username and password use the values supplied by technical support. Then, under the 'Mobiliser Events' tab will be a list of all configurations, similar to that shown in the diagram below.

The information available is described in the next sections.

9.5.1 Event Summary

The total number of events that have been generated (of all types) and processed (by all types) by the known number of handlers active at that instance in time, since this server instance startup.

Figure 9.5: Web Console Mobiliser Event Plugin

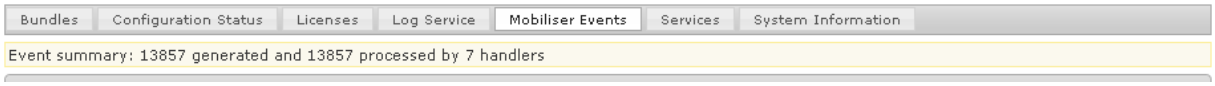

Figure 9.6: Event Summary

9.5.2 Overview/Generator

Figure 9.7: Overview/Generator

The total number of events of different event classes that have passed through the generator, since this server instance startup.

9.5.3 Queues

Shows a list of known physical and virtual queues and an instantaneous count of events for the queues.

Physical Queues There will be a list of queues for the Physical Queues; ProcessQ, DelayedQ and CatchupQ. This shows the number of events that are present in that physical queue at that instance in time.

Virtual Queues A virtual queue is an indication of how many events are in the physical queues for each event name at that instance in time. (One queue per event name.) If no virtual queues are shown then no

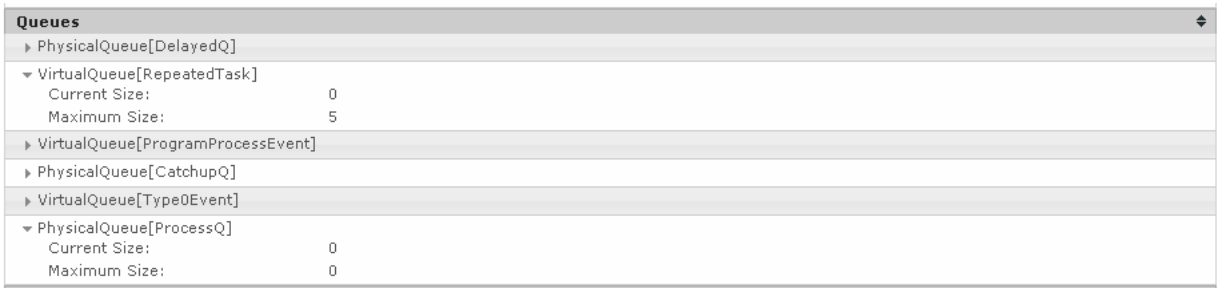

Figure 9.8: Queues

events have been created.

Note: Only under heavy loading will you likely see any significant event counts in these queues.

9.5.4 Scheduled Events

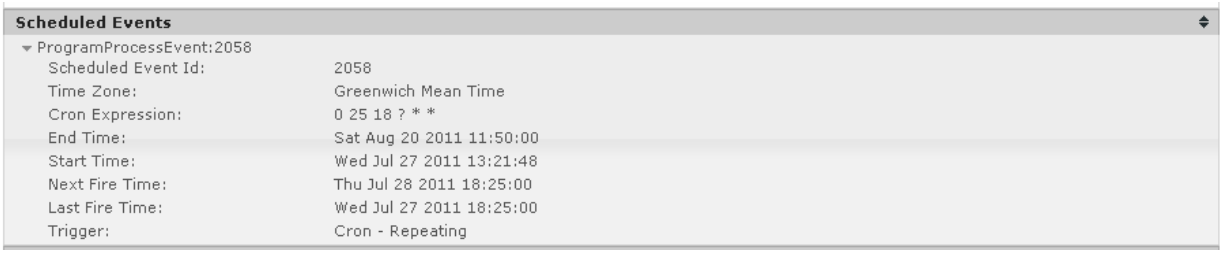

Figure 9.9: Scheduled Events

This list shows the internal scheduler system view of all events that are scheduled for triggering. If the list is empty, there are no known scheduled events.

Entries in the list will either be marked:

Trigger: Simple - One-off For an event that will be triggered once and once only. For this type of scheduled event, the detal will show:

Scheduled Event Id: The internal id (database entity PK) of the scheduled event.

Fire Time: The expected to,e of the once-only trigger.

Trigger: Cron – Repeating For an event that has a repeating interval set. For this type of scheduled event, the detail will show:

Scheduled Event Id: The internal id (database entity PK) of the scheduled event.

Start Time: The time the first trigger fired.

End Time: The time beyond which no more triggers will fire.

Next Fire Time: The expected time of the next trigger fire.

Last Fire Time: The last time the trigger fired.

Cron Expression: An expression conforming to Unix cron standards for specifying repeats.

9.5.5 Event Handlers

Shows a list of known event handlers registered with the event system ListenerRegistrar. For each event handler, the following information is shown:

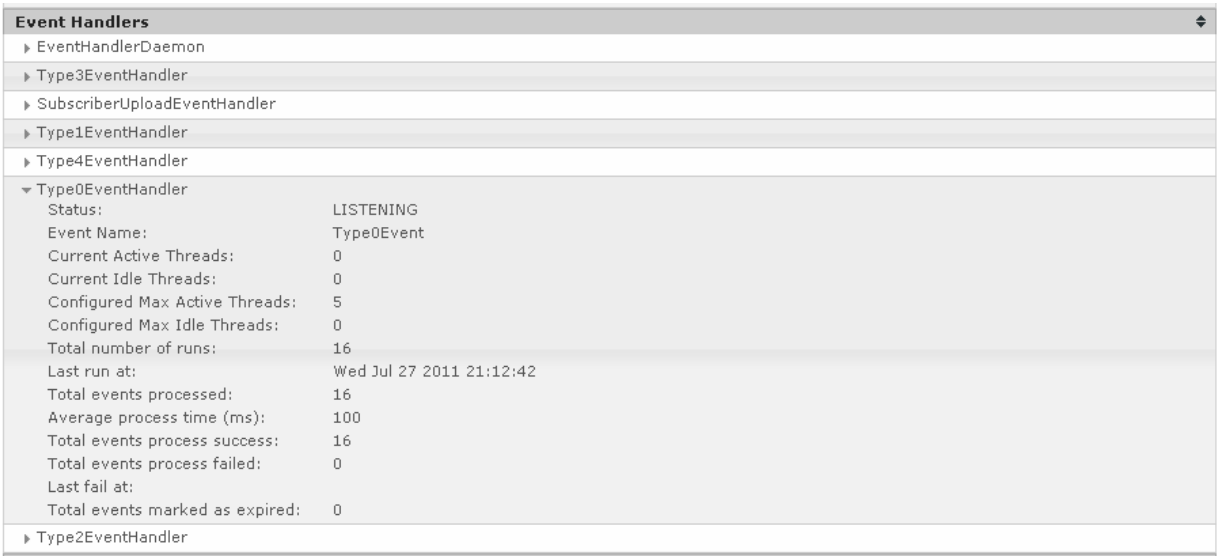

Figure 9.10: Event Handlers

Status: Either CATCHUP if listener is still processing regenerated events, or LISTENING if active.

Event: The event name against which the handler is registered.

Current Active Threads: Threads running at this point in time.

Current Idle Threads: Threads allocated to this handlers pool, but not active at this point in time.

Configured Max Active Threads: Maximum size of event handler thread pool.

Configured Max Idle Threads: Maximum number of idle, but not active threads in the pool.

Total number of runs: Number of times the event handler was invoked – maybe different from 'Total events processed' because a handler run does not cause an event to be processed if the handler cannot get a processing status lock on the event or if the event is expired.

Last run at: Date and time of last run.

Total events processed: Number of times the handler process method was called.

Average process time (ms): The average amount of time spent in the handler's process method.

Total events process success: The number of events that returned 'true' from its handler process.

Total events process failed: The number of events that returned 'false' or through an exception from the handler process.

Last fail at: Date and time of last indicated failed processing event.

Total events marked as expired: The number of events whose expire time has been reached before processing. **Note:** doesn't include events not regenerated due to expiry .

9.5.6 Tasks

This list shows the internal scheduler system view of all tasks that are scheduled for triggering. If the list is empty, there are no known scheduled tasks.

Entries in the list will be marked: Trigger: Cron – Repeating – For an task that has a repeating interval set.

Task Name: The unique task name.

Time Zone: An optional time zone in which the cron expression should be evaluated.

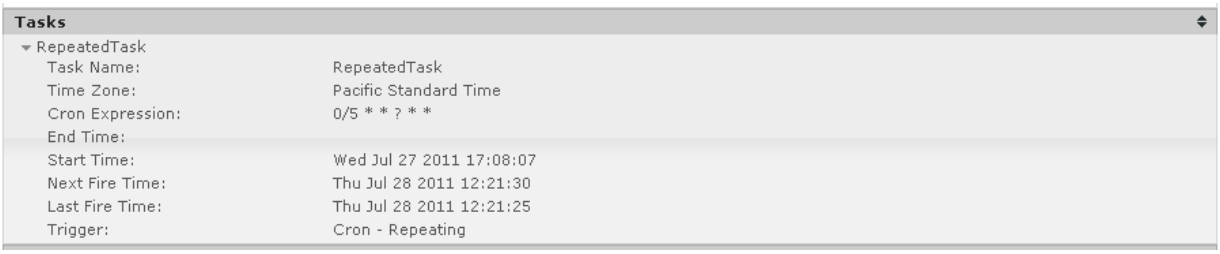

Figure 9.11: Tasks

Start Time: The time the first trigger fired.

End Time: The time beyond which no more triggers will fire or empty if never set.

Next Fire Time: The expected time of the next trigger fire.

Last Fire Time: The last time the trigger fired.

Cron Expression: An expression conforming to Unix cron standards for specifying repeats.

9.5.7 Task Handler

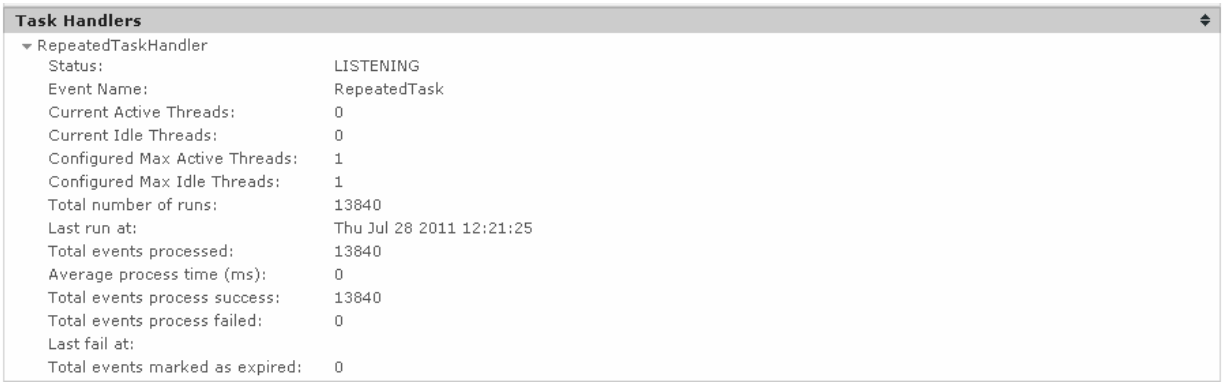

Figure 9.12: Tasks Handler

Shows a list of known task handlers registered with the event system ListenerRegistrar. For each task handler, the following information is shown:

Status: LISTENING,if active.

Event Name: The task name against which the handler is registered.

Current Active Threads: Threads running at this point in time.

Current Idle Threads: Threads allocated to this handlers pool, but not active at this point in time.

Configured Max Active Threads: Maximum size of handler thread pool.

Configured Max Idle Threads: Maximum number of idle, but not active threads in the pool.

Total number of runs: Number of times the task handler was invoked – maybe different from 'Total events processed' because a handler run does not cause an event to be processed if the handler cannot get a processing status lock on the event or if the event is expired.

Last run at: Date and time of last run.

Total events processed: Number of times the handler process method was called.

Average process time (ms): The average amount of time spent in the handler's process method.

Total events process success: The number of events that returned 'true' from its handler process.

Total events process failed: The number of events that returned 'false' or through an exception from the handler process.

Last fail at: Date and time of last indicated failed processing event.

Total events marked as expired: The number of events whose expire time has been reached before processing. **Note:** doesn't include events not regenerated due to expiry .

9.6 Alerts

Alerts are specialised versions of events that can result in a customer receiving a notification of the alert delivered through the messaging system.

Alerts are implemented by using the conditional class of events to evaluate the condition under which an alert is to be raised and subsequent notification passed to the customer.

An alert is not raised for notification unless;

- An action or event has occurred that may result in an alert - the affected area of code needs to
- The customer affected by the action or event has subscribed to receive the alert.
- The customer subscribed to the alert with the specific conditions.

Some alerts will have no specific conditions associated with them, such as a Password Changed Alert, while others may have specific conditions, such as Account Thresholds alert.

By using the events system to generate, process and handle alerts and alert notifications, the principle of loose coupling between the action that caused the alert and any subsequent notification of alert is maintained for alert notification. Similarly, loose coupling is maintained between the point of alert code that initiates the alert and the customer configuration. Also, because alerts are not generated into the events system unless all the conditional criteria specific to the customer for the alert are met, no unecessary events traffic is generated for alerts.

9.6.1 Alert API

All alerts sub-class the base alert event class, AlertEvent, which implements the IAlert interface.

For example;

```
1  ...
2  /**
3   * A regular alert representing the most basic processing object
4   * for an alert. The basic alert extends the base expression conditional event
5   * to allow an evaluation expression for alert creation to be associated with
6   * it.
7   * <p>
8   * The expression is based on event data information and extra information
9   * populated from the associated customer data values.  The expression must
10  * evaluate to true or false.
11  * <p>
12  * &copy 2012 Sybase Inc., an SAP Company
13  * <p>
14  * @see com.sybase365.mobiliser.framework.event.model.ExpressionConditionalEvent
15  *
16  * @author msw
17  */
```

```
18 public class AlertEvent extends ExpressionConditionalEvent implements IAlert {
19
20     private static final Logger LOG = LoggerFactory.getLogger(AlertEvent.class);
21
22     public static final String CUSTOMER_ALERT_ID = "customerAlertId";
23
24     /**
25      *
26      */
27     public AlertEvent() {
28         super();
29     }
30
31     /**
32      *
33      * @param data
34      */
35     public AlertEvent(EventData data) {
36         super(data);
37     }
38
39     /**
40      *
41      * @param name
42      * @param data
43      */
44     public AlertEvent(String name, EventData data) {
45         super(name, data);
46     }
47
48     /**
49      *
50      */
51     @Override
52     public void init() {
53
54     }
55
56     /**
57      * Always treat alert events as transient, thus not to be stored in the
58      * event system
59      *
60      * @return true, unless overridden in subclass
61      */
62     @Override
63     public boolean isTransient() {
64         return Boolean.TRUE;
65     }
66
67     /**
68      * Get the associated customer alert object pk for this alert event
69      *
70      * @return long id
71      */
72     public long getCustomerAlertId() {
73         if (getData().get(CUSTOMER_ALERT_ID) != null) {
74             return Long.valueOf(getData().get(CUSTOMER_ALERT_ID));
75         }
76         else {
77             return -1L;
78         }
79     }
80
81     /**
82      * Set the associated customer alert object pk for this alert event
```

```
83       *
84       * @param value long id
85       */
86      public void setCustomerAlertId(long value) {
87          addData(CUSTOMER_ALERT_ID, String.valueOf(value));
88      }
89  }
```

Listing 9.28: Base AlertEvent class

To assign an expression to an alert, you can use the setExpression() method from within the init() method.

For example;

```
1  ...
2  /**
3   * <p>
4   * The <code>AccountBalanceSummary</code> notifies a customer with the
5   * end of day balance for a specified account.
6   * </p>
7   *
8   * <p>
9   * &copy; 2012 Sybase, Inc.
10  * </p>
11  *
12  * @author <a href='mailto:msw@sybase.com'>Mark White</a>
13  */
14 public class AccountBalanceSummary extends AlertEvent {
15
16      private static final long serialVersionUID = 1L;
17
18      public static final String ACCOUNT_PI_ID = "accountPiId";
19      public static final String ACCOUNT_PI_ALIAS = "accountPiAlias";
20      public static final String ACCOUNT_PI_BALANCE = "accountPiBalance";
21      public static final String ACCOUNT_PI_BALANCE_DISPLAY
22          = "accountPiBalanceDisplay";
23      public static final String ACCOUNT_PI_CURRENCY = "accountPiCurrency";
24      public static final String CUSTOMER_PI_ID = "customerPiId";
25      public static final String CUSTOMER_COUNTRY = "customerCountry";
26      public static final String CUSTOMER_LANGUAGE = "customerLanguage";
27      public static final String CUSTOMER_ORGUNIT = "customerOrgUnit";
28
29      /** note: must match to database table ALR_TYPE.STR_NAME */
30      public static final String EVENT_TYPE = "AccountBalanceSummaryAlert";
31
32      public AccountBalanceSummary() {
33      super();
34      }
35
36      public AccountBalanceSummary(EventData data) {
37          super(data);
38      }
39
40      public AccountBalanceSummary(String name, EventData data) {
41          super(name, data);
42
43      }
44
45      public void setAccountPiId(long value) {
46          addData(ACCOUNT_PI_ID, String.valueOf(value));
47      }
48
49      public long getAccountPiId() {
50          return Long.valueOf(this.getData().get(ACCOUNT_PI_ID));
51      }
```

```
52
53 ...
54
55     public void setAccountPiCurrency(String value) {
56         addData(ACCOUNT_PI_CURRENCY, value);
57     }
58
59     public String getAccountPiCurrency() {
60         return this.getData().get(ACCOUNT_PI_CURRENCY);
61     }
62
63     @Override
64     public void init() {
65         this.setName(EVENT_TYPE);
66
67         if (this.getData() == null) {
68             this.setData(new EventData());
69         }
70
71         StringBuilder expression = new StringBuilder();
72         expression.append("\${ ");
73         expression.append("eventData.").append(CUSTOMER_PI_ID)
74             .append(" == ").append("-1");
75         expression.append(" || ");
76         expression.append("eventData.").append(ACCOUNT_PI_ID)
77             .append(" == ").append("eventData.").append(CUSTOMER_PI_ID);
78         expression.append(" }");
79
80         this.setExpression(expression.toString());
81     }
82 }
```

Listing 9.29: Example Alert with expression

In the example above, the expression to evaluate is;

```
${ eventData.customerPiId == -1 || eventData.accountPiId == eventData.customerPiId }
```

As described in the section above, Expression Conditional Events, the prefix "eventData." allows the expression to reference fields from the instance of the event. The two values to be evaluated are "eventData.accountPiId" and "eventData.customerPiId":

eventData.customerPiId - This value is provided by the customer subscription to this alert and stored in the Mobiliser database under

```
ALR_CUSTOMER_DATA
```

Note: All data stored against an alert set by the customer is automatically added into the alert event data, by the Alert Generator. This allows it to be used in the conditional expression. In general for all fields, the UI stores a value of "-1" to indicate when any field is not set to any specific value.

eventData.accountPiId - This value is set by the code that is attempting to generate the alert.

9.7 Retry Mechanism

The new event system of Mobiliser does not provide a native retry mechanism. If you need such a mechanism in your event handler, you just have to extend the MoneyEventHandler which is defined in the money/jobs/event/handler/util bundle. The MoneyEventHandler extends the ParallelHandler and provides a mechanism for retry event handling.

To create a retry event you must call the createRetryEvent method and pass the event that should be retried as first parameter value. The second parameter indicates whether a new transaction should be used to create the retry event. This method call will create a new event of the same type and with same data as the current event. The new event is marked, so that it will only be processed by an instance of the current event handler type. If the new (retry) event was accepted and processed by the event handler, the method returns true. If something went wrong or the maximum number of retries has been reached, the methods returns false.

9.8 Configuration

Each money event handler expects a configuration object that must be a subclass of MoneyEventHandlerConfiguration. This bean required a preferences node injected and must be used to configure an event handler. This configuration objects also implements IInternalServiceCallAuthenticationData and can hence also be used to configure internal service calls. The concrete instance of the provided MoneyEventHandlerConfiguration is a generic type argument of MoneyEventHandler, the configuration object is available via the getConfiguration() method. In case you need more configuration for a particular event handler, simply extend the MoneyEventHandlerConfiguration and add further properties to it. The preferences node is accessible as the protected preferences property.

By default, the maximum number of retries as well as the delay for the retry events can be configured via preferences. The keys for the corresponding preference entries are:

Retry delay: "event.retry.delay"

Maximum number of retries: "event.retry.maxRetries" (**Note:** The maximum number of tries is: 1 + maximum number of retries)

This behaviour can be changed by overriding the methods getRetryDelay() and/or getMaxRetries() in the configuration object.

Note: When you subclass MoneyEventHandler make sure that you inject all instances required by your implementation. If you use the createRetryEvent method you have to inject a MoneyEventHandlerConfiguration object into field configuration. As soon as you use the createRetryEvent method, you have to inject the event generator EventGenerator into field eventGenerator. If you call the createRetryEvent method and pass true as second parameter – which requires a new transaction for event creation, – you have to inject a transaction manager PlatformTransactionManager into field transactionManager. In case you use one of the fields (from above) in the subclass, you must, of course, inject them as well.

To wire it all up, you'll need preferences, transaction manager, event generator, and the internal service call authenticator

```
1  <!-- OSGI Service Imports used by the Event Handler -->
2  <osgi:reference id="internalAuthenticatorFactory" filter="(instance=default)">
3    <osgi:interfaces>
4      <value>com.sybase365.mobiliser.security.
           InternalServiceCallAuthenticatorFactory</value>
5    </osgi:interfaces>
6  </osgi:reference>
7
8  <osgi:reference id="transactionManager" interface="org.springframework.
       transaction.PlatformTransactionManager" context-class-loader="service-
       provider" />
9
10 <osgi:reference id="daoFactory" interface="com.sybase365.mobiliser.money.
       persistence.dao.factory.api.DaoFactory" />
11 <osgi:reference id="preferencesService" interface="com.sybase365.mobiliser.util.
       prefs.api.IPreferencesService" />
12
13 <osgi:reference id="messagingService" interface="com.sybase365.mobiliser.util.
       messaging.service.api.IMessagingService" />
14
```

```
15 <osgi:reference id="eventGenerator" interface="com.sybase365.mobiliser.framework.
       event.generator.EventGenerator" />
16
17 <osgi:reference id="encryptionStrategy" interface="com.sybase365.mobiliser.util.
       prefs.encryption.IPreferencesEncryptionStrategy" />
18 <!-- OSGI Service Export to publish the Event Handler-->
19 <osgi:service ref="balanceAlertEventHandler" interface="com.sybase365.mobiliser.
       framework.event.model.Handler" />
```

Listing 9.30: Event Handler OSGI Configuration Example

Then you can fill all that in your event handler

```
1  <tx:annotation-driven transaction-manager="transactionManager" />
2
3  <bean id="balanceAlertEventHandler" class="com.sybase365.mobiliser.money.jobs.
       event.handler.balancealert.BalanceAlertEventHandler">
4    <property name="daoFactory" ref="daoFactory" />
5    <property name="configuration" ref="configuration" />
6    <property name="messagingService" ref="authenticatingMessagingService" />
7    <property name="eventGenerator" ref="eventGenerator" />
8  </bean>
9
10 <prefs:node id="prefsNode" class="com.sybase365.mobiliser.money.jobs.event.
       handler.balancealert.BalanceAlertEventHandler" encrypted-keys="internal.
       service.password" />
11
12 <bean id="configuration" class="com.sybase365.mobiliser.money.jobs.event.handler.
       balancealert.BalanceAlertEventHandlerConfiguration">
13   <property name="preferences" ref="prefsNode" />
14 </bean>
15
16 <bean id="authenticatingMessagingService" class="org.springframework.aop.
       framework.ProxyFactoryBean">
17   <property name="target" ref="messagingService" />
18   <property name="interfaces">
19     <list><value>com.sybase365.mobiliser.util.messaging.service.api.
           IMessagingService</value></list>
20   </property>
21   <property name="interceptorNames">
22     <list><value>authenticatingInterceptor</value></list>
23   </property>
24 </bean>
25
26 <bean id="authenticatingInterceptor" class="com.sybase365.mobiliser.security.
       InternalServiceCallAuthenticatorFactoryBean">
27   <constructor-arg ref="internalAuthenticatorFactory" />
28   <constructor-arg ref="configuration" />
29 </bean>
```

Listing 9.31: Event Handler Spring Configuration Example

Chapter 10

Workflows

10.1 Setup and dependencies

Each workflow bundle is located in a separate project with the parent project com.sybase365.mobiliser.money.flows. The reason behind the separate workflow bundle is to offer an additional abstraction layer between our services and business logic and the currently used Activiti workflow engine. Each workflow bundle functions as an universal entry point to process a certain workflow, without knowing the details of which workflow engine is used. A workflow bundle currently holds both API and implementation packages, with the exception of the transaction workflow bundle. Each implementation is published to the OSGi service registry under the used interface.

The general structure of a workflow bundle looks like this:

```
1     src/main/java/com/sybase365/mobiliser/money/flows/customer/api
2     src/main/java/com/sybase365/mobiliser/money/flows/customer/impl
```

Listing 10.1: Package Structure

The common parent for each new workflow bundle is

```
1 <parent>
2     <groupId>com.sybase365.mobiliser.money</groupId>
3     <artifactId>com.sybase365.mobiliser.money.flows</artifactId>
4     <version>5.1.0-SNAPSHOT</version>
5 </parent>
```

Listing 10.2: Workflow Parent Project

Each new workflow bundle has to use the following two dependencies to be able to use the Activiti functionality. These dependencies include the Activiti engine as well as the mobiliser Activiti utilities bundle, which offers an abstraction of the Activiti engine, to ease the implementation of the workflow. The mobiliser Activiti utilities, especially the IWorkflow interface is the main entry point to orchestrate and communicate with the Activiti engine.

```
1 <dependency>
2       <groupId>com.sybase365.mobiliser.util.workflow</groupId>
3       <artifactId>com.sybase365.mobiliser.util.workflow.activiti</artifactId>
4       <version>${version.workflow}</version>
5 </dependency>
6
7 <dependency>
8       <groupId>org.activiti</groupId>
9       <artifactId>activiti-engine</artifactId>
10 </dependency>
```

Listing 10.3: Required Dependencies

Additionally to those dependencies, frame business logic, service and utility dependencies have to be added in order to integrate the business logic and services layer, if needed.

10.2 API Requirements

The API has to define an interface to abstract away the implementation methods needed to use the workflow implementation. The currently available implementations have proven, that the following set of methods is useful and should be implemented by any future workflow bundle:

```
1 startProcess(MobiliserRequestType request, long callerId) throws
      WorkflowFailedException;
2 continueProcess(MobiliserRequestType request, long callerId, final Set<String>
      privileges) throws WorkflowFailedException;
3 findPendingProcesses(MobiliserRequestType request, long callerId, final Set<
      String> privileges) throws WorkflowFailedException;
4 getPendingProcessDetails(final String taskId, final long callerId, final Set<
      String> privileges);
```

Listing 10.4: Standrad API Methods

Almost every workflow bundle implements those method in their specific context with corresponding request and response type implementations. Some bundles also add additional methods to the set. Currently no base interface definition exists to keep the versatility of each workflow bundle, but might be introduced in the future in favor of re-usability.

```
1  public interface ICustomerFlowControl {
2
3      CreateCustomerResponse startProcess(CreateCustomerRequest request,
4          long callerId) throws WorkflowFailedException;
5
6      CreateFullCustomerResponse startProcess(CreateFullCustomerRequest request,
7          long callerId) throws WorkflowFailedException;
8
9      ContinuePendingCustomerResponse continueProcess(
10         ContinuePendingCustomerRequest request, long callerId,
11         final Set<String> privileges) throws WorkflowFailedException;
12
13     FindPendingCustomersResponse findPendingProcesses(
14         FindPendingCustomersRequest request, long callerId,
15         final Set<String> privileges) throws WorkflowFailedException;
16
17     GetPendingcustomerDetailsResponse getPendingProcessDetails(
18         final String taskId, final long callerId,
19         final Set<String> privileges);
20 }
```

Listing 10.5: Example Customer Workflow Interface

The actual implementation of the workflow interfaces will utilize the mobiliser Activiti workflow bundle, which offers access to the Activiti workflow engine. The principle behind a workflow bundle is to be able to suspend a process until further approval. This loose requirement leads to the set of standard methods listed above in order to start a process resulting in the creation of a new Activiti process, find suspended processes (or more precise the task in which the process was suspended), get details about a suspended process (current taskId, etc.) and ultimately continue the suspended process (from the task it was suspended in) by either approving or rejecting it. The Activiti workflow engine offers corresponding methods to initiate an Activiti process flow, to pause and continue it. As well as methods to store and retrieve information as the workflow progresses. This

information is ultimately serialized and stored in Activiti specific tables in the database. The retrieval of this information follows the same deserialisation path.

10.3 Defining the Workflow

The center of the workflow bundle is the workflow definition, which uses a BPM XML notation.

Each worklfow definition is written in its own .xml file within the META-INF/spring folder.

```
/src/main/resources/META-INF/customer-creation.bpmn20.xml
```

Listing 10.6: Workflow Definition File

The whole workflow definition is done within the element <process>

```
<process id="com.sybase365.mobiliser.money.flows.customer.creation" name="Process
    creating and approving a new full customer">
```

Listing 10.7: The <process> element

Each workflow is started by <startEvent>, which is the entry point for the first sequence flow.

```
<startEvent id='theStart' />
<sequenceFlow id='flow1' sourceRef='theStart' targetRef='approveTask' />
```

Listing 10.8: the <startEvent> Element

The whole workflow is composed of sequence flows, service and user tasks, which define which path to follow and which actions to take along the workflow. In order to define a sequence flow an entry point (source) and an exit point (target) have to be defined. Usually each sequence flow is accompanied with a service task, which defines what actions to take in order ot proceed with the workflow. The source of the sequence flow then has to be identical with the service tasks Id in order to link both.

```
<serviceTask id="setBlacklistReasonFullCustomer" name='set blacklist reason'
      Activiti:delegateExpression="${
          com_sybase365_mobiliser_money_flows_customer_creation_
        setBlacklistReasonDelegate}" />
<sequenceFlow id='flow5' sourceRef='setBlacklistReasonFullCustomer' targetRef='
    createBlrFullCustomer' />
<serviceTask id="createBlrFullCustomer" name='create blr customer'
      Activiti:delegateExpression="${
          com_sybase365_mobiliser_money_flows_customer_creation_
        createFullCustomerInvoker}" />
<sequenceFlow id='flow6' sourceRef='createBlrFullCustomer' targetRef='
    theEndNotApproved' />
```

Listing 10.9: The <serviceTask Element

Multiple sequence flows in conjunction with a gateway can be used to make decisions based on the Activiti context, without actually calling any external code.

```
<exclusiveGateway id="exclusiveGw" name="Exclusive Gateway approval" />
<sequenceFlow id="beginCallFlow1" sourceRef="exclusiveGw" targetRef="
    createFullCustomer">
      <conditionExpression xsi:type="tFormalExpression"><![CDATA[${
          approvedByChecker && fullCustomer}]]></conditionExpression>
</sequenceFlow>
<sequenceFlow id="endFlow2" sourceRef="exclusiveGw" targetRef="
    setBlacklistReasonFullCustomer">
```

```
6       <conditionExpression xsi:type="tFormalExpression"><![CDATA[${!
            approvedByChecker && fullCustomer}]]></conditionExpression>
7 </sequenceFlow>
```

Listing 10.10: The <exclusiveGateway Element

The gateway has to be the source for all "deciding" sequence flows that you want to group logically together. A SPEL expression can then be used to evaluate variables stored in the Activiti context belonging to the current task.

Each workflow ends in one or multiple end events. Usually one success event and multiple failure events.

```
1 <endEvent id='theEnd' />
2 <endEvent id='theEndNotApproved' />
```

Listing 10.11: The <endevent> Element

For the end event (without distinguishing between different end events), a execution listener can and should be defined.

```
1 <extensionElements>
2       <Activiti:executionListener event="end" delegateExpression="${
            com_sybase365_mobiliser_money_flows_customer_creation_executionListener
            }" />
3 </extensionElements>
```

Listing 10.12: Registering an execution listener

```
1 <bean id="executionListener"
2     class="com.sybase365.mobiliser.util.workflow.Activiti.tasks.
          ThreadLocalVariableExecutionListener">
3     <property name="variableName" value="response" />
4     <property name="workflow" ref="workflow" />
5 </bean>
6 \end{lstlisting]
7
8 This execution listener from the Activiti utils bundle makes sure that the
      response of a service task is serialized in a response variable and stored in
       the Activiti context.
9 This is important if the response of a service call is needed after a pending
      task has been completed, e.g. to indicate a successful customer creation by
      returning the createCustomer response.
10
11 \begin{lstlisting}[label=lst:WFExecutionListener2,caption={Registering an
      execution listener},language=xml, style=eclipse]
12 <extensionElements>
13       <Activiti:executionListener event="end" delegateExpression="${
             com_sybase365_mobiliser_money_flows_customer_creation_executionListener
             }" />
14 </extensionElements>
```

Listing 10.13: Excerpt form bundle.context.xml

Another very important element of the Activiti flow, especially in the dual approval context is a user task.

```
1 <userTask id="approveTask" name="Approve customer" />
2 <sequenceFlow id='flow3' sourceRef='approveTask' targetRef='exclusiveGw' />
```

Listing 10.14: The <userTask> Element

A user task is the signal for the Activiti engine to pause the currently running process until manual intervention. Upon suspending a task ID is available which can be used by the workflow implementation to identify the exact user task (currently our workflow bundles all use just one user task) and continue the workflow from there.

Another thing to note is: Activiti does support direct method calls from within the workflow. We on the other hand chose to use the delegation pattern, which means instead of defining a direct method call in a service task, we just use a "key", which is mapped to a corresponding delegate bean (implementing JavaDelegate) performing the operation we want.

To better organize the delegates we keep a map mapping the key used in the workflow definition to the delegate beans:

```
<bean id="beans"
    class="com.sybase365.mobiliser.util.workflow.Activiti.api.ExpressionBeansImpl
        ">
    <constructor-arg>
      <util:map>
        <beans:entry
          key="com_sybase365_mobiliser_money_flows_customer_creation_
            createFullCustomerInvoker">
          <beans:ref bean="serviceInvokerFull" />
        </beans:entry>
        <beans:entry
          key="com_sybase365_mobiliser_money_flows_customer_creation_
            createCustomerInvoker">
          <beans:ref bean="serviceInvoker" />
        </beans:entry>
    [...]
      </util:map>
    </constructor-arg>
</bean>
```

Listing 10.15: Excerpt from bundle-context.xml

The delegate bean can be a simple POJO implementing the JavaDelegate interface

```
<bean id="setBlacklistReasonDelegate"
    class="com.sybase365.mobiliser.money.flows.customer.impl.
        SetBlacklistReasonRequestHelper">
    <property name="requestVariableName" value="request" />
    <property name="blacklistReason" value="5" />
</bean>
```

Listing 10.16: A delaegate bean definition

Or a more sophisticated bean used in conjunction with a security interceptor and the run as manager to manipulate the request caller's privileges. Examples of this can be found in the customer and transaction workflow bundles. This is particularly useful, when integrating a dual approval workflow into existing services (e.g. createFullCustomer). It allows for the reuse of the same service when starting and continuing the workflow, simply by manipulating the caller's privileges.

```
<bean id="serviceInvokerFull" class="org.springframework.aop.framework.
    ProxyFactoryBean">
    <property name="target" ref="serviceInvokerFullDelegate" />
    <property name="autodetectInterfaces" value="true" />
    <property name="interceptorNames">
      <list>
        <value>methodSecurityMetadataSourceAdvisor</value>
      </list>
    </property>
</bean>

<bean id="serviceInvokerFullDelegate"
    class="com.sybase365.mobiliser.util.workflow.Activiti.tasks.
        MobiliserServiceInvoker">
    <property name="argumentVariableName" value="request" />
    <property name="method">
```

```
15        <bean class="org.springframework.util.ClassUtils"
16          factory-method="getMethodIfAvailable">
17          <constructor-arg
18            value="com.sybase365.mobiliser.money.services.api.ICustomerEndpoint" />
19          <constructor-arg value="createFullCustomer" />
20          <constructor-arg>
21            <util:list value-type="java.lang.Class">
22              <beans:value>com.sybase365.mobiliser.money.contract.v5_0.customer.
                    CreateFullCustomerRequest</beans:value>
23            </util:list>
24          </constructor-arg>
25        </bean>
26      </property>
27      <property name="resultVariableName" value="response" />
28      <property name="service" ref="customerEndpoint" />
29      <property name="transactionTemplate" ref="serviceInvokerTxnTemplate" />
30      <property name="autodetectGlobalRollbackOnly" value="true" />
31  </bean>
```

Listing 10.17: Using the Security Interceptor (excerpt from bundle-context.xml)

The following excerpt shows the complete bundle-context-security.xml used in order to configure the security interceptor. This shows how the run as manager uses another Java delegate to replace or add privileges the caller might additionally need in order to call the method encapsulated within the interceptor.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:beans="http://www.
         springframework.org/schema/beans"
4    xmlns:aop="http://www.springframework.org/schema/aop" xmlns:security="http://
         www.springframework.org/schema/security"
5    xmlns:util="http://www.springframework.org/schema/util"
6    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
         springframework.org/schema/beans/spring-beans-3.1.xsd
7    http://www.springframework.org/schema/security http://www.springframework.org/
         schema/security/spring-security-3.1.xsd
8    http://www.springframework.org/schema/aop http://www.springframework.org/schema
         /aop/spring-aop-3.0.xsd
9    http://www.springframework.org/schema/util http://www.springframework.org/
         schema/util/spring-util-3.1.xsd">
10
11   <beans:bean id="customerPrivilegeHelper"
12     class="com.sybase365.mobiliser.money.flows.customer.impl.
           CustomerTypePrivilegeHelper">
13     <beans:property name="customerLogic" ref="customerLogic" />
14   </beans:bean>
15
16   <!-- SECURITY STUFF -->
17
18   <beans:bean id="runAsManager"
19     class="com.sybase365.mobiliser.util.workflow.util.security.
           ExpressionBasedRunAsManager">
20     <beans:property name="bindVariables">
21       <beans:map>
22         <beans:entry key="privilegeHelper" value-ref="customerPrivilegeHelper" />
23       </beans:map>
24     </beans:property>
25     <beans:property name="key" value="mobiliser_run_as" />
26   </beans:bean>
27
28   <beans:bean id="delegateMetadataSource"
29     class="org.springframework.security.access.method.
           MapBasedMethodSecurityMetadataSource">
30     <beans:constructor-arg>
```

```
31        <beans:map>
32          <beans:entry key="org.activiti.engine.delegate.JavaDelegate.execute">
33            <list>
34              <beans:bean
35                class="com.sybase365.mobiliser.util.workflow.util.security.
                      ExpressionConfigAttribute">
36                <beans:constructor-arg>
37                  <beans:bean factory-bean="parser"
38                    factory-method="parseExpression">
39                    <beans:constructor-arg
40                      value="T(org.apache.commons.lang.StringUtils).defaultIfEmpty
                            (#privilegeHelper.getExecutePrivilege(#invocation.
                            getArguments()[0].getVariable('request').getCustomer().
                            getCustomerTypeId()), '__DUMMY_ROLE__')" />
41                  </beans:bean>
42                </beans:constructor-arg>
43              </beans:bean>
44              <beans:bean
45                class="org.springframework.security.access.SecurityConfig">
46                <beans:constructor-arg>
47                  <util:constant
48                    static-field="org.springframework.security.access.vote.
                          AuthenticatedVoter.IS_AUTHENTICATED_FULLY" />
49                </beans:constructor-arg>
50              </beans:bean>
51              <beans:bean
52                class="org.springframework.security.access.SecurityConfig">
53                <beans:constructor-arg>
54                  <util:constant
55                    static-field="org.springframework.security.access.vote.
                          AuthenticatedVoter.IS_AUTHENTICATED_REMEMBERED" />
56                </beans:constructor-arg>
57              </beans:bean>
58              <beans:bean
59                class="org.springframework.security.access.SecurityConfig">
60                <beans:constructor-arg>
61                  <util:constant
62                    static-field="org.springframework.security.access.vote.
                          AuthenticatedVoter.IS_AUTHENTICATED_ANONYMOUSLY" />
63                </beans:constructor-arg>
64              </beans:bean>
65            </list>
66          </beans:entry>
67        </beans:map>
68      </beans:constructor-arg>
69    </beans:bean>
70
71    <beans:bean id="unanimousBasedAccessMgr"
72      class="org.springframework.security.access.vote.AffirmativeBased">
73      <beans:property name="decisionVoters">
74        <beans:list>
75          <beans:bean
76            class="org.springframework.security.access.vote.AuthenticatedVoter" />
77        </beans:list>
78      </beans:property>
79    </beans:bean>
80
81    <beans:bean id="securityInterceptor"
82      class="org.springframework.security.access.intercept.aopalliance.
          MethodSecurityInterceptor">
83      <beans:property name="accessDecisionManager" ref="unanimousBasedAccessMgr" />
84      <beans:property name="authenticationManager" ref="authenticationManager" />
85      <beans:property name="securityMetadataSource" ref="delegateMetadataSource" />
86      <beans:property name="runAsManager" ref="runAsManager" />
87    </beans:bean>
```

```
88
89   <beans:bean id="methodSecurityMetadataSourceAdvisor"
90     class="org.springframework.security.access.intercept.aopalliance.
         MethodSecurityMetadataSourceAdvisor">
91     <beans:constructor-arg value="securityInterceptor" />
92     <beans:constructor-arg ref="delegateMetadataSource" />
93     <beans:constructor-arg value="delegateMetadataSource" />
94   </beans:bean>
95
96 </beans>
```

Listing 10.18: Complete bundle-context-security.xml)

10.4 Implementing the workflow interface

Just configuring the workflow however is not all one needs to do. As described beforehand, we will have to define and implement a interface in order to work with the Activiti engine, or more precise with the mobiliser Activiti utility bundle, to start and finish a process instance along several (user) tasks and to manipulate the Activiti context and the process instance.

In the following the before mentioned "standard" interface methods are explained.

Each workflow implementation needs to utilize the IWorkflow of the mobiliser Activiti utility bundle in order to manage the Activiti engine.

10.4.1 startProcess(MobiliserRequestType request, long callerId) throws WorkflowFailedException;

```
1 CreateFullCustomerResponse startProcess(CreateFullCustomerRequest request, long
      callerId) throws WorkflowFailedException;
```

Listing 10.19: Example startProcess method

This method is usually triggered by a mobiliser service, but can also be used from the business logic directly (see transaction workflow). Its sole purpose is to store any important information, like the request or certain request parameters, in the Activiti context and to start the actual process.

```
1 String processInstanceId = getWorkflow().startProcess(getProcessId(),
      workflowParameters);
```

Listing 10.20: Starting a new process instance

The process ID is the unique Id used in the workflow definition. Workflow parameters may include any Java object which implements the Serializable interface. The returned process instance ID is merely informative and not needed in the course of the workflows lifespan, but for debuging purposes. the finding of a suspended process and the continuation is done via a task ID, which has to be queried.

10.4.2 findPendingProcesses(MobiliserRequestType request, long callerId, final Set<String> privileges) throws WorkflowFailedException;

```
1 FindPendingCustomersResponse findPendingProcesses(FindPendingCustomersRequest
      request, long callerId, final Set<String> privileges) throws
      WorkflowFailedException;
```

Listing 10.21: Example findPendingProcesses method

This method is triggered by a separate service, most likely used by some front end like the CST, to search for currently suspended processes. The request usually holds search parameters which are used by the Activiti engine to find matching tasks. The privilege set is used to check if the caller has the appropriate privileges to see each task that was found by the engine. The mobiliser Activiti utility bundle offers callback mechanism via findTasks(), which calls a certain method of a visitor object each time a task instance that matched the given search criteria was found. The findTasks method, or more precise the TaskVisitor that needs to be implemented offers the possibility to return a set of parameters for each task that was found, which then can be used to build a response object returned by the findPendingProcesses() method.

```
getWorkflow().findTasks(getProcessId(), KEY_APPROVE_TASK,
        FormatUtils.getSaveDate(request.getToDate()),
        FormatUtils.getSaveDate(request.getFromDate()),
        searchParameters, 0, getMaxSearchResults(),
        new IWorkflow.TaskVisitor() {

            @Override
            public void visit(String processInstanceId, String taskId,
                Map<String, Object> parameters) {

            // relying on the cache here
            final WrkCustomer wrkCustomer = customerLogic
                .getWrkCustomer(((Integer) parameters
                    .get(KEY_CUSTOMER_TYPE_ID)).intValue());

            if (wrkCustomer != null) {
                if (!privileges.contains(wrkCustomer
                    .getCheckerPrivilege())) {
                return; // not allowed access to this one
                }
            }

            // each time we get "visited" a new task has been found
            // matching the search criteria, so we convert it into a
            // PendingCustomer object and add it to the list of
            // searchResults
            PendingCustomer pc = new PendingCustomer();

            pc.setCustomerTypeId(((Integer) parameters
                .get(KEY_CUSTOMER_TYPE_ID)));
            pc.setFirstName((String) parameters.get(KEY_FIRST_NAME));
            pc.setLastName((String) parameters.get(KEY_LAST_NAME));
            pc.setMsisdn((String) parameters.get(KEY_MSISDN));
            pc.setUserName((String) parameters.get(KEY_USERNAME));
            pc.setEmail((String) parameters.get(KEY_EMAIL));
            pc.setDisplayName((String) parameters
                .get(KEY_DISPLAYNAME));
            pc.setTaskId(taskId);

            searchResults.add(pc);
            }

            @Override
            public Set<String> getTaskParameters() {

            // just return the set of paraemter names, we want to
            // get for each task, which matches the search criteria
            Set<String> variables = new HashSet<String>();
            variables.add(KEY_CUSTOMER_TYPE_ID);
            variables.add(KEY_FIRST_NAME);
            variables.add(KEY_LAST_NAME);
            variables.add(KEY_MSISDN);
            variables.add(KEY_USERNAME);
            variables.add(KEY_DISPLAYNAME);
            variables.add(KEY_EMAIL);
```

```
56              variables.add(KEY_REQUEST);
57
58              return variables;
59              }
60          });
```

Listing 10.22: Search for tasks

10.4.3 getPendingProcessDetails(final String taskId, final long callerId, final Set<String> privileges);

```
1 GetPendingcustomerDetailsResponse getPendingProcessDetails(final String taskId,
      final long callerId, final Set<String> privileges);
```

Listing 10.23: Example getPendingProcessDetails method

This method is usually used by a front end to retrieve more details about a suspended process. It usually returns values previously stored in the Activiti context when the process was started.

```
1 Map<String, Object> taskVariables = getWorkflow().getTaskVariables(taskId,
      parameterNames);
```

Listing 10.24: Example startProcess method

The variables returned by the getTaskVariables() can be used to build an appropriate response. It is noteworthy that getTaskVariables() does not claim, that is block, the process.

10.4.4 continueProcess(MobiliserRequestType request, long callerId, final Set<String> privileges) throws WorkflowFailedException;

```
1 ContinuePendingCustomerResponse continueProcess(ContinuePendingCustomerRequest
      request, long callerId, final Set<String> privileges) throws
      WorkflowFailedException;
```

Listing 10.25: Example continuePendingProcess method

This method is, as the name suggests, used to continue a previously suspended process. the request going in the method usually contains the task ID and an approval/reject flag. However before a task can be finished, it has to be claimed. When this happens no other person is able to work with the suspended process, until it is released again.

```
1 Map<String, Object> parameters = getWorkflow().claimTaskAndSelectParameters(
      request.getTaskId(), callerId, parameterNames);
```

Listing 10.26: Claim a task

The method to claim a task takes a set of parameter names, for which the values found in the Activiti context are returned. These can be used to determine how to continue with the process flow. After successfully claiming the task and evaluating any parameters returned, the method completeTask() has to be used. One could think to just call the business logic or a service directly, but that would bypass the workflow definition and would leave the task in an unfinished state. The elements of the result set that goes into completeTask() is stored in the Activiti context and can be used to determine sequence flows from the workflow definition. Usually the approve/reject flag is stored in the results and thus in the Activiti context, which is then used by the gateway of the workflow definition to continue with the appropriate sequence flow and the next service task (which will then call the appropriate business logic or service method).

```
1 results.put("approvedByChecker", Boolean.valueOf(request.isApprove()));
```

```
2 getWorkflow().completeTask(request.getTaskId(), results, callerId);
```

Listing 10.27: Approve a task

After the task was completed we have access to the response through the Activiti context, since it was put in the context by the execution listener mentioned in the workflow definition.

```
1 final Object _resp = getWorkflow().getGlobalVariable("response");
```

Listing 10.28: Retrieve the response variable

10.5 Integrating the workflow

The so defined and implemented interface can then be integrated in already available services and business logic. Usually, if integrating in an already existing service, the service request is intercepted, analyzed and a potential dual approval configuration is applied (checking for the existence of a configuration, checking the callers maker, checker and execute privileges, etc.).

```
1 // check if there exists a dual approval configuration for the request's
2     // customer type
3     WrkCustomer dpConf = this.customerLogic.getWrkCustomer(request
4         .getCustomer().getCustomerTypeId());
5     if (dpConf != null
6         && !this.callerUtils.hasPrivilege(dpConf.getExecutePrivilege())) {
7
8         // there exists a dual approval configuration for the request's
9         // customer type and the caller does not have the configured execute
10        // privilege => check if he is allowed to start the dual approval
11        // process
12        if (!this.callerUtils.hasPrivilege(dpConf.getMakerPrivilege())) {
13        // the caller misses the execute and the maker privilege
14        throw new AccessDeniedMobiliserServiceException(
15            "The caller has insufficient privileges to execute this service");
16        }
17
18        // caller has the privilege to start the dual approval process
19
20        // start process ...
21        return this.customerFlowControl
22            .startProcess(request, getCallerId());
23    }
```

Listing 10.29: Example of a workflow integration

When the workflow is continued usually the same service is called to proceed with the workflow. Since the calling user is in most cases the same, who started the workflow (to keep track of the original initiator), the run as manager comes in handy, by adding the execute privilege to the callers privilege set. In other workflow configurations the process is started directly from the business logic, and also continued by directly calling business logic methods (see transaction workflow). There is no limitation as how to integrate the workflow into the mobiliser service and business logic layers.

Alongside the integration a set of new services needs to be written to search for suspended processes and to continue tasks. Those services however mostly just “redirect” to the workflow API.

```
1 @Override
2     public GetPendingcustomerDetailsResponse getPendingcustomerDetails(
3         GetPendingcustomerDetailsRequest request) {
4
5     return getCustomerFlowControl().getPendingProcessDetails(
6         request.getTaskId(),
```

```
7           getCallerId(),
8           new HashSet<String>(this.callerUtils.getCallerInformation()
9               .getActorPrivs()));
10      }
11
12      @Override
13      public ContinuePendingCustomerResponse continuePendingCustomer(
14          ContinuePendingCustomerRequest request)
15          throws EntityNotFoundException {
16
17      final ContinuePendingCustomerResponse response = this.customerFlowControl
18          .continueProcess(request, getCallerId(),
19              new HashSet<String>(this.callerUtils
20                  .getCallerInformation().getActorPrivs()));
21
22      getAuditContext().setAuditParameter(
23          MoneyAuditContextKeys.CUSTOMER_ID.toString(),
24          response.getCustomerId());
25      return response;
26      }
```

Listing 10.30: Search for and continue suspended tasks services

Chapter 11

Message Gateway

Clients can leverage the message gateway to submit messages to clients. The customer does not have to be present in the system to receive a notification.

The interface **com.sybase365.mobiliser.util.messaging.service.api.IMessagingService** provides commons operations for sending notifications.

```
1 public interface IMessagingService {
2
3     SendPlainTextResponse sendPlainText(final SendPlainTextRequest request);
4
5     SendTemplateResponse sendTemplate(final SendTemplateRequest request);
6
7 }
```

Listing 11.1: Messaging Service API

Common parameters for all message types are as follows:

- **receivers** list of recipients for the message, must match the message type. i.e. email addresses for emails, phonenumbers for sms
- **type** the type of message being sent, currently either sms or email
- **channel** the channel to use for sending this message. This is optional, but the client can specify a specif channel to use. Otherwise a default configured in the message gateway will be used. Example being sending of transaction notifications through a different channel than say mass marketing messages
- **subject** for message types which support subjects, sets the subject of the message. For non-templated emails, this sets the subject. For templated emails, this overrides the subject set in the template configuration.
- **referencedEntity** - clients can set the ID of the customer to whom this message is being sent and the message gateway will store this id along with the sent message to allow CST agents to see the messages sent to a particular customer.

The gateway allows messages to be templated, the client of the service only providing the necessary parameters for the template or the client may just submit the text of the message for a one-off notification.

Plain one-off messages have the following options:

- **sender** the sender of the message, should be in format matching the message type, see above
- **text** the actual text to send to the customer

Templated messages have the following options:

- **sender** allow overwriting the sender configured in the template
- **template** the name of the template to send
- **locale** optionally the locale (language, country, variant) used to match the template. Each template name can appear more than once in the database, so clients can send a locale along to match a specific configuration. NULL is allowed and will also only match the default template. A more specific request will match a less specific configuration is there is no match for the specific configuration. Example: client requests to send template XYZ sending locale en_NZ_vodafone. The template is configured three times in the database with en, de, and NULL. The en template would be selected.
- **parameters** - template texts can contain ${placeholders} which can then be replaced with parameters sent by the client. Parameters are discussed in detail further on.

11.0.1 Templates

Templates can be stored in the database for defining messages in a standard way and allowing clients to simply provide placeholders to fill in needed data. A template consists of the following parameters:

- **name** the name of the template
- **locale** consists of language, country and variant (the combination name + locale + type must be unique)
- **sender** the sender of the message (From: for email, sending MSISDN/shortcode for SMS)
- **template type** currently either sms or email
- **content** the actual string content of the message (for SMS the text, for email the body, for multipart messages the plain part)
- **content type** the content type of the content. in most instances text/plain, but if you are sending emails with a single part html, then text/html or whatever matches what you are sending
- **charset** the database can of course hold strings any unicode codepoint, but this should give the backend a hint about which charset is needed. If the text for example only contains ascii characters, set us-ascii (even though the database stores it as utf-8). If the templates are all in russian for example, set it to UTF-8 so the backend knows it needn't even attempt any simpler charset. This setting only affects what gets sent out over the wire. Java will handle everything correctly internally.
- **alt content** for message types which support multiple parts, the alternative part. Currently only used for mime multipart emails, this is the alternative part - most likely the text/html part.
- **alt content type** as above but for the alternative part
- **alt charset** as above but for the alternative part

Attachments may be added to outgoing messages if the message types supports it (email for example). An attachment consists of the following parameters:

- **name** the name of the attachment. for messages that support attachments (email) this will be the attachment name (will most likely be a file name)
- **the content type of the attachment** (application/pdf, image/png etc)
- **charset** [optional] if the attachment is textual, the charset with which the bytes were encoded into the BLOB field. REQUIRED for text types. Do not make the gateway send blibb.txt as application/octet-stream or even worse text/plain without a charset!
- **content id** [optional] used for embedded images in HTML emails. Your HTML must reference a content id with cid:name. This is that name.
- **is related** whether this attachment is related to the text of the message. Only set true here if you plan to reference this attachment from your text (which requires you to also set the content id)
- **blob content** the actual content bytes of the attachment. Please set the content type correctly and if configuring text contents please convert to bytes using the correct charset and configure it here.

11.0.2 Template Parameters

The framework supports a few template parameter types out of the box. New types can be added by implemented an interface and added it to the container.

Get Parameters

Called “get” Parameters, these are simply parameters fetched unchanged from the client map and added to the template. They come in two variants: get.<parameter name> and get-confidential.<parameter name>. The only difference being that the second variants causes the message to be marked confidential, even if the template itself was not configured as confidential.

GET Parameter Usage The parameter is simply configure embedded in the text where the parameter should appear.

```
1 Here is a ${get.someparameter} text.
```

Listing 11.2: Example GET Usage

Clients would then provide the following in the parameter map: **someparameter=somevalue**

The resulting string would be:

```
1 Here is a somevalue text.
```

Listing 11.3: Example GET Result

There is no difference in functionality between the get and get-confidential variants.

Attachment Parameters

These parameters allow an attachment to be dynamically added to a message by fetching it from an inlined base64 encoded parameter passed along with the request. The data is serialised into a single parameter. Each parameter consists of a key and a value with ’:’ used as the delimiter and ’;’ used as the delimiter between the two. Currently there are two keys supported: content and content-type. content should hold the base64 encoded bytes of the data. content-type includes the content type and optionally the filename and charset of the data. If you add either of these optional values, you’ll need to escape the ; with a backslash. Here is an example:

```
1 Please find the log attached to this message.${base64_att.${get.attdata}}
```

Listing 11.4: Example base64_att

Clients would then provide the following in the parameter map: **attdata=content-type: text/plain\;filename=application.log\;charset=UTF-8; content: YmxpYml=**

```
1 Please find the log attached to this message.
```

Listing 11.5: Example base64_att Result

A file with those bytes, the name application.log and content type text/plain;charset=utf-8 would be added as an attachment to the message.

Fetch Parameters

Called “fetch” Parameters, these parameters allow an attachment to be dynamically added to a message by fetching it from a URL. They come in two variants: fetch.<link> and fetchcid.<cid>.<link>. The only difference being the second variants allows assigning a CID so it can be referenced by an inline message part, as seen previously in the static attachments. The parameters will most likely be used in combination with a get parameters to make the URL have a dynamic portion.

Fetch Parameter Usage The fetch parameter may appear anywhere in the text itself, but it will then just be removed and the resolved attachment added to the message. For clarity, it is best to add them as the last part of the message text. The fetchcid parameters on the other hand should appear exactly where the inline attachment should appear in the text.

```
1 Please find your invoice attached to this message.
2
3 ${fetch.http://localhost:8080/mobiliser/rest/getInvoiceBytes?invoiceId=${get.
      invoiceId}}
```

Listing 11.6: Example FETCH Usage

Clients would then provide the following in the parameter map: **invoiceId=123456**

The resulting string would be:

```
1 Please find your invoice attached to this message.
```

Listing 11.7: Example FETCH Result

The invoice 123456 would be added as an attachment to the message.

A fetchcid example would be:

```
1 In this image ${fetchcid.38734.http://localhost:8080/mobiliser/rest/
      getPictureBytes?pictureId=${get.mobPicId}} you see Mobiliser.
```

Listing 11.8: Example FETCHCID Usage

Clients would then provide the following in the parameter map: **mobPicId=123456**

The resulting string would be:

```
1 In this image cid:38734 you see Mobiliser.
```

Listing 11.9: Example FETCHCID Result

The attachment would have been added to the message and the parameter replaced by the cid of the attachment. This would normally be used in HTML, but here is seen in plain text to make the example simpler.

11.0.3 Custom Parameters

You can add a new custom parameter type to the framework by implementing a **IParameterResolver**.

```
1 public interface IParameterResolver {
2
3     ResolveResult resolve(final String key, final Map<String, String> parameters)
          ;
4
```

```
5       boolean supports(final String key);
6   }
```

Listing 11.10: IParameterResolver API

```
1       boolean confidential;
2
3       List<IResolvedReference> references;
4
5       String value;
6   }
```

Listing 11.11: ResolveResult API

So your implementation will accept a key found by the template parser and the current set of parameters and will resolve it into a string value, a list of references (attachments) and also a confidentiality flag.

To export it to the OSGi registry:

```
1   <osgi:service ref="myCustomerResolver"
2     interface="com.sybase365.mobiliser.util.messaging.template.impl.api.
          IParameterResolver" ranking="10">
3     <osgi:service-properties>
4       <beans:entry key="origin" value="mobiliser" />
5     </osgi:service-properties>
6   </osgi:service>
```

Listing 11.12: Parameter Resolver Export

Chapter 12

Channel Manager

Mobiliser provides a framework for sending out correspondence to customers. Standard implementations are able to send things like SMS, USSD or emails and customized channel can be added at any time. The Channel Manager provides the abstract between the message gateway and the actual protocol implementation used to send the message to the customer. The manager, as the name suggests, manages the channels in the system and routes messages to the correct outgoing channel which then does the actual sending.

This chapter describes the custom configuration and how to implement and add customized channels. See “Mobiliser Framework Architecture and Design” guide for more information on the design and architecture.

12.1 Configuration

Channels are implemented as bundles, as are the other business logic units. Since channels must be instantiated based on configuration, each bundle will export an **IChannelFactoryBean**. This bean will know how to instantiate and configure the requested channel and also how to clean up and properly destroy the instance.

```
1  public interface IChannelFactoryBean {
2
3      /**
4       * Destroy a previously created bean.
5       *
6       * @param channel
7       *             the channel
8       */
9      void destroyChannel(final Channel channel);
10
11     /**
12      *
13      * @param preferencesNode
14      *             the preferences node to use for creating the channel
15      * @return the channel
16      */
17     Channel instantiateChannel(final IPreferences preferencesNode);
18 }
```

Listing 12.1: Java IChannelFactoryBean API

For customisation work, you should not have to implement this interface. A default implementation com.sybase365.mobiliser.util.messaging.channelmanager.util.impl.ChannelFactoryBean is available and simply takes the preferences node and creates a bean factory from a spring configuration file and pulls out a configured **Channel** based on the preferences node. The factory bean can collect other services from the OSGi registry and inject them into each instance created by the factory. These we call dependent beans. It needs **Resource**s which are just the XML configurations for the channel as well as default properties if necessary.

The channel configuration will use standard ${} constructs to allow configuration and we can define the defaults here. When the channel factory is exported, we add the mandatory service property **channelType**, which is used when configuring channels to reference the type of channel we want to instantiate.

12.1.1 Channel XML Configuration

Your bundle configuration will simply export the factory bean as in the following example.

```
<?xml version="1.0" encoding="utf-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:beans="http://www.
      springframework.org/schema/beans"
  xmlns:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
      springframework.org/schema/beans/spring-beans-3.0.xsd
  http://www.springframework.org/schema/util http://www.springframework.org/
      schema/util/spring-util-3.0.xsd">

  <bean id="emailChannelFactory"
    class="com.sybase365.mobiliser.util.messaging.channelmanager.util.impl.
        ChannelFactoryBean">
    <property name="beanName" value="emailChannel" />
    <property name="dependentBeans">
      <util:map>
        <entry key="systemClock" value-ref="systemClock" />
      </util:map>
    </property>
    <property name="resources">
      <list>
        <value>classpath:/com/sybase365/mobiliser/util/messaging/channelmanager/
            channels/email/spring-beans.xml</value>
      </list>
    </property>
    <property name="defaultProperties">
      <bean class="org.springframework.beans.factory.config.PropertiesFactoryBean
          ">
        <property name="location" value="classpath:/com/sybase365/mobiliser/util/
            messaging/channelmanager/channels/email/default-properties.properties
            " />
      </bean>
    </property>
  </bean>
</beans>
```

Listing 12.2: Bundle Configuration for Channel

```
<?xml version="1.0" encoding="utf-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:osgi="http://www.eclipse.org/gemini/blueprint/schema/blueprint"
xmlns:beans="http://www.springframework.org/schema/beans"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
    springframework.org/schema/beans/spring-beans-3.0.xsd http://www.eclipse.org/
    gemini/blueprint/schema/blueprint http://www.eclipse.org/gemini/blueprint/
    schema/blueprint/gemini-blueprint-1.0.xsd">

  <osgi:service ref="emailChannelFactory"
  interface="com.sybase365.mobiliser.util.messaging.channelmanager.util.
      IChannelFactoryBean"
  ranking="10">
    <osgi:service-properties>
      <beans:entry key="channelType" value="email" />
```

```
13        <beans:entry key="origin" value="mobiliser" />
14      </osgi:service-properties>
15    </osgi:service>
16    <osgi:reference id="systemClock"
17    interface="com.sybase365.mobiliser.util.tools.dateutils.ISystemClock"
18    sticky="false" />
19 </beans>
```

Listing 12.3: OSGi Bundle Configuration for Channel

In the example we are configuring a factory to create **EmailChannel**s, which need a **ISystemClock** reference, and whose configuration is located at the path seen in the XML. A properties file is also referenced which contains the default configuration options. The channel type is called “email”, but could be any pertinent unique name.

The channel factory configuration is done separately.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
       springframework.org/schema/beans/spring-beans-3.0.xsd">
5
6    <bean id="emailChannel"
7      class="com.sybase365.mobiliser.util.messaging.channelmanager.channels.email.
         EmailChannel">
8      <property name="mailSender" ref="mailSender" />
9      <property name="channelId" value="${channelId}" />
10     <property name="systemClock" ref="systemClock" />
11     <property name="pgpUtils">
12       <bean
13         class="com.sybase365.mobiliser.util.messaging.channelmanager.channels.
             email.config.PgpUtilsFactoryBean">
14         <property name="sign" value="${mail.sign}" />
15         <property name="hashAlgorithm" value="${sign.hashAlgorithm}" />
16         <property name="keyId" value="${sign.keyId}" />
17         <property name="passphrase" value="${sign.passphrase}" />
18         <property name="secretRingResource" value="${sign.secretRingResource}" />
19       </bean>
20     </property>
21     <property name="sign" value="${mail.sign}" />
22   </bean>
23
24   <bean id="mailSender"
25     class="com.sybase365.mobiliser.util.messaging.channelmanager.channels.email.
         config.JavaMailSenderFactoryBean">
26     <property name="host" value="${mail.host}" />
27     <property name="port" value="${mail.port}" />
28     <property name="username" value="${mail.username}" />
29     <property name="password" value="${mail.password}" />
30     <property name="protocol" value="${mail.protocol}" />
31   </bean>
32
33 </beans>
```

Listing 12.4: Configuration for Channel

There you can see how the two pieces fit together. The **ChannelFactory** pulls out the bean instance from the **BeanFactory** after configuring it based on the preferences values.

12.2 Runtime Configuration

All channel bundles may be added to the container and the **ChannelFactory** references will be fetched from the registery. To actually activate one, we need to add preferences. Currently that is done by simply adding a new subnode to the preferences node:
/com/sybase365/mobiliser/util/messaging/channelmanager/engine/impl/ChannelInstantiator/.
The name of the subnode doesn't matter, but try to choose something descriptive.

There you will add the configuration values needed by your channel implementation. These will be passed as described previously via the factory to the channel itself.

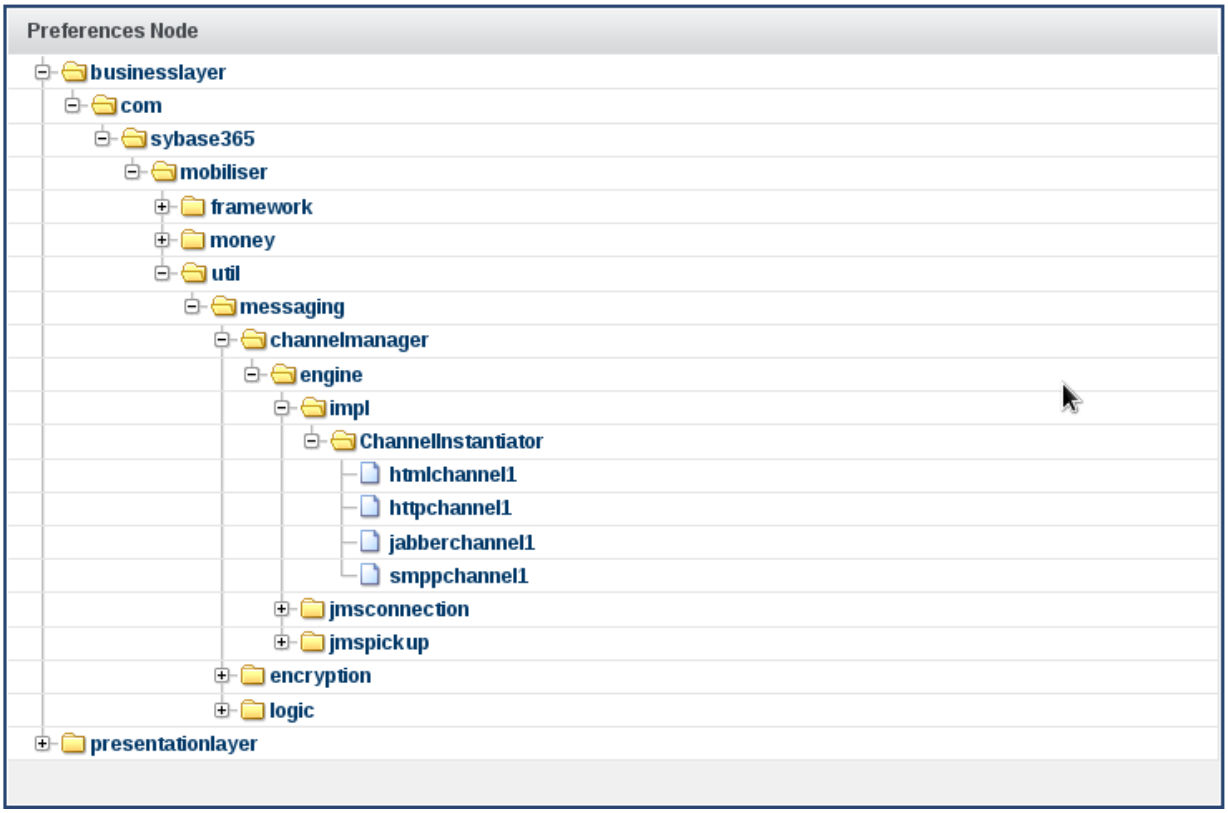

Figure 12.1: Sample Tree of Channel Preferences

There are also a couple of implicit configuration values that all channels use.

- **_channelType** This tells the framework which channel factory to use for this instance. This is the same value we used when exporting the factory through the OSGi registry.
- **_active** Configurations nodes are implicitly active, but at times it may be nice to have the option to turn the channel off without deleting the preferences node. By setting this value to "false", this configuration node will be disabled.
- **_host** Sometimes a channel should only run on a single node in a multi-node setup. This optional configuration allows setting the hostname on which this channel should run. It will only be configured in that container if the names match.

12.3 Implementation

The first thing to consider when implementing a new channel is how it will be used. That will affect the decision of which interfaces we need to implement. Use the descriptions in the section API of Mobiliser Framework Architecture and Design.

12.3.1 Common Methods

getChannelId

```
1     /**
2      * This channel's id.
3      *
4      * @return the channel's id
5      */
6     String getChannelId();
```

Listing 12.5: SendChannel getChannelId Method

All channels must implement this method which simply returns the channel's id.

12.3.2 Asynchronous

Such a channel will implement **AsynchronousSendReceiveChannel** if the messages between user and system flow asynchronously, e.g. like in the SMPP protocol.

There are a few methods you need to implement:

supports

```
1     /**
2      * Returns whether this channel supports the given message.
3      *
4      * @param message
5      *            the message
6      * @return true or false
7      */
8     boolean supports(final Message message);
```

Listing 12.6: SendChannel supports Method

This method allows **ChannelManager** to check if the channel is even capable of sending the message at all. It may be advantageous to configure multiple channels with the same id, but each one being only capable of sending messages of a certain type. **ChannelManager** will then be able to choose the correct one. Implementations of this method will most likely do an instanceof check.

```
1     @Override
2     public boolean supports(final Message message) {
3     return message instanceof EmailMessage;
4     }
```

Listing 12.7: SendChannel supports Method Implementation

send

```
1     /**
2      * Sends the message using this channel. You should first check whether this
3      * channel is capable of sending the message with {@link #supports(Message)}
4      *
5      * @param message
6      *            the message to send
7      * @throws ChannelException
```

```
8       *             if the message cannot be sent
9       */
10      void send(final Message message) throws ChannelException;
```

Listing 12.8: SendChannel send Method

This is the actual important part of the implementation. Here you will use the message parameter to actually send the information to the user. Using our email channel example, we will use the JavaMail API to construct an email message and to send it to the user via SMTP. The channel should only throw **ChannelException**s, using **NonTransientChannelException** for non-recoverable errors and **TransientChannelException** for recoverable errors.

```
1       @Override
2       public void send(final Message message) {
3       final EmailMessage emailMessage = (EmailMessage) message;
4
5       final MimeMessageHelper msgHelper = new MimeMessageHelper(
6           this.mailSender.createMimeMessage(),
7           MimeMessageHelper.MULTIPART_MODE_NO);
8
9           // set the mime message parameters
10
11      try {
12          // send message
13          this.mailSender.send(msgHelper.getMimeMessage());
14      } catch (final Exception e) {
15          throw new TransientChannelException(e);
16      }
17
18      try {
19          emailMessage.setExternalId(msgHelper.getMimeMessage()
20              .getMessageID());
21      } catch (final MessagingException e) {
22          LOG.warn("Failed to retrieve message id after sending message", e);
23      }
24      }
```

Listing 12.9: SendChannel send Method Implementation

setReceiveCallback

```
1       /**
2        * Set the receive callback through which this channel can push out incoming
3        * message to some consumer.
4        *
5        * @param receiveCallback
6        *             the receive callback
7        */
8       void setReceiveCallback(AsynchronousChannelReceiveCallback receiveCallback);
```

Listing 12.10: SendChannel setReceiveCallback Method

This will basically be a simple setter method for your channel, but the callback passed as an argument will be used by your channel to asynchronously receive incoming messages.

```
1       /**
2        * @param receiveCallback
3        *             the receiveCallback to set
4        */
5       public void setReceiveCallback(
6           final AsynchronousChannelReceiveCallback receiveCallback) {
```

```
7       this.receiveCallback = receiveCallback;
8       }
9
10      /**
11       * SMPP incoming.
12       */
13      public void someCustomerListenerMethod(final DeliverSm deliverSm) {
14
15      try {
16              // omitting message bytes / data coding checks
17
18          final String smsText = this.dataCodingUtils.createString(
19              shortMessage, dataCoding);
20
21          final SmsMessage message = (SmsMessage) this.messagingEngine
22              .parseSimpleTextMessage("sms", smsText);
23
24          message.setRecipient(deliverSm.getDestAddress());
25          message.setSender(deliverSm.getSourceAddr());
26
27              this.receiveCallback.receiveMessage(message,
28          this.sendChannelId, this.incomingChannelId);
29
30      } catch (final Exception e) {
31          LOG.warn("Failed to process incoming message", e);
32          throw new ProcessRequestException(
33              "Rejecting incoming message since unable to submit",
34              SMPPConstant.STAT_ESME_RX_T_APPN, e);
35      }
36      }
```

Listing 12.11: SendChannel setReceiveCallback Method Implementation

12.3.3 Synchronous

Such a channel will implement **SynchronousReceiveChannel** when you need to communicate to the SMS/USSD gateway in a synchronous fashion: The gateway sends a request to the channel, the channels forwards the message to Brand Mobiliser, receives the message back, and responds synchronously to the gateway. These kind of channels only support MO (=Mobile Originated (user initiated)) sessions. Push messages are not supported for this channel.

If a gateway supports synchronous messages and push messages too, two channels must be registered with the ChannelManager, one only handling the push messages, the other only the synchronous messages.

The synchronization of request and response in ChannelManager is managed internally. The current default wait time is 60 seconds, but is configurable through the preferences. After that an empty object is returned. Each channel may also choose how long it actually waits for the response, if for example, the protocol itself places futher restrictions on how long the connection may remain open.

Synchronous channels will implement **supports** in the same manner as asynchronous channels.

setReceiveCallback

```
1       /**
2        * Set the receive callback through which this channel can push out incoming
3        * messages to some consumer and receive a response.
4        *
5        * @param receiveCallback
6        *             the receive callback
7        */
```

```
8       void setReceiveCallback(
9           final SynchronousChannelReceiveCallback receiveCallback);
```

Listing 12.12: SynchronousChannel setReceiveCallback Method

This will basically be a simple setter method for your channel, but the callback passed as an argument will be used by your channel to synchronously receive incoming messages. Typically your channel will listen for incoming messages and then use this reference to submit the message for processing and then wait for the response which then gets sent back to the client.

```
1       @Override
2       public void setReceiveCallback(
3           final SynchronousChannelReceiveCallback receiveCallback) {
4       this.receiveCallback = receiveCallback;
5       }
6
7
8       @Override
9       public void processRequest(final HttpServletRequest request,
10          final HttpServletResponse response) throws ServletException,
11          IOException {
12
13      LOG.debug("Incoming {} request", request.getMethod());
14
15      checkAndPrepare(request, response, false);
16
17      final MultiValueMap<String, String> result = (MultiValueMap<String, String>)
            this.converter
18          .read(null, new ServletServerHttpRequest(request));
19
20      final List<String> textList = result.get("text");
21      final List<String> fromList = result.get("from");
22      final List<String> toList = result.get("to");
23      final List<String> typeList = result.get("type");
24
25      if (textList == null || textList.isEmpty()) {
26          throw new MissingServletRequestParameterException("text", "string");
27      }
28
29      if (fromList == null || fromList.isEmpty()) {
30          throw new MissingServletRequestParameterException("from", "string");
31      }
32
33      if (toList == null || toList.isEmpty()) {
34          throw new MissingServletRequestParameterException("to", "string");
35      }
36
37      final String type;
38      if (typeList.isEmpty()) {
39          type = "sms";
40      } else {
41          type = typeList.get(0);
42      }
43
44      final Message req = this.messagingEngine.parseSimpleTextMessage(type,
45          textList.get(0));
46      req.setSender(fromList.get(0));
47      req.setRecipient(toList.get(0));
48
49      if (LOG.isDebugEnabled()) {
50          LOG.debug("{} message received for {} from {}", new Object[] {
51              type, req.getRecipient(), req.getSender() });
52      }
53
```

```
54      final Future<Message> responseMessage = this.receiveCallback
55          .receiveAndRespondMessage(req, this.channelId,
56              this.incomingChannelId);
57
58      if (LOG.isDebugEnabled()) {
59          LOG.debug("Handed off message to {} for {} awaiting response",
60              this.receiveCallback, this.incomingChannelId);
61      }
62
63      final SmsMessage message;
64      try {
65          message = (SmsMessage) responseMessage.get();
66
67          if (message == null) {
68          LOG.warn("Timed out waiting for response from {}",
69              responseMessage);
70
71          throw new NestedServletException(
72              "Timed out waiting for message");
73          }
74      } catch (final InterruptedException e) {
75          Thread.currentThread().interrupt(); // reset flag
76
77          throw new NestedServletException("Interrupted during processing", e);
78
79      } catch (final ExecutionException e) {
80          if (e.getCause() instanceof InterruptedException) {
81          throw new NestedServletException(
82              "Interrupted during processing", e.getCause());
83          }
84
85          throw new NestedServletException("Processing message failed",
86              e.getCause());
87      }
88
89      LOG.debug("Writing response back to client");
90
91      final LinkedMultiValueMap<String, Object> responseMap = new
            LinkedMultiValueMap<String, Object>();
92
93      responseMap.add("from", message.getSender().getAddress());
94      responseMap.add("to", message.getRecipient().getAddress());
95      responseMap.add("text", new String(message.getText().getContent(),
96          message.getText().getCharset()));
97
98      this.converter.write(responseMap, this.mediaType,
99          new ServletServerHttpResponse(response));
100
101     }
```

Listing 12.13: SynchronousChannel setReceiveCallback Method Implementation which is also an HTTP Channel

Chapter 13

Report

The Mobiliser uses the “Crystal Reports Framework” for generation and display of reports. Reports are automatically discovered from the hard disk and available through the Mobiliser reporting API as well as through the Mobiliser front-end.

This chapter describes how to add new report templates with the report designer, report deployment and security. See “Mobiliser Framework Architecture and Design” guide for more information on the report architecture and its components.

13.1 Report creation

Reports are created or modified by using Crystal Reports Designer to create or modify a .rpt file. The Crystal Reports report (.rpt) file specifies the data source,query and data presentation information. These rpt files are binary only and are created/updated using the Crystal Reports Designer product. As the JRC component only supports a subset of the full Crystal Reports functionality, it is recommended to use Crystal Reports Designer 2008 for maximum compatibility.

Note: There are a number of standard reports included in the standard money product you can use to kickstart your report development by simply customizing the parameters,queries and design inside.

1. Open the Crystal Reports designer
2. "New" -> "Standard Report"
3. Expand "Create new Connection" and expand the type of connection you want to add (e.g. JDBC)
4. DblClick "Make new Connection" and enter the connection parameters.
5. DblClick "Add Command" to enter your SQL script.
   (a) If the sql requires parameters you can create params by clicking "Create..." the param will then appear in the "Parameter List". You can add the param to your query by double clicking the parameter in the "Parameter List".
   (b) If you click next your query will be validated and you will be asked for values for the params you defined.
6. Your query will be executed and the result will be shown on the next screen of the wizard, here you can select which fields of the resultset you want to use in your report.
7. You have additional options regarding grouping of the results, design templates etc. when clicking "next", alternatively click "finish" to use the fields you selected.
8. If your query had any results and you selected fields from the query in the "Report wizard" the values will be shown instantly in the "Preview" view. To realign and design the report click "Design" in the left upper corner of the preview pane.

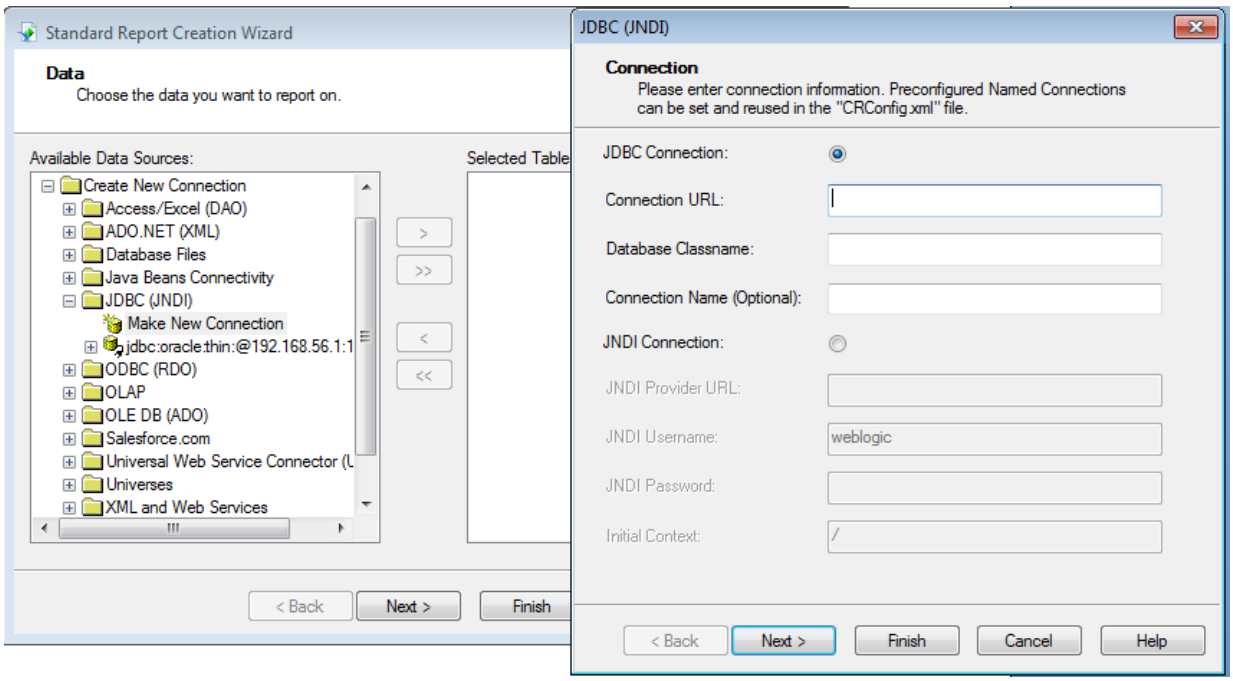

Figure 13.1: The Crystal Reports 2008 Designer Database configuration wizard

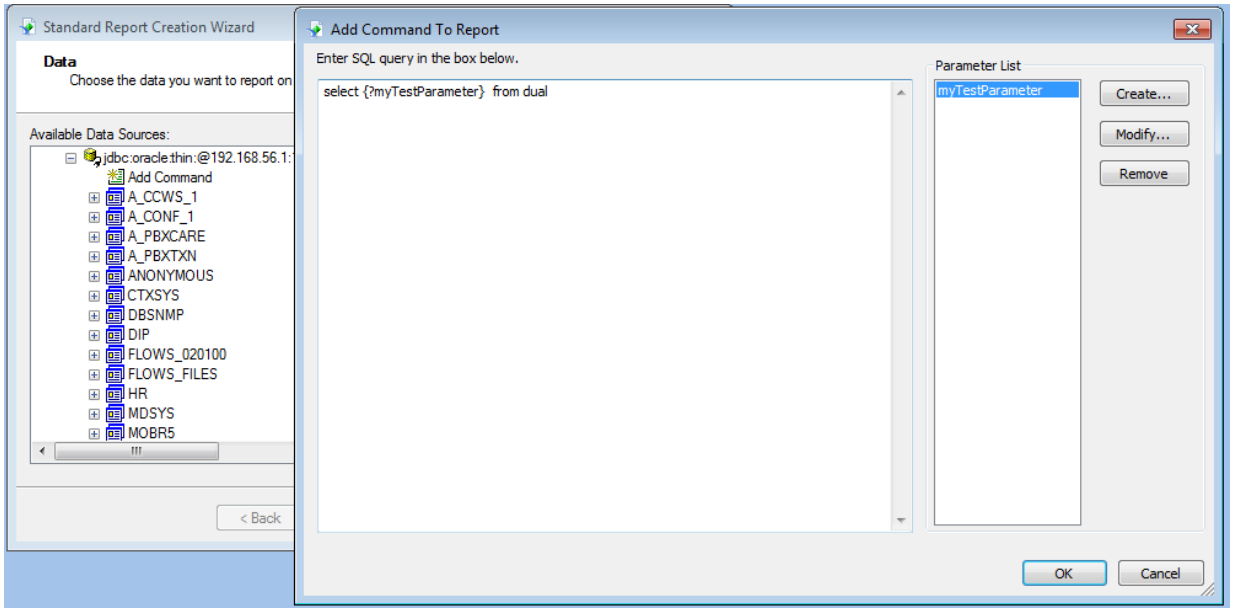

Figure 13.2: The Crystal Reports 2008 Designer Database configuration wizard

Note: You can see all available fields and parameters (as well as define new ones and drag and drop existing ones into the report) in the "Field Explorer" pane on the right.

Reports with Dates and Times: Crystal Reports has poor support for multi time zone reports. We have added a few crutches to make it halfway work. First you should consider that your report will most likely be requested for a time zone other than the database time zone, since our recommendation is to run Mobiliser in UTC.

Crystal Reports have a function ShiftDateTime(timestamp, fromTz, toTZ). Of course it does not use standard Olson database ID but completely home baked solution. Here is the breakdown, best explained with an exam-

ple.

Taking the TZ ID Pacific/Auckland, the correct format would be:

New Zealand Standard Time,-720,New Zealand Daylight Time,-60;9.5.0/02:00,4.1.0/03:00

The fields are as follows:

- name / description of the standard time zone
- standard time offset in minutes (meaning how many minutes to add to local time to get UTC time. Here is is negative since we are west of UTC)
- offset between standard time and daylight saving time (so how many minutes to add to daylight time to get standard local time)
- start of daylight saving time (MM.W.dd/HH:mm - so the month of the year, the week of the month, the day of the week, where Sunday is day 0 and the hour of the day)
- end of daylight saving time - same format as above

As you can see, this format will not work if you request a reports across two daylight periods in a year where the rules have changed. But we must make do with what we have.

The crystal reports SDK includes a utility to convert between Olson IDs and this internal format:

com.crystaldecisions.reports.common.TimeZoneUtil.getTimeZoneString(TimeZone)

It does seem to have its bugs though. For the example above it returns a correct string but reports generated during daylight saving time are off by an hour (The actual time range sent to the database is correct, the formatted dates in the report are just displayed incorrectly). I am guessing that probably has something to do with that timezone being in the southern hemisphere, but without source code I cannot say for sure.

Make sure when displaying run dates or date ranges in your report header that you always use this function, otherwise they will be shown in the local time of the back end server (most likely UTC) definitely leading to user complaints.

Helpful parameters you may use in your reports, which will not be requested from the report generator himself, but added automatically by the framework:

_PARAM_USER_TIMEZONE Will be filled with the crystal reports proprietary time zone format based on the agent's time zone

_PARAM_RAW_USER_TIMEZONE Will be filled with the raw agent's time zone, so America/Los_Angeles for example

_PARAM_SERVER_TIMEZONE Will be filled with the crystal reports proprietary time zone format based on Mobiliser's time zone. This requires that Mobiliser and the database be configured with the same time zone which we have set as a requirement elsewhere

_MIDNIGHT_USER_TIMEZONE To allow keeping the reports generic across all database variants, you can use this parameter to allow getting the value of midnight of the current day in the user's time zone (For example in reports where you need to aggregate something for the current day or X days in the past). So for an agent with TZ America/Los_Angeles, the result for a value like 1328256000 would be: assuming the database is running in UTC will be set for our TIMESTAMP columns as '2012-02-03 08:00:00'. This solves the problem of converting the input value using SQL on databases like DB2 and ASE where the time zone support is lacking.

Note: For details on the usage of the CR2008 Designer please refer to its user manual.

13.2 Security

Caution! Crystal Reports is NOT SQL-Injection safe, it only validates the datatype of the parameters so you must be very cautious when using "text" datatypes such as String as crystal reports will simply append the param to the query.

13.3 Deploying Reports:

To make the reports available to the users, the rpt file you have created must be placed in the

<mobiliser-installation>/reports/live

directory (or the path configured for the reportfiles), remember this during configuration of the packaging. If you want to deploy your report on the fly, simply drop it into the mentioned folder, since the Reportwatcher component scans the directory for new Report files regularly, the report defined by the file should be available for creation almost instantly.

Chapter 14

Utilities

This chapter lists the Mobiliser utility modules, services, and packages, which can be used when implementing on top of Mobiliser. The utility packages provides convenient methods for common tasks, such as formatting, encryption, . . . The usage of these packages also ensures unique implementation and single point of changes.

14.1 Framework Services

14.1.1 Interface ICustomerOtpLogic

Class: com.sybase365.mobiliser.money.businesslogic. customer.ICustomerOtpLogic

Overview

Service to create and verify One-Time-Passwords for customers. OTPs can be used for registration to check the entered MSISDN of a customer and are used for example as authentication in a financial transaction. OTPs can also be used for other purposes. New OTP types must first be configured in the database.

An implementation of this class is exported via the OSGi service registry.

Use this to import the service into your code:

```
<osgi:reference id="customerOtpLogicBean" interface="com.sybase365.mobiliser.money
businesslogic.  customer.ICustomerOtpLogic" availability="optional" sticky="false" />
```

Methods

createCustomerOtp Creates a new CustomerOtp in the database. The OTP itself is either automatically generated according to the rules defined with the OTP type or the provided otp is used. The OTP is sent out in an email or SMS to the customer. A list of parameters can be provided to further customise the message. The key for the otp parameter is "otp". A SecureRandom is used to generate the OTP.

Parameters

customerId for whom to create the OTP

otp if this is provided it will be used instead of an automatic generated one (optional). It needs to be numeric in case the OTP type is configured to be numeric, otherwise an `OtpValidation Exception` is thrown.

otpType the type of OTP to generate/send, defines length, message template, and send mode (email/SMS) (optional)

referenceId is stored with the CustomerOtp and can be retrieved when validating the OTP. This can be for example a transaction id in case the OTP is issued for a specific transaction

msgParams List of message parameters that is appended to the message that is sent out to the customer. Those can be referenced in the message template (which is configured for the OTP type)

callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

Throws

OtpValidationException

Returns the newly created CustomerOtp

createCustomerOtp Variation of `createCustomerOtp` that does not contain the otp and the list of message parameters in the list of parameters.

Parameters

customerId for whom to create the OTP

otpType the type of OTP to generate/send, defines length, message template, and send mode (email/SMS) (optional)

referenceId is stored with the CustomerOtp and can be retrieved when validating the OTP. This can be for example a transaction id in case the OTP is issued for a specific transaction

callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

Throws

OtpValidationException

EntityNotFoundException

Returns the newly created CustomerOtp

createCustomerOtp Variation of `createCustomerOtp` that does not contain the list of message parameters in the list of parameters.

Parameters

customerId for whom to create the OTP

otp if this is provided it will be used instead of an automatic generated one (optional). It needs to be numeric in case the OTP type is configured to be numeric, otherwise an `OtpValidation Exception` is thrown.

otpType the type of OTP to generate/send, defines length, message template, and send mode (email/SMS) (optional)

referenceId is stored with the CustomerOtp and can be retrieved when validating the OTP. This can be for example a transaction id in case the OTP is issued for a specific transaction

callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

Throws

OtpValidationException

Returns the newly created CustomerOtp

createCustomerOtp Variation of `createCustomerOtp` that does not contain the otp in the list of parameters.

Parameters

customerId for whom to create the OTP

otpType the type of OTP to generate/send, defines length, message template, and send mode (email/SMS) (optional)

referenceId is stored with the CustomerOtp and can be retrieved when validating the OTP. This can be for example a transaction id in case the OTP is issued for a specific transaction

msgParams List of message parameters that is appended to the message that is sent out to the customer. Those can be referenced in the message template (which is configured for the OTP type)

callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

Throws

OtpValidationException

EntityNotFoundException

Returns the newly created CustomerOtp

createNonPersitentOtp Create a new OTP. The OTP is returned as a String. The OTP is not stored and not send out. A SecureRandom is used to generate the OTP.

Parameters

otpType the type of OTP that is to be generated.

Throws

OtpValidationException

Returns the generated OTP.

verifyCustomerOtp Verifies if the provided OTP matches the one stored in the DB. In case the OTP in the DB already expired, was not found, or the threshold of failed verifications has been exceeded an `OtpValidationException` is thrown.

Parameters

otp the OTP to validate

customerId the customer Id for whom to validate hte OTP

otpType the type of OTP to validate

referenceId the reference Id as provided when the OTP was created

callerId the ID of the caller of the operation. Used for tracking/audit purposes.

Throws

OtpValidationException is thrown if no OTP was found in the DB, the OTP in the DB has expired, the threshold of failed verifications has been exceeded.

Returns `true` when the otp matched, `false` when the otp did not match.

14.1.2 Interface INotificationLogic

Class: com.sybase365.mobiliser.money.businesslogic. customer.ICustomer.OtpLogic

Overview

This service provides common helper methods used when sending notifications to a customer, like retrieving the preferred MSISDN or email address based on the customer configuration.

An implementation of this class is exported via the OSGi service registry.

Use this to import the service into your code:

```
<osgi:reference id="notificationLogic" interface="com.sybase365.mobiliser.money.
businesslogic.  customer.INotificationLogic" sticky="false" />
```

Methods

getCustomersMsisdn Retrieve a list of MSISDNs sorted from the most to least preferred. The call returns an empty list if no MSISDN is available. The default implementation will return a maximum of one MSISDN. It only returns MSISDNs that are registered as Identifications.

Parameters

customer for whom to retrieve the MSISDNs

actor who has invoked the service (for tracking purposes)

Returns a list of MSISDN sorted from most to least preferred.

getCustomersNotificationMsisdn Same as `getCustomersMsisdn` but will also require that the Notification mode of the customer indicates that he/she wants to receive SMS notifications. If this is not the case an empty list is returned.

Parameters

customer for whom to retrieve the MSISDNs

actor who has invoked the service (for tracking purposes)

Returns a list of MSISDN sorted from most to least preferred.

getCustomersEmail Retrieve a list of email addresses sorted from the most to least preferred. The call returns an empty list if no email address is available. The default implementation only returns one email address which is taken from the Customer's main address

Parameters

customer for whom to retrieve the email addresses

actor who has invoked the service (for tracking purposes)

Returns a list of email addresses sorted from most to least preferred.

getCustomersNotificationEmail Same as `getCustomersEmail` but will also require that the Notification mode of the customer indicates that he/she wants to receive email notifications. If this is not the case an empty list is returned.

Parameters

customer for whom to retrieve the email addresses

actor who has invoked the service (for tracking purposes)

Returns a list of email addresses sorted from most to least preferred.

14.1.3 Interface com.sybase365.mobiliser.money.businesslogic. umgr.ISecurityLogic

Overview

Set of services that deal with customer authentication (login), session handling.

An implementation of this class is exported via the OSGi service registry.

Use this to import the service into your code:

```
<osgi:reference id="securityLogic" interface="com.sybase365.mobiliser.money.businesslogic.
umgr.ISecurityLogic" sticky="false" />
```

Methods

loginCustomer

Logs in an end-user into an application by checking username, credential, and creating a new session. This method runs in its own transactionto make sure that wrong credential checks are persisted. The session id that is returned should be used for each subsequent request and is valid either until it times out or a logout is called.

In case of an incorrect credential the Credential Policies are used to determine the next action. In case the credential was correct but it is temporary (system generated) or has expired the `CredentialOkButExpired Exception` is thrown.

Parameters

identification the identification to load the customer by (MSISDN, username, ...).

credential the credential to check (PIN, password, ...).

identificationType the type of the identification provided in `identification`.

credentialType the type of the credential provided in `credential` .

origin the origin is stored with the session that is created. Specifies where the login operation is initiating from.

callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

Throws

EntityNotFoundException
CustomerInactiveException
CustomerBlacklistedException
CredentialBlockedException
CredentialOkButExpiredException
CredentialWrongException
CredentialNotFoundException

Returns the created CustomerSession

checkCredential Similar to loginCustomer but does not create a session of the customer upon success. Instead of an Identification the id of the customer must be provided. Other than that the same business logic is executed.

Parameters

customerId the ID of the customer for whom the credential is checked
credential the credential to check
credentialType the type of the credential that is provided
callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

Throws

EntityNotFoundException
CustomerInactiveException
CustomerBlacklistedException
CredentialBlockedException
CredentialOkButExpiredException
CredentialWrongException
CredentialNotFoundException
CredentialOkButTemporaryException
CredentialWrongAndActionTakenException

replaceCredential Replaces an existing credential of the same type with the supplied one. The old one is marked as in-active and the new one is saved. The credential must be supplied as plain text and will be stored hashed. The credential is NOT checked against the policy. This must be done before the call to this method via `checkCredentialStrength`

Parameters

customerId the ID of the customer for whom the credential will be stored
credential the credential to store
credentialType the type of the credential that is provided
credentialStatus specifies the status of the credential that is to be stored (usually 0 is used).
callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

replaceCredential Replaces an existing credential of the same type with the supplied one. The old one is marked as in-active and the new one is saved. The credential must already be provided in hashed format. This method is mainly called internally.

Parameters

credential a pre-hashed credential
callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

logoutCustomer Closes the session that was created by `loginCustomer`.

Parameters

sessionId the id of the session that is to be closed
callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

Throws

EntityNotFoundException if the provided sessionId was not found.

getAllPrivileges Returns all privileges that are attached to a customer, either via CustomerType, RiskCategory, Roles, or directly. Does not take the OrgUnit into account.

Parameters

customerId the ID of the customer for whom to retrieve the privileges.

callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

Throws

EntityNotFoundException

CustomerInactiveException

Returns the list of privileges (`Strings`)

getDefaultPrivileges Returns all privileges that are attached to the role of the CustomerType and RiskCategory. If the provided riskCategoryId is null, the default one from the CustomerType is used. This can be used to check the privileges before a new user gets created.

Parameters

customerTypeId

riskCategoryId

callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

Throws

EntityNotFoundException

Returns generateCredential Generates a new credential that is in compliance with the provided policy.

Parameters

policy the CredentialPolicy that the credential must adhere to.

numericOnly indicates if the credential should only contain numbers.

callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

Throws

CredentialGenerationException if it was not possible to generate a credential that is compliant to the policy. Probably means that the policy is wrongly configured.

Returns the generated credential in plain text

checkCredentialStrength Checks if the credential matches all of the requirements defined in the given policy. In case it does not an appropriate exception is thrown.

Parameters

policy the policy that is used for checking the credntial against

plainTextCredential the credential in plain text (not hashed)

callerId represents the ID of the caller of the operation. Used for tracking/audit purposes.

Throws

CredentialTooLongException

CredentialTooShortException

CredentialHasWeakBlockException

CredentialHasInvalidPatternException

checkCredentialRetention Checks if the credential matches any retained credential. Mainly used internally.

Parameters

customer

credentialType

plainTextCredential

handleWrongCredentialInput Handles wrong credential input (update counter, black-list customer, ...) based on the settings in the security policy. Mainly used internally.

Parameters

cred

callerId

Returns `true` if the number of wrong entries was exceeded and the customer was either black-listed or the credential is temporarily blocked. Returns `false` otherwise.

handleCorrectCredentialInput Handles the correct entry of a the credential (resets wrong-entry counter and alike). Mainly used internally.

Parameters

cred

callerId

14.2 Bully

The bully mediator can be used by applications that access a single resource which may only be accessed sequentially. The new bully can be found in com.sybase365.mobiliser.util.tools:com.sybase365.mobiliser.util.tools.bully.

14.2.1 API

The IBullyMediatorFactory interface provides the central entry point through which clients can retrieve a pre-configured bully.

```
1  public interface IBullyMediatorFactory {
2
3      /**
4       * Returns a new {@link IBullyMediator} for the given service.
5       *
6       * @param service
7       *            the service
8       * @return the bully mediator
9       */
10     IBullyMediator getBullyMediator(final String service);
11 }
```

Listing 14.1: IBullyMediatorFactory interface

The IBullyMediator interface defines the actual bully.

```
1  public interface IBullyMediator {
2
3      /**
4       * <p>
5       * Returns the service value of this instance.
6       * </p>
7
8       *
9       * @return service the <tt>service</tt> value
10      */
11     String getService();
12
13     /**
14      * <p>
```

```
15      * Checks if this instance is responsible for executing a service.
16      * Automatically extends the lease if responsible.
17      * </p>
18      *
19      * @return <code>true</code> if this instance is "the bully";
20      *         <code>false</code> otherwise
21      */
22     boolean isResponsible();
23
24     /**
25      * Releases the lock for this application. Should be called when shutting
26      * down the bullying application.
27      */
28     void release();
29 }
```

Listing 14.2: IBullyMediator interface

14.2.2 Databases

The standard bully supports Oracle, DB2, and H2 databases. Configuration for other database types can easily be plugged in.

The statement factory allows the BullyFactory to return bullies which are unaware of the underlying database. The database type is looked up using the JDBC metadata.

An example statement factory configuration is shown in listing 14.3. The bean id is used for the look up based on the meta-data.

```
1  <beans>
2    <bean id="Oracle"
3      class="com.sybase365.mobiliser.util.tools.bully.impl.BullyStatement">
4      <property name="checkStatement">
5        <value>
6          <![CDATA[
7          SELECT
8              STR\_Worker,
9              SIGN(cast(DAT\_Lease\_End as date) - SYSDATE)
10         FROM
11             Mob\_Bully
12         WHERE
13             STR\_Service = ?
14             AND BOL\_Is\_Active = 'Y'
15         FOR UPDATE OF STR\_Worker, DAT\_Lease\_End WAIT 1
16         ]]>
17       </value>
18     </property>
19     <property name="updateStatement">
20       <value>
21         <![CDATA[
22         UPDATE
23             Mob\_Bully
24         SET
25             STR\_Worker = ?,
26             DAT\_Lease\_End = SYSDATE + (1 / 24 / 60 / 60 * INT\_Lease\_Term)
27         WHERE
28             STR\_Service = ?
29         ]]>
30       </value>
31     </property>
32     <property name="releaseStatement">
33       <value>
34         <![CDATA[
```

```
35          UPDATE
36              Mob\_Bully
37          SET
38              STR\_Worker = ?,
39              DAT\_Lease\_End = SYSDATE
40          WHERE
41              STR\_Service = ?
42          ]]>
43        </value>
44      </property>
45    </bean>
46 </beans>
```

Listing 14.3: Bully statement configuration

The Spring configuration defines the statement factory to get the bully statements for the database back-end.

```
1   <bean id="statementFactory"
2     class="com.sybase365.mobiliser.util.tools.bully.impl.BullyStatementFactory" /
        >
```

Listing 14.4: BullyStatementFactory configuration

A custom database configuration can be explicitly defined.

```
1   <bean id="statementFactory"
2     class="com.sybase365.mobiliser.util.tools.bully.impl.BullyStatementFactory">
3     <constructor-arg>
4       <bean class="org.springframework.core.io.ClassPathResource">
5         <constructor-arg value="/com/sybase365/mobiliser/blabb/bully-statements.
              xml" />
6       </bean>
7     </constructor-arg>
8   </bean>
```

Listing 14.5: Custom BullyStatementFactory configuration

The bully factory and the actual bully are also configured through Spring

```
1   <bean id="bullyFactory"
2     class="com.sybase365.mobiliser.util.tools.bully.impl.BullyMediatorFactory">
3     <property name="dataSource" ref="dataSource" />
4     <property name="jdbcTemplate" ref="jdbcTemplate" />
5
6     <property name="statementFactory" ref="statementFactory" />
7     <property name="transactionManager" ref="transactionManager" />
8   </bean>
9
10  <bean id="bully"
11    class="com.sybase365.mobiliser.util.tools.bully.impl.BullyMediatorFactoryBean
        ">
12    <constructor-arg ref="bullyFactory" />
13    <constructor-arg ref="bullyId" />
14  </bean>
```

Listing 14.6: Bully configuration

14.3 Identification Normalizer

14.3.1 Overview

The `IdentificationNormalizer` takes an incoming identification and converts it to a unified format which the system will understand. For example msisdns may be converted to the 00xy international format from the +xy format. The Identification normalizer is invoked by mobiliser internally when doing things such as logins to check the identification used in a defined format, no matter in which format the identification is specified in the request.

14.3.2 Adding a custom Normalizer

To add a custom Normalizer you simply need to implement the com.sybase365.mobiliser.money.businesslogic.normalizer.IIdenti Interface.

Ideally you should set the coverage and order params via spring injection

```
1  <bean id="msisdnNormalizer"
2    class="com.sybase365.mobiliser.money.businesslogic.normalizer.impl.
       MsisdnIdentificationNormalizer">
3    <property name="coverage" value="0" />
4    <property name="order" value="10" />
5  </bean>
```

Listing 14.7: Bully configuration

Since the defaultNormalizer has a default coverage of 1 – 7 you need to have an “order” value greater than 10 if you want to add a normalizer for any of the identification types between 1 and 7 (inclusive).

You then need to register the Normalizer in the osgi registry. Simply add an entry to the bundle-context-osgi.xml.

```
1  <osgi:service ref="msisdnNormalizer" ranking="10">
2    <osgi:interfaces>
3      <value>com.sybase365.mobiliser.money.businesslogic.normalizer.
         IIdentificationValueNormaliser
4      </value>
5      <value>org.springframework.core.Ordered</value>
6    </osgi:interfaces>
7    <osgi:service-properties>
8      <beans:entry key="origin" value="mobiliser" />
9    </osgi:service-properties>
10 </osgi:service>
```

Listing 14.8: Bully configuration

14.4 Client Utils

The client utils classes allows easy service access for any client that accesses the Mobiliser platform services. It includes helper classes for rest and soap web services.

14.4.1 Class com.sybase365.mobiliser.util.tools.clientutils.api.Cookie

Overview

Constructors

Cookie

Parameters

cookieName
cookieValue

Methods

getCookieName

Returns the cookieName

getCookieValue

Returns the cookieValue

14.4.2 Interface com.sybase365.mobiliser.util.tools.clientutils.api.IClientConfiguration

Overview

Simple client configuration interface.

Methods

getMobiliserEndpointUrl Return the web service base url.

Returns the base url

getWsPassword Return the web service password.

Returns the password

getWsUserName Return the web service user name.

Returns the user name

14.4.3 Interface com.sybase365.mobiliser.util.tools.clientutils.api.IServiceClientFactory

Overview

Implementations of this interface can create service clients from generic business interfaces.

Methods

createClient Create a client for the webservice details passed to this method.

Parameters

interfaceClasses the endpoint interfacees for which a client should be created
endpointUrl the url of the endpoint

Returns the client for the web service

createClient Create a client for the webservice details passed to this method.

Parameters

interfaceClasses the endpoint interfaces for which a client should be created
endpointUrl the url of the endpoint
userName the user name
password the password

Returns the client for the web service

createClient Create a client for the webservice details passed to this method.

Parameters

interfaceClasses the endpoint interfaces for which a client should be created
endpointUrl the url of the endpoint
userName the user name
password the password
classLoader the classloader to use

Returns the client for the web service

createClient Create a client for the webservice details passed to this method.

Parameters

interfaceClasses the endpoint interfaces for which a client should be created
endpointUrl the url of the endpoint
userName the user name
password the password
classLoader the classloader to use
cookieConfiguration the cookie configuration for the client

Returns the client for the web service

createClient Create a client for the webservice details passed to this method.

Parameters

interfaceClass the endpoint interface for which a client should be created
endpointUrl the url of the endpoint

Returns the client for the web service

createClient Create a client for the webservice details passed to this method.

Parameters

interfaceClass the endpoint interface for which a client should be created
endpointUrl the url of the endpoint
userName the user name
password the password

Returns the client for the web service

createClient Create a client for the webservice details passed to this method.

Parameters

interfaceClass the endpoint interface for which a client should be created
endpointUrl the url of the endpoint
userName the user name
password the password
classLoader the classloader

Returns the client for the web service

createClient Create a client for the webservice details passed to this method.

Parameters

interfaceClass the endpoint interface for which a client should be created
endpointUrl the url of the endpoint
userName the user name
password the password
classLoader the classloader
cookieConfiguration cookie configuration

Returns the client for the web service

14.4.4 Class RefreshableClientTargetSource

Class: com.sybase365.mobiliser.util.tools.clientutils.api.RefreshableClientTargetSource

Overview

TargetSource returning the webservice clients which allows dynamic refreshing.

Constructors

RefreshableClientTargetSource

Methods

afterPropertiesSet

destroy

getLastRefreshTime

getRefreshCount

getTarget

getTargetClass

isStatic

refresh

releaseTarget

setBeanClassLoader

setClientFactory

Parameters

clientFactory the clientFactory to set

setClientInterface

Parameters

clientInterface the clientInterface to set

setClientInterfaces

Parameters

clientInterfaces the clientInterfaces to set

setConfiguration

Parameters

configuration the configuration to set

setCookieConfiguration

Parameters

cookieConfiguration the cookieConfiguration to set

setEndpointSuffix

Parameters

endpointSuffix the endpointSuffix to set

14.4.5 Interface com.sybase365.mobiliser.util.tools.clientutils.api.ICookieConfiguration

Overview

Cookie abstraction for web service clients.

Methods

getCookieRequestString Return the url string needed to request a cookie from the service.

Returns thec cookie string.

isUseCookies Whether to use cookies or not.

Returns true if cookies should be use, false otherwise.

pollCookie Get the current cookie, if available. NULL would indicate that a new cookie should be requested, based on `isUseCookies`.

Returns the current cookie

addCookies Callback to update the cookie after sending a response.

Parameters

newCookies cookies from response

14.4.6 Interface com.sybase365.mobiliser.util.tools.clientutils.api.impl.DummyClient

Overview

Dummy Client for some test cases.

Methods

doSomeCall

Parameters

str

Returns blah

14.4.7 Class AbstractClientFactoryTest

Class: com.sybase365.mobiliser.util.tools.clientutils.api.impl.AbstractClientFactoryTest

Overview

Test case for the class `AbstractClientFactory`.

Constructors

AbstractClientFactoryTest

Methods

setUp Setup test case.

testSimple Tests simple

Throws

Throwable

14.4.8 Class com.sybase365.mobiliser.util.tools.clientutils.api.impl.AbstractClientFactory

14.4.9 Class: com.sybase365.mobiliser.util.tools.clientutils.api.impl.AbstractClientFactory

Overview

Base implementation which creates a dynamic proxy which delegates creation of the method interceptor to the child class. This implementation proxies your service interface and sets up the interceptor to run based on the pointcut: **args(com.sybase365.mobiliser.framework.contract.v5_0.base. MobiliserRequestType)**

Constructors

AbstractClientFactory

Methods

createClient

See Also `#createClient(java.lang.Class[], java.lang.String)`

createClient

See Also `#createClient(java.lang.Class[], java.lang.String, java.lang.String, java.lang.String)`

createClient

See Also `#createClient(java.lang.Class[], java.lang.String, java.lang.String, java.lang.String, java.lang.ClassLoader)`

createClient

See Also `#createClient(java.lang.Class[], java.lang.String, java.lang.String, java.lang.String, java.lang.ClassLoader, com.sybase365 .mobiliser.util.tools.clientutils.api.ICookieConfiguration)`

createClient

See Also `createClient`

createClient

See Also `createClient`

createClient

See Also `createClient`

createClient

See Also `createClient`

createMethodInterceptor Delegates creation of the method interceptor to the child implementation.

Parameters

endpointUrl the endpoint url
userName the userName
password the password
classLoader
cookieConfiguration cookie configuration

Returns the interceptor

14.4.10 Enum com.sybase365.mobiliser.util.tools.clientutils.rest.ClientType

Overview

Enum for the different "rest"client types.

Enum Constants

JSON use json as the content type.

XML use xml as the context type.

Methods

values

valueOf

14.4.11 Class CookieAwareCommonsClientHttpRequestFactory

Class: com.sybase365.mobiliser.util.tools.clientutils.rest.CookieAwareCommonsClientHttpRequestFactory

Overview

Extends the standard .CommonsClientHttpRequestFactory to allow requesting cookies.

Constructors

CookieAwareCommonsClientHttpRequestFactory

Parameters

cookieConfiguration
httpClient

Methods

createRequest

See Also .org.springframework.http.client.CommonsClientHttpRequestFactory.createRequest(java.net.URI, org.springframework.http.HttpMethod)

14.4.12 Class com.sybase365.mobiliser.util.tools.clientutils.rest.RestClientFactory

Overview

This implementation makes use of Spring Web-MVC to create a `RestOperations` which is used by a proxy object to delegate method calls to the backend.

Constructors

RestClientFactory Constructs a new client for the given content type.

Parameters

clientType the client type.

RestClientFactory Constructs a new client for the given content type. Also allows configuring a hierarchy of types for the client.

Parameters

clientType the client type
parentType the parent type
childTypes children of the the parent type.

RestClientFactory Constructs a new client for the given content type. Also allows configuring a hierarchy of types for the client.

Parameters

clientType the client type
typeBuilderFactory the type builder factory to use.
parentType the parent type
childTypes children of the the parent type.

Methods

createMethodInterceptor

setAuthenticationPreemptive

Parameters

authenticationPreemptive the authenticationPreemptive to set

14.4.13 Class com.sybase365.mobiliser.util.tools.clientutils.soap.SoapClientFactory

Overview

This implementation makes use of Spring-WS to create a `WebServiceOperations` which is used by a proxy object to delegate method calls to the soap backend. You may configure a .SoapMessageFactory to use with this factory or a default one will be configured.

SoapClientFactory

SoapClientFactory

Parameters

soapMessageFactory

Methods

createMethodInterceptor

setAcceptGzipEncoding

Parameters

acceptGzipEncoding the acceptGzipEncoding to set

setAuthenticationPreemptive

Parameters

authenticationPreemptive the authenticationPreemptive to set

14.4.14 Interface SwappableMarshaller

Class: com.sybase365.mobiliser.util.tools.clientutils.soap.impl.SwappableMarshaller

Overview

Methods

swap Swaps the marshaller.

Parameters

marshaller the marshaller to set

remove Remove the marshaller.

14.4.15 Class DynamicJaxb2Marshaller

Class: com.sybase365.mobiliser.util.tools.clientutils.soap.impl.DynamicJaxb2Marshaller

Overview

Constructors

DynamicJaxb2Marshaller

Parameters

marshallers

Methods

marshal

See Also .org.springframework.oxm.Marshaller.marshal(java.lang.Object, javax.xml.transform.Result)

marshal

remove

supports

See Also .org.springframework.oxm.Marshaller.supports(java.lang.Class)

supports

swap

14.4.16 Class SoapMethodInterceptor

Class: com.sybase365.mobiliser.util.tools.clientutils.soap.impl.SoapMethodInterceptor

Overview

Simple method interceptor which delegates method calls to a configured .WebServiceOperations instance.

Constructors

SoapMethodInterceptor

Parameters

classLoader
jaxbMarshallers
marshallerSwap
template
unmarshallerSwap

Methods

getJaxbContext Returns a .Jaxb2Marshaller for the given class.

Parameters

clazz the class to return the context for

Throws

HttpMessageConversionException in case of JAXB errors

Returns the `Jaxb2Marshaller`

invoke

14.4.17 Interface SwappableUnmarshaller

Class com.sybase365.mobiliser.util.tools.clientutils.soap.impl.SwappableUnmarshaller

Overview

Methods

swap Swaps the unmarshaller.

Parameters

unmarshaller the unmarshaller to set

remove Remove the marshaller.

14.4.18 Class DynamicJaxb2Unmarshaller

Class: com.sybase365.mobiliser.util.tools.clientutils.soap.impl.DynamicJaxb2Unmarshaller

Overview

Constructors

DynamicJaxb2Unmarshaller

Parameters

unmarshallers

Methods

remove

supports

See Also .org.springframework.oxm.Marshaller.supports(java.lang.Class)

supports

swap

unmarshal

See Also .org.springframework.oxm.Unmarshaller.unmarshal(javax.xml.transform.Source)

unmarshal

14.4.19 Class CookieAwareCommonsHttpMessageSender

Class: com.sybase365.mobiliser.util.tools.clientutils.soap.impl.CookieAwareCommonsHttpMessageSender

Overview

Extends the standard .CommonsHttpMessageSender to allow requesting cookies.

Constructors

CookieAwareCommonsHttpMessageSender

Parameters

cookieConfiguration

Methods

createConnection

14.4.20 Class CookieExtractingSoapMessageFactory

Class: com.sybase365.mobiliser.util.tools.clientutils.soap.impl.CookieExtractingSoapMessageFactory

Overview

Constructors

CookieExtractingSoapMessageFactory

Parameters

cookieConfiguration
delegate

Methods

createWebServiceMessage

See Also .org.springframework.ws.soap.SoapMessageFactory.createWebServiceMessage()

createWebServiceMessage

See Also .org.springframework.ws.soap.SoapMessageFactory.createWebServiceMessage(java.io.InputStream)

setSoapVersion

See Also .org.springframework.ws.soap.SoapMessageFactory.setSoapVersion(org.springframework.ws.soap.SoapVersion)

14.5 Date and Calendar

The date and calendar utilities allows common access to dates and calendars.

14.5.1 Interface com.sybase365.mobiliser.util.tools.dateutils.ISystemClock

Overview

Interface to the system clock. The methods in the Java API are static and lead to non-deterministic code.

Methods

getCurrentDate Returns the current date.

Returns the current date.

getCurrentTimeMillis Returns the current time in milliseconds.

Returns the current time in milliseconds

14.5.2 Class com.sybase365.mobiliser.util.tools.dateutils.DateUtils

Overview

The `DateUtils` provide convenient methods related to the `Calendar` and `Date` classes.

Fields

DF SimpleDateFormat with the pattern *yyyy-MM-dd HH:mm:ss*

DF_COMPLETE SimpleDateFormat with the pattern *yyyy-MM-dd HH:mm:ss,SSS*

DF_INTERNATIONAL SimpleDateFormat with the pattern *yyyy-MM-dd HH:mm:ss,SSS z*

DF_HUMAN SimpleDateFormat with the pattern *dd.MM.yy HH:mm:ss,SSS*

DF_TIME SimpleDateFormat with the pattern *HH:mm:ss,SSS*

DF_DATE SimpleDateFormat with the pattern *yyyy-MM-dd*

Methods

copy

Creates a copy of the specified date. This method comes in handy to prevent exposing internal fields (aka "May expose internal representation by returning reference to mutable object"reported by FindBugs).

Parameters

date the `Date` to copy

Returns a `Date` copy; `null` if `date` is `null`

copy

Creates a copy of the specified timestamp.

Parameters

timestamp the `Timestamp` to copy

Returns a `Timestamp` copy; `null` if `date` is `null`

See Also `copy`

getGreatestDivergentFieldWithWeek

Returns the greatest calendar field that is divergent for the two specified calendars. This method checks these fields in the listed order:

1. `ERA`
2. `YEAR`
3. `MONTH`
4. `WEEK_OF_YEAR`
5. `DAY_OF_MONTH`
6. `HOUR_OF_DAY`
7. `MINUTE`
8. `SECOND`
9. `MILLISECOND`

Parameters

first the first `Calendar` to compare

second the second `Calendar` to compare

Returns the greatest divergent field; `null` if all fields equal each other

getGreatestDivergentField

Returns the greatest calendar field that is divergent for the two specified dates. Turns the specified dates into `GregorianCalendar` s and delegates the call to `getGreatestDivergentField`.

Parameters

first the first `Calendar` to compare

second the second `Calendar` to compare

Returns the divergent field; `null` if all fields equal each other

See Also `getGreatestDivergentField`

getGreatestDivergentFieldWithWeek

Returns the greatest calendar field that is divergent for the two specified dates. Turns the specified dates into `GregorianCalendar` s and delegates the call to `getGreatestDivergentFieldWithWeek`.

Parameters

first the first `Calendar` to compare

second the second `Calendar` to compare

Returns the divergent field; `null` if all fields equal each other

See Also `getGreatestDivergentFieldWithWeek`

getGreatestDivergentField

Returns the greatest calendar field that is divergent for the two specified calendars. This method checks these fields in the listed order:

1. `ERA`
2. `YEAR`
3. `MONTH`
4. `DAY_OF_MONTH`
5. `HOUR_OF_DAY`
6. `MINUTE`
7. `SECOND`

8. `MILLISECOND`

Parameters

first the first `Calendar` to compare

second the second `Calendar` to compare

Returns the greatest divergent field; `null` if all fields equal each other

14.5.3 Class com.sybase365.mobiliser.util.tools.dateutils.impl.JdkSystemClockImpl

Overview

Default implementation of the `ISystemClock` interface. Uses the static methods in the jdk or creates new objects.

Constructors

JdkSystemClockImpl

Methods

getCurrentDate

See Also `getCurrentDate`

getCurrentTimeMillis

See Also `getCurrentTimeMillis`

14.6 Encryption

The encryption utility classes allow easy an common access to the Java crypto API. It also unify the processing. Please use these classes rather than coding you own access.

14.6.1 Class com.sybase365.mobiliser.util.tools.encryptionutils.TripleDESUtils

Overview

The `SecurityUtils` provide convenient methods related to encryption & decryption.

Fields

DES The *DES* cipher algorithm.

TRIPLEDES XXX-Missing-XXX

Constructors

TripleDESUtils

Methods

create3DesSymmetricKey Convenience method to create a 3DES secret key. it s padded with spaces when the key is not long enough.

Parameters

key

Throws

InvalidKeyException
EncryptionException

Returns secret key

create3DesSymmetricKey Creates a SecretKey from the provided byte array. The byte array must contain exactly 24 bytes.

Parameters

key

Throws

InvalidKeyException
EncryptionException

Returns the Proper SecretKey

encrypt3Des Encrypts the given password with the given key and returns it in BASE64 format.

Parameters

password
key

Throws

InvalidKeyException
EncryptionException

Returns encrypted password

decrypt3Des Decrypts the given bytes using the given key and returns the bytes.

Parameters

encryptedBytes
key

Throws

InvalidKeyException
EncryptionException

Returns decrypted bytes

decrypt3Des Decrypts the given BASE64 encoded password using the given key and returns the password in clear text.

Parameters

encryptedPassword
key

Throws

InvalidKeyException
EncryptionException

Returns decrypted password

14.6.2 Class com.sybase365.mobiliser.util.tools.encryptionutils.AES256Utils

Overview

The `AESUtils` provide convenient methods related to AES encryption & decryption.

This is bad design with static methods and also with static member fields.

Fields

ENCRYPTED_BYTES_INDEX index of the encrypted bytes.

IV_INDEX index of the initialisation vector.

Constructors

AES256Utils

Methods

createAesSymmetricKey Convenience method to create an AES secret key. It is padded with spaces when the key is not long enough.

Parameters

key

Throws

EncryptionException

Returns secret key

decryptAes Decrypts the given BASE64 encoded ciphertext using the given key and initialisation vector and returns the clear text.

Parameters

cipherText to decrypt
initialisationVector to use
key encryption key

Throws

EncryptionException

Returns clear text

decryptAes Decrypts the given BASE64 encoded ciphertext using the given key and initialisation vector and returns the clear text.

Parameters

cipherText to decrypt
initialisationVector to use
key encryption key

Throws

EncryptionException

Returns clear text

decryptAesRaw Decrypts the given encrypted bytes using the given initialisation vector and key and returns the clear text.

Parameters

encryptedBytes to decrypt

initialisationVector the initialisation vector

key encryption key

Throws

EncryptionException

Returns clear text

decryptAesRaw Decrypts the given encrypted bytes using the given initialisation vector and key and returns the clear text.

Parameters

encryptedBytes to decrypt

initialisationVector the initialisation vector

key encryption key

Throws

EncryptionException

Returns clear text

decryptAesSingle Decrypts the given encoded ciphertext with embedded initialisation vector using the given key and returns the clear text.

Parameters

cipherText to decrypt

key encryption key

Throws

EncryptionException

Returns clear text

decryptAesSingle Decrypts the given encoded ciphertext with embedded initialisation vector using the given key and returns the clear text.

Parameters

cipherText to decrypt

key encryption key

Throws

EncryptionException

Returns clear text

encryptAes Encrypts the given password with the given key and returns it in BASE64 format. Index 0 will hold the random IV, base64 encoded, index 1 the encrypted text, base64 encoded.

Parameters

clearText the input data to encrypt

key the key

Throws

EncryptionException

Returns encrypted password + initialisation vector

encryptAes Encrypts the given password with the given key and returns it in BASE64 format. Index 0 will hold the random IV, base64 encoded, index 1 the encrypted text, base64 encoded.

Parameters

clearText the input data to encrypt

key the key

Throws

EncryptionException

Returns encrypted password + initialisation vector

encryptAesRaw Encrypts the given password with the given key and returns the encrypted bytes. Index 0 will hold the encrypted bytes, index 1 the random IV.

Parameters

clearText the input data to encrypt

key the key

Throws

EncryptionException

Returns encrypted password + initialisation vector

encryptAesRaw Encrypts the given password with the given key and returns the encrypted bytes. Index 0 will hold the random IV, index 1 the encrypted bytes.

Parameters

clearText the input data to encrypt

key the key

Throws

EncryptionException

Returns encrypted password + initialisation vector

encryptAesSingle Encrypts the given password with the given key and returns it as a single string in BASE64 format. The first 16 bytes are the initialisation vector.

Parameters

clearText the input data to encrypt

key the key

Throws

EncryptionException

Returns initialisation vector + encrypted password

encryptAesSingle Encrypts the given password with the given key and returns it as a single string in BASE64 format. The first 16 bytes are the initialisation vector.

Parameters

clearText the input data to encrypt

key the key

Throws

EncryptionException

Returns initialisation vector + encrypted password

14.6.3 Exception EncryptionException

com.sybase365.mobiliser.util.tools.encryptionutils.EncryptionException

Overview

Thrown for encryption related problems.

Constructors

EncryptionException

Parameters

message

t

EncryptionException

Parameters

message

14.6.4 Class com.sybase365.mobiliser.util.tools.encryptionutils.AsymmetricKeyUtils

Overview

Some utilities to work with asymmetric keys for encryption and decryption.

generate new key pair into new or existing keystore (valid for 20 years)

```
>keytool -genkey -validity 7305 -keystore mob.ks -alias mobiliser -storepass changeit
-keypass geheim -keyalg RSA -dname "CN=Mobiliser Platform, OU=System, O=Sybase,
L=Raunheim, S=Hessen, C=DE"
```

export public key

```
>keytool -export -alias mobiliser -keystore mob.ks -rfc -storepass changeit -file pub.cer
```

import public key into new or existing keystore

```
>keytool -import -alias mobiliser -file pub.cer -keystore pub.ks -storepass changeit
```

Constructors

AsymmetricKeyUtils

Methods

encrypt

Encrypts the given text with the PublicKey. The returned String is a BASE64 encoded version of the encrypted bytes.

Parameters

text

key either `PrivateKey` to sign or `PublicKey` to encrypt the text

Throws

EncryptionException in case any keystore or key related error occurs

Returns Base64 encoded version of encrypted input

decrypt Decrypts the given text. Before decryption a Base64 decoding is dobne to retrieve the bytes to be decoded.

Parameters

text

key either `PrivateKey` to decrypt or `PublicKey` to verify a signature

Throws

EncryptionException

Returns The String represantation of the decrypted input

getPublicKey

Retrieves the PublicKey of a Certificate from a JKSE keystore. First attempt is to load the keystore from the classpath with the class loader. If this fails it is tried to be retrieved from the file system.

Parameters

keyStoreName
keyStorePassword
keyAlias

Throws

KeyStoreException

@deprecated please use `AsymmetricKeyUtils`

Returns

getPublicKey

Retrieves the PublicKey of a Certificate from a keystore. First attempt is to load the keystore from the classpath with the class loader. If this fails it is tried to be retrieved from the file system.

Parameters

keyStoreName
keyStorePassword
keyAlias
keyStoreType

Throws

KeyStoreException in case an error occurred

Returns the PublicKey

@since Mobiliser 5.0

getCertificate

Retrieves the Certificate from a JKSE keystore. First attempt is to load the keystore from the classpath with the class loader. If this fails it is tried to be retrieved from the file system.

Parameters

keyStoreName
keyStorePassword
keyAlias

Throws

KeyStoreException in case an error occurred

@deprecated please use `AsymmetricKeyUtils`

Returns the PublicKey

getCertificate

Retrieves the Certificate from a keystore. First attempt is to load the keystore from the classpath with the class loader. If this fails it is tried to be retrieved from the file system.

Parameters

keyStoreName
keyStorePassword
keyAlias
keyStoreType

Throws

KeyStoreException in case an error occurred

Returns the PublicKey

@since Mobiliser 5.0

getPrivateKey

Retrieves the PrivateKey from a JKSE keystore. First attempt is to load the keystore from the classpath with the class loader. If this fails it is tried to be retrieved from the file system.

Parameters

keyStoreName
keyStorePassword
keyAlias
keyPassword

Throws

KeyStoreException in case an error occurred

@deprecated please use `getPrivateKey`

Returns the PrivateKey

getPrivateKey

Retrieves the PrivateKey from a keystore. First attempt is to load the keystore from the classpath with the class loader. If this fails it is tried to be retrieved from the file system.

Parameters

keyStoreName
keyStorePassword
keyAlias
keyPassword
keyStoreType

Throws

KeyStoreException in case an error occurred

Returns the PrivateKey

@since Mobiliser 5.0

getKeyStore

Opens an existing JKSE keystore with the provided information.

Parameters

keyStoreName the full path to the keystore
keyStorePassword the password to open the keystore

Throws

KeyStoreException
CertificateException
IOException
KeyStoreException
NoSuchAlgorithmException
CertificateException

@deprecated please use `AsymmetricKeyUtils`

Returns the KeyStore

getKeyStore

Opens an existing keystore with the provided information.

Parameters

keyStoreName the full path to the keystore

keyStorePassword the password to open the keystore
keyStoreType the keystore type

Throws

KeyStoreException
CertificateException
IOException
KeyStoreException
NoSuchAlgorithmException
CertificateException

Returns the KeyStore

@since Mobiliser 5.0

14.7 Formatting

14.7.1 Class com.sybase365.mobiliser.util.tools.formatutils.FormatUtils

Overview

The `FormatUtils` provide various convenience methods for formatting.

Fields

DEFAULT_NULL_VALUE The default value for `null` values.

DEFAULT_ASSIGNMENT The default assign value.

DEFAULT_SEPERATOR The default seperator.

DEFAULT_BRACKET_OPEN The default *open* bracket.

DEFAULT_BRACKET_CLOSE The default *close* bracket.

Constructors

FormatUtils

Methods

formatString Formats the specified text right aligned within a given length (minimum and maximum).

Missing characters are filled with spaces.

`format("Seppo rocks",5)` will end up in "rocks"

`format("Seppo rocks",20)` will end up in "Seppo rocks"

Parameters

text The text to format
width The width

Returns The formatted string

formatString Formats the specified text with the given alignment and the given length (minimum and maximum).

Additional text parts are cut from the tail if the text is left aligned or from the head if the text is right aligned.

Missing characters are filled with spaces.

`format("Seppo rocks",5,false)` will end up in "rocks"

`format("Seppo rocks",20,false)` will end up in "Seppo rocks"and is equal to `format("Seppo rocks",20)`.

`format("Seppo rocks",20,true)` will end up in "Seppo rocks "

Parameters

text The text to format

width The width

leftAlign flag which indicates, whether or not to align the retrived text left in the result

Returns The formatted string

formatString Formats the specified text right aligned with the maximum and the minimum length.

Additional text parts are cut the head.

Missing characters are filled with spaces.

`format("Seppo rocks",5,10)` will end up in "eppo rocks"the retrieved text will have a length of at least five and at maximum ten characters.

`format("Seppo",10,20)` will end up in "Seppo"the retrieved text will have a length of at least ten and at maximum twenty characters.

Parameters

text The text to format

minWidth The minimum width

maxWidth The maximum width. All additional characters are cut from the head.

Returns The formatted string

formatString Formats the specified text with the specified alignment, the maximum and the minimum length.

Additional text parts are cut from the tail if the text is left aligned or from the head if the text is right aligned.

Missing characters are filled with spaces.

`format("Seppo rocks",5,10,false)` will end up in "eppo rocks"the retrieved text will have a length of at least five and at maximum ten characters and is right aligned and cut from head.

`format("Seppo rocks",5,10,true)` will end up in "Seppo rock"the retrieved text will have a length of at least five and at maximum ten characters and is left aligned and cut from tail.

`format("Seppo",10,20,false)` will end up in "Seppo"the retrieved text will have a length of at least ten and at maximum twenty characters and is right aligned.

Parameters

text The text to format

minWidth The minimum width

maxWidth The maximum width.

leftAlign Flag which indicates, whether to left or to right align the text.

Returns The formatted string

toString

Creates a `String` representation of the specified map.

Parameters

map the `Map` to represent

nullValue the `String` to use for contained `null` values

assignValue the `String` with which to represent the *mapped to* relation

seperator the separator `String`

Returns a `String` representation of `map` or `null` if `map` is `null` or empty

toString

Creates a `String` representation of the specified map; using `DEFAULT_NULL_VALUE` as *`null` value*.

Parameters

map the `Map` to represent

assignValue the `String` with which to represent the *mapped to* relation

seperator the separator `String`

Returns a `String` representation of `map` or `null` if `map` is `null` or empty

toString

Creates a `String` representation of the specified map; using `DEFAULT_NULL_VALUE` as *`null` value*, `DEFAULT_ASSIGNMENT` as assignment char and `DEFAULT_SEPERATOR` as seperator.

Parameters

map the `Map` to represent

Returns a `String` representation of `map` or `null` if `map` is `null` or empty

toString

Creates a `String` representation of the specified collection.

Parameters

c the `Collection` to represent

nullValue the `String` to use for contained `null` values

seperator the separator `String`

Returns a `String` representation of `c` or `null` if `c` is `null` or empty

toString

Creates a `String` representation of the specified collection; using `DEFAULT_NULL_VALUE` as *`null` value*.

Parameters

c the `Collection` to represent

seperator the seperator `String`

Returns a `String` representation of `c` or `null` if `c` is `null` or empty

See Also `toString`

toString

Creates a `String` representation of the specified collection; using `DEFAULT_NULL_VALUE` as *`null` value* and `DEFAULT_SEPERATOR` as separator.

Parameters

c the `Collection` to represent

Returns a `String` representation of `c` or `null` if `c` is `null` or empty

See Also `toString`

toString

Creates a `String` representation of the specified array.

Parameters

array the `Object` array to represent

nullValue the `String` to use for contained `null` values

seperator the seperator `String`

Returns a `String` representation of `array` or `null` if `array` is `null` or empty

toString

Creates a `String` representation of the specified array; using `DEFAULT_NULL_VALUE` as *`null` value*.

Parameters

array the `Object` array to represent

seperator the seperator `String`

Returns a `String` representation of `array` or `null` if `array` is `null` or empty

See Also `toString`

toString

Creates a `String` representation of the specified array; using `DEFAULT_NULL_VALUE` as *`null` value* and `DEFAULT_SEPERATOR` as separator.

Parameters

array the `Object` array to represent

Returns a `String` representation of `array` or `null` if `array` is `null` or empty

See Also `toString`

arrayToString

Creates a `String` representation of the specified array. This method is intended to be used with arrays of primitives, for `Object` arrays, you should use `toString`.

Parameters

array the `Object` array to represent

nullValue the `String` to use for contained `null` values

seperator the separator `String`

Returns a `String` representation of `array` or `null` if `array` is `null` or empty

See Also `toString`

arrayToString

Creates a `String` representation of the specified array; using `DEFAULT_NULL_VALUE` as *`null` value*. This method is intended to be used with arrays of primitives, for `Object` arrays, you should use `toString`.

Parameters

array the `Object` array to represent

seperator the separator `String`

Returns a `String` representation of `array` or `null` if `array` is `null` or empty

See Also `toString`

arrayToString

Creates a `String` representation of the specified array; using `DEFAULT_NULL_VALUE` as *`null` value* and `DEFAULT_SEPERATOR` as seperator. This method is intended to be used with arrays of primitives, for `Object` arrays, you should use `toString`.

Parameters

array the `Object` array to represent

Returns a `String` representation of `array` or `null` if `array` is `null` or empty

See Also `toString`

toString

Creates a `String` representation of the specified `Object`.

Parameters

o the `Object` to represent

openBracket the *opening* bracket to use for nested elements

closeBracket the *closing* bracket to use for nested elements

Returns a `String` representing `o` or `null` if `o` is `null` or empty

See Also `arrayToString`, `toString`, `toString`

toString

Creates a `String` representation of the specified `Object`, using the default nesting bracket.

Parameters

o the `Object` to represent

Returns a `String` representing `o` or `null` if `o` is `null` or empty

See Also `DEFAULT_BRACKET_CLOSE`, `DEFAULT_BRACKET_OPEN`, `toString`

formatDuration

Formats the specified amount of time to an human readable format like *3 d 03:05:12.008*.
>Note that the hours, minutes, seconds and milliseconds of the duration are filled with leading zeros if necessary. The *days* are just printed if the duration is longer than one day, whereas the *milliseconds* are just rendered if they are <>0.

Parameters

d The duration in milliseconds.

Returns The string representation in human readable form.

toCurrency

Returns the specified *currency amount* (given in smallest/fractional currency unit) as a big decimal.

Parameters

amount the amount to transform

currency the `Currency` to use for transformation

Returns the appropriate `BigDecimal`

See Also `getDefaultFractionDigits`

formatCurrencyAmount

Formats the specified *currency amount* based on the specified currency and locale (without currency symbol).
>The reason why `getCurrencyInstance` is not used here is that, although one can change the used currency afterwards, the fraction digits are determined by the locale's settings and not by the currency's.

Parameters

amount the amount to format

currency the `Currency` to determine the fraction digits with

locale the `Locale` to format the amount with

Returns the formatted `String`

See Also `getDefaultFractionDigits`

formatCurrencyAmount

Formats the specified *currency amount* (given in smallest/fractional currency unit) based on the specified currency and locale (without currency symbol).

Parameters

amount the amount to transform & format

currency the `Currency` to use for transformation & determining the fraction digits

locale the `Locale` to format the amount with

Returns the formatted `String`

See Also `toCurrency`, `formatCurrencyAmount`

formatCurrencyAmount

Formats the specified *currency amount* (given in smallest/fractional currency unit) based on the specified locale (without currency symbol).

Parameters

amount the amount to transform & format

locale the `Locale` to format the amount with

Returns the formatted `String`

toAmount

Converts the given *currency string* (without currency symbol) to a *currency amount* in the smallest/fractional currency unit.

The parameter `locale` is used to determine the applicable `NumberFormat` and the default fraction scale.

Parameters

amount the amount to transform

locale the `Locale` used to convert the *currency string*

Throws

ParseException

Returns the *currency amount* in the smallest/fractional currency unit.

toAmount

Converts the given *currency string* (without currency symbol) to a *currency amount* in the smallest/fractional currency unit.

The parameter `locale` is used to determine the applicable `NumberFormat`, while the `currency` parameter is used to determine the default fraction scale.

Parameters

amount the amount to transform

currency the `Currency` used to convert the *currency string*

locale the `Locale` used to convert the *currency string*

Throws

ParseException

Returns the *currency amount* in the smallest/fractional currency unit.

getSaveXMLGregorianCalendar Converts a `java.util.Date` to a `XMLGregorianCalendar`

Parameters

date The date to convert

Returns An `XMLGregorianCalendar` instance or `null`

getSaveDate Converts a `XMLGregorianCalendar` to a `java.util.Date`

Parameters

calendar The `XMLGregorianCalendar` to convert

Returns A `java.util.Date` instance or `<null`

splitParameters Converts a `Comma seperated value list as used by mobiliser-cronjob` to a `java.util.Map<String,String>` containing key value pairs of the params.

Parameters

parameters The String to split

Returns A `java.util.Map` instance

14.8 Jackson

14.8.1 Interface ITypeResolverBuilderFactory

Class: com.sybase365.mobiliser.util.tools.jackson.ITypeResolverBuilderFactory

Overview

Methods

getTypeResolverBuilderForHierarchie Create a .TypeResolverBuilder which can resolve type hierarchies.

Parameters

parentType the parent type

childTypes the child types

Returns the resolver builder

14.8.2 Class MapUnwrappingObjectMapper

Class: com.sybase365.mobiliser.util.tools.jackson.MapUnwrappingObjectMapper

Overview

A custom extension of the Jackson ObjectMapper that unwraps a Map object if the map contains only a single entry. The unwrapping support allows the endpoint to return a model as a map but no have an extra Map wrapper object in the generated JSON.

This class also implements the Spring InitializingBean interface to register the JAXB annotation inspector and a custom serializer type.

Constructors

MapUnwrappingObjectMapper

Methods

afterPropertiesSet

See Also .org.springframework.beans.factory.InitializingBean.afterPropertiesSet()

canDeserialize

canSerialize Method that can be called to check whether mapper thinks it could serialize an instance of given Class. Check is done by checking whether a serializer can be found for the type.

Returns True if mapper can find a serializer for instances of given class (potentially serializable), false otherwise (not serializable)

writeValue

See Also .org.codehaus.jackson.map.ObjectMapper.writeValue(java.io.File, java.lang.Object)

writeValue

See Also .org.codehaus.jackson.map.ObjectMapper.writeValue(org.codehaus.jackson. JsonGenerator, java.lang.Object)

writeValue

See Also .org.codehaus.jackson.map.ObjectMapper.writeValue(org.codehaus.jackson. JsonGenerator , java . lang . Object , org . codehaus . jackson . map . SerializationConfig)

14.8.3 Class TypeResolverBuilderFactoryImpl

Class: com.sybase365.mobiliser.util.tools.jackson.impl.TypeResolverBuilderFactoryImpl

Overview

Default implementation.

Constructors

TypeResolverBuilderFactoryImpl

Methods

getTypeResolverBuilderForHierarchie

See Also `getTypeResolverBuilderForHierarchie`

14.8.4 Class SybaseTypeNameIdResolver

Class: com.sybase365.mobiliser.util.tools.jackson.impl.SybaseTypeNameIdResolver

Overview

Override standard .TypeNameIdResolver to use for both serialisation and deserialisation.

Constructors

SybaseTypeNameIdResolver

Parameters

baseType
typeToId
idToType

Methods

construct

14.9 JAXB

14.9.1 Class com.sybase365.mobiliser.util.tools.jaxbutils.PropagatingErrorHandler

Overview

`ErrorHandler` which propagates the exceptions back to the caller.

Constructors

PropagatingErrorHandler

Methods

error

See Also `error`

fatalError

See Also `fatalError`

warning

See Also `warning`

14.9.2 Class com.sybase365.mobiliser.util.tools.jaxbutils.SchemaWrapper

Overview

Wrapper for a `Schema` which implements `toString` to print the schema files associated with the delegate schema.

Constructors

SchemaWrapper

Parameters

schema
schemaResources

Methods

newValidator

See Also `newValidator`

newValidatorHandler

See Also `newValidatorHandler`

toString

See Also `toString`

14.9.3 Interface com.sybase365.mobiliser.util.tools.jaxbutils.IJaxbObjectValidator

Overview

Implementations of this interface can validate jaxb objects against schema objects.

Methods

validateObject Validates an object against the configured schema.

Parameters

jaxbObject the jaxb object

14.9.4 Class JaxbObjectValidatorFactoryBean

Class: com.sybase365.mobiliser.util.tools.jaxbutils.JaxbObjectValidatorFactoryBean

Class: com.sybase365.mobiliser.util.tools.jaxbutils.JaxbObjectValidatorFactoryBean

Overview

Constructors

JaxbObjectValidatorFactoryBean

Parameters

marshaller

Methods

getObjectType

createInstance

14.9.5 Class com.sybase365.mobiliser.util.tools.jaxbutils.JaxbObjectValidator

Overview

Validates a JAX-B object against a schema. It defaults to a `PropagatingErrorHandler` which propagates schema exceptions up the call stack to the caller.

Constructors

JaxbObjectValidator

Methods

afterPropertiesSet

convertJaxbException Convert the given `JAXBException` to an appropriate exception from the `org.springframework.oxm` hierarchy.

Parameters

ex `JAXBException` that occured

Returns the corresponding `XmlMappingException`

setContext

Parameters

context the context to set

setErrorHandler

Parameters

errorHandler the errorHandler to set

setSchema

Parameters

schema the schema to set

toString

See Also `toString`

validateObject

See Also `validateObject(java.lang.Object)`

14.10 JMS

14.10.1 Class UnsupportedOperationInterceptor

Class: com.sybase365.mobiliser.util.tools.jmsutils.UnsupportedOperationInterceptor

Overview

Just a helper interceptor so we can lazily wire-up the JMS connection factory. Until our configuration is loaded with the real configuration, just throw an `UnsupportedOperationException`.

Constructors

UnsupportedOperationInterceptor

Methods

invoke

See Also .org.aopalliance.intercept.MethodInterceptor.invoke(org.aopalliance.intercept.MethodInvocation)

14.10.2 Class UpdatableConnectionFactoryTargetSource

Class: com.sybase365.mobiliser.util.tools.jmsutils.UpdatableConnectionFactoryTargetSource

Overview

Helper class for processing configadmin, preferences or some other updates. This class will always create a new delegate activemq connection factory.

Constructors

UpdatableConnectionFactoryTargetSource Constructor.

Parameters

initialTarget initial target.

Methods

releaseTarget

setInitialConfiguration Update the properties.

Parameters

properties the properties

updateCallback Update the properties.

Parameters

properties the properties

14.11 Large Binary Object

14.11.1 Class com.sybase365.mobiliser.util.tools.lob.LobHandlerHolder

Overview

Just a simple holder object for the handler.

Constructors

LobHandlerHolder

Methods

getLobHandler

Returns the lobHandler

getDatabaseProductNames

Returns the databaseProductNames

setLobHandler

Parameters

lobHandler the lobHandler to set

setDatabaseProductNames

Parameters

databaseProductNames the databaseProductNames to set

14.11.2 Class com.sybase365.mobiliser.util.tools.lob.LobHandlerFactoryBeanTest

Overview

Constructors

LobHandlerFactoryBeanTest

Methods

setUp setup.

tearDown Teardown.

Throws

Exception

testCustom Tests H2.

Throws

Exception

testDB2 Tests DB2.

Throws

Exception

testH2 Tests H2.

Throws

Exception

testH2FromDataSource Tests H2.

Throws

Exception

testOracle Tests Oracle.

Throws

Exception

14.11.3 Class com.sybase365.mobiliser.util.tools.lob.LocalNativeJdbcExtractor

Overview

Implementation of Spring's .NativeJdbcExtractor interface that returns a Spring-managed .NativeJdbcExtractor, determined by LobHandlerFactoryBean's "nativeJdbcExtractor"property.

Constructors

LocalNativeJdbcExtractor

Methods

isNativeConnectionNecessaryForNativeStatements

See Also .org.springframework.jdbc.support.nativejdbc.NativeJdbcExtractor.isNativeConnectionNecessaryForNativeStatements()

isNativeConnectionNecessaryForNativePreparedStatements

See Also .org.springframework.jdbc.support.nativejdbc.NativeJdbcExtractor.isNativeConnectionNecessaryForNativePreparedStatements()

isNativeConnectionNecessaryForNativeCallableStatements

See Also .org.springframework.jdbc.support.nativejdbc.NativeJdbcExtractor.isNativeConnectionNecessaryForNativeCallableStatements()

getNativeConnection

See Also .org.springframework.jdbc.support.nativejdbc.NativeJdbcExtractor.getNativeConnection(java.sql.Connection)

getNativeConnectionFromStatement

See Also .org.springframework.jdbc.support.nativejdbc.NativeJdbcExtractor.getNativeConnectionFromStatement(java.sql.Statement)

getNativeStatement

See Also .org.springframework.jdbc.support.nativejdbc.NativeJdbcExtractor.getNativeStatement(java.sql.Statement)

getNativePreparedStatement

See Also .org.springframework.jdbc.support.nativejdbc.NativeJdbcExtractor.getNativePreparedStatement(java.sql.PreparedStatement)

getNativeCallableStatement

See Also .org.springframework.jdbc.support.nativejdbc.NativeJdbcExtractor.getNativeCallableStatement(java.sql.CallableStatement)

getNativeResultSet

See Also .org.springframework.jdbc.support.nativejdbc.NativeJdbcExtractor.getNativeResultSet(java.sql.ResultSet)

14.11.4 Class com.sybase365.mobiliser.util.tools.lob.LobHandlerFactoryBean

Overview

Factory interface for getting the .LobHandler for a datasource.

Constructors

LobHandlerFactoryBean

LobHandlerFactoryBean

Parameters

overrideResource additional resource to load containg lob handler definitions

Methods

getConfigNativeJdbcExtractor Return the NativeJdbcExtractor for the currently configured for this factory, to be used by LocalNativeJdbcExtractor.

This instance will be set before initialisation of the corresponding LobCreator, and reset immediately afterwards. It is thus only available during configuration.

Returns the extractor

See Also `setNativeJdbcExtractor`, `LocalNativeJdbcExtractor`

afterPropertiesSet Eagerly create the singleton instance, if necessary.

createInstance

destroyInstance

See Also .org.springframework.beans.factory.config.AbstractFactoryBean.destroyInstance(java.lang.Object)

getObjectType

setDataSource

Parameters

dataSource the dataSource to set

setDbName

Parameters

dbName the dbName to set

setNativeJdbcExtractor Set the NativeJdbcExtractor to be used by the LobCreator.

Parameters

nativeJdbcExtractor the nativeJdbcExtractor to set

14.12 Maths

14.12.1 Class com.sybase365.mobiliser.util.tools.mathutils.MathUtils

Overview

The `MathUtils` class provides convenient calculations for problems commonly found throughout the *Mobiliser Platform*.

Methods

getPercentage

Calculates the percental share of the specified amount.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

amount the base amount (being a `BigDecimal`)

percentage the percentage value

Returns the percental share

getPercentage

Calculates the percental share of the specified amount.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

amount the base amount (being a `long`)

percentage the percentage value

Returns the percental share

See Also `getPercentage`

getPercentageAsLong

Calculates the percental share of the specified amount, returning a rounded `long` value.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

amount the base amount (being a `BigDecimal`)

percentage the percentage value

roundingMode the `RoundingMode` to apply

Throws

ArithmeticException if the result has a nonzero fractional part (`UNNECESSARY`), or will not fit in a `long`

Returns the percental share as `long`

See Also `getPercentage`

getPercentageAsLong

Calculates the percental share of the specified amount, returning a rounded `long` value.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

amount the base amount (being a `long`)

percentage the percentage value

roundingMode the `RoundingMode` to apply

Throws

ArithmeticException if the result has a nonzero fractional part (`UNNECESSARY`), or will not fit in a `long`

Returns the percental share as `long`

See Also `getPercentage`

getPercentageAsLong

Calculates the percental share of the specified amount, returning a rounded (`half-up`) `long` value.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

amount the base amount (being a `BigDecimal`)

percentage the percentage value

Throws

ArithmeticException if the result will not fit in a `long`

Returns the percental share as `long`

See Also `getPercentage`

getPercentageAsLong

Calculates the percental share of the specified amount, returning a rounded (`half-up`) `long` value.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

amount the base amount (being a `long`)

percentage the percentage value

Throws

ArithmeticException if the result will not fit in a `long`

Returns the percental share as `long`

See Also `getPercentage`

getNetAmount

Performs a net calculation for the specified amount and rate. This is typically used when calculating the net amount based on gross amount and VAT rate.
>If the exact quotient does not have a terminating decimal expansion (see `divide`), the specified `scale` and `roundingMode` will be used to constrain the result. If such a condition does not occur, the `scale` and `roundingMode` values will be ignored.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

grossAmount the gross amount (being a `BigDecimal`)

percentage the percentage value

scale the scale to apply if a *non-terminating decimal expansion* occurs

roundingMode the `RoundingMode` to apply if a *non-terminating decimal expansion* occurs

Returns the net amount as `BigDecimal`

getNetAmount

Performs a net calculation for the specified amount and rate. This is typically used when calculating the net amount based on gross amount and VAT rate.
>If the exact quotient does not have a terminating decimal expansion (see `divide`), the specified `scale` and `roundingMode` will be used to constrain the result. If such a condition does not occur, the `scale` and `roundingMode` values will be ignored.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

grossAmount the gross amount (being a `long`)

percentage the percentage value

scale the scale to apply if a *non-terminating decimal expansion* occurs

roundingMode the `RoundingMode` to apply if a *non-terminating decimal expansion* occurs

Returns the net amount as `BigDecimal`

See Also `getNetAmount`

getNetAmount

Performs a net calculation for the specified amount and rate. This is typically used when calculating the net amount based on gross amount and VAT rate.
>If the exact quotient does not have a terminating decimal expansion (see `divide`), the specified `scale` and `HALF_UP` will be used to constrain the result. If such a condition does not occur, the `scale` value will be ignored. , a scale value of `9` will be used to constrain the result. If such a condition does not occur, no scaling is applied.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

grossAmount the gross amount (being a `BigDecimal`)

percentage the percentage value

scale the scale to apply if a *non-terminating decimal expansion* occurs

Returns the net amount as `BigDecimal`

See Also `getNetAmount`

getNetAmount

Performs a net calculation for the specified amount and rate. This is typically used when calculating the net amount based on gross amount and VAT rate.
>If the exact quotient does not have a terminating decimal expansion (see `divide`), the specified `scale` and `HALF_UP` will be used to constrain the result. If such a condition does not occur, the `scale` value will be ignored. , a scale value of `9` will be used to constrain the result. If such a condition does not occur, no scaling is applied.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

grossAmount the gross amount (being a `long`)

percentage the percentage value

scale the scale to apply if a *non-terminating decimal expansion* occurs

Returns the net amount as `BigDecimal`

See Also `getNetAmount`

getNetAmount

Performs a net calculation for the specified amount and rate. This is typically used when calculating the net amount based on gross amount and VAT rate.

>If the exact quotient does not have a terminating decimal expansion (see `divide`), a scale value of `9` and `HALF_UP` will be used to constrain the result. If such a condition does not occur, no scaling is applied.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

grossAmount the gross amount (being a `BigDecimal`)
percentage the percentage value

Returns the net amount as `BigDecimal`

See Also `getNetAmount`

getNetAmount

Performs a net calculation for the specified amount and rate. This is typically used when calculating the net amount based on gross amount and VAT rate.
>If the exact quotient does not have a terminating decimal expansion (see `divide`), a scale value of `9` and `HALF_UP` will be used to constrain the result. If such a condition does not occur, no scaling is applied.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

grossAmount the gross amount (being a `long`)
percentage the percentage value

Returns the net amount as `BigDecimal`

See Also `getNetAmount`

getNetAmountAsLong

Performs a net calculation for the specified amount and rate, returning a rounded `long` value. This is typically used when calculating the net amount based on gross amount and VAT rate.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

grossAmount the gross amount (being a `BigDecimal`)
percentage the percentage value
roundingMode the `RoundingMode` to apply

Returns the net amount as `long`

See Also `getNetAmount`

getNetAmountAsLong

Performs a net calculation for the specified amount and rate, returning a rounded `long` value. This is typically used when calculating the net amount based on gross amount and VAT rate.
>This method uses `HALF_UP` for rounding.

This method assumes that the `percentage` is given in a *percentage notation* like this:
>10% ≡ `10`
>.12% ≡ `0.12`
>2.3% ≡ `2.3`

Parameters

grossAmount the gross amount (being a `long`)
percentage the percentage value

Returns the net amount as `long`

See Also `getNetAmountAsLong`

14.13 Phone Number

14.13.1 Class com.sybase365.mobiliser.util.tools.phoneutils.PhoneNumber

Overview

The `PhoneNumber` is mainly used to transform an MSISDN between national and international notation.

Fields

DEFAULT_COUNTRY_CODE A default country code.

Constructors

PhoneNumber

Creates a new instance of `PhoneNumber`. Tries to parse the specified MSISDN and falls back to use the specified country code if no internationalised format is recognised.

Parameters

msisdn the phone number (not necessarily a mobile phone number); e.g. +491791234567
countryCode the country code; e.g. 49

PhoneNumber

Creates a new instance of `PhoneNumber`. Tries to parse the specified MSISDN and falls back to use the `DEFAULT_COUNTRY_CODE` if no internationalised format is recognised.

Parameters

msisdn the phone number (not necessarily a mobile phone number)

Methods

getInternationalFormat

Returns the phone number in international form (e.g. +491701234567).

Returns the MSISDN in international format

getNumericInternationalFormat

Returns the phone number in numeric international form (e.g. 00491701234567 for +491701234567).

Returns the MSISDN in numeric international format

getShortInternationalFormat

Returns the phone number in a truncated international form (e.g. 491701234567 for +491701234567).

Returns the MSISDN in truncated international format

getNationalFormat

Returns the phone number in the national form.
>An example is 01701234567 for the phone number +491701234567

Returns the MSISDN in national format

toString

Returns a string representation of this instance.

Returns `getInternationalFormat`

equals

Indicates if another object is *equal to* this one.

Parameters

o the `Object` to compare with

Returns `true` if `o` is a `PhoneNumber` representing the same MSISDNF

hashCode

Returns a hash value.

Returns a hash value for this instance

14.14 Random

14.14.1 Interface com.sybase365.mobiliser.util.tools.random.IRandomRepository

Overview

Repository for getting `SecureRandom`s.

Methods

clear Clear the currently cached secure random, if any.

getSecureRandom Returns the `SecureRandom` instance to use. Clients should always call this method and not cache the result since the repository may decide that the `SecureRandom` may be expired after a certain time ect.

Returns the secure random to use.

14.14.2 Class com.sybase365.mobiliser.util.tools.random.impl.RandomRepositoryImpl

Overview

Default impl.

Fields

LOG

Constructors

RandomRepositoryImpl

Methods

clear

getSecureRandom

See Also `getSecureRandom`

14.14.3 Class com.sybase365.mobiliser.util.tools.random.impl.SecureRandomReseeder

Overview

Based on this **article** (at), adds new material to the secure random at intervals, and also throws it away and creates a new one at certain intervals.

Constructors

SecureRandomReseeder

Methods

afterPropertiesSet

destroy

setClearingInterval

Parameters

clearingInterval the clearingInterval to set

setExecutor

Parameters

executor the executor to set

setRepository

Parameters

repository the repository to set

14.15 Spring

14.15.1 Class DynamicOsgiServiceFactoryBean<T>

Class: com.sybase365.mobiliser.util.tools.spring.DynamicOsgiServiceFactoryBean<T>

Overview

This bean can be registered in a spring application context and it will export the target bean through the osgi registry. It includes a refresh method to unregister and reregister the target bean. (This assumes the refresh itself has somehow affected the given bean)

Constructors

DynamicOsgiServiceFactoryBean

Methods

afterPropertiesSet

destroy

getLastRefreshTime

getRefreshCount

refresh

setBeanClassLoader

setBeanFactory

setBundleContext

setTarget

Parameters

target the target to set

setInterfaces

Parameters

interfaces the interfaces to set

setServiceProperties

Parameters

serviceProperties the serviceProperties to set

14.15.2 Class com.sybase365.mobiliser.util.tools.spring.OrderedImpl

Overview

Implementation of the ordered interface, useful for creating introductions for objects that should implement .Ordered.

Constructors

OrderedImpl

Parameters

order

Methods

getOrder

See Also .org.springframework.core.Ordered.getOrder()

14.15.3 Class OSGiPropertyPlaceholderConfigurer

14.15.4 Class: com.sybase365.mobiliser.util.tools.spring.OSGiPropertyPlaceholderConfigurer

Overview

We register a custom .PropertyPlaceholderConfigurer which is "dependency aware"which means it is only executed after waiting for any dependent objects to become available. This class also polls the .ConfigurationAdmin until the configured PID is available. A timeout may be configured. The default is 60 seconds. Overview of the problem can be found **here** (at http://forum.springsource.org/showthread.php?p=281561).

Constructors

OSGiPropertyPlaceholderConfigurer

Methods

getOrder

postProcessBeanFactory

setBeanFactory

setBeanName

setBundleContext

setConfigurationAdmin Sets the configuration admin.

Parameters

configurationAdmin admin

setInitTimeout Init timeout in seconds.

Parameters

initTimeout the initTimeout to set

setPersistentId Sets the ConfigurationAdmin persistent Id that the bean should read.

Parameters

persistentId The persistentId to set.

setProperties Sets the local properties.

Parameters

properties

Chapter 15

Testing

Mobiliser uses Easy Mock[1] and the Spring testing support[2] for unit testing. All these frameworks are based on JUnit4[3]. Aside frameworks, there is a few principles to consider in software design to facilitate easy testing of components.

15.1 General Guidelines

By the use of interfaces and IoC patterns, each class (that's worth it) must be unit-testable without any dependencies on other classes implementing complex business logic. When unit testing a class, rely on any injected classes being verified as correct already, provide mock objects for any injected class, and concentrate on the validation of the class under test. In case you can not mock a dependency and you experience difficulties because of that in the test execution, reconsider the design of your class and change so that this other class is injected and implements an interface so that you can easily mock it away.

We recommend to use EasyMock to create mock objects for all beans that are injected into the class under test; if the testing scenario requires too complex logic, it is also okay to explicitly provide a test implementation of the required interface and inject an instance of this test implementation.

Each test class must have all mock objects, commonly used test data, and the class under test as private class properties. Use the $\mathrm{@Before}$ method to initialize the mocks, test data, and class under test with all behavior that is common to all test cases. In each single $\mathrm{@Test}$ method, set the expectations on the mock objects. Follow the expected call flow that you have for that particular test, so that it is easy to read and understand what the test is trying to validate. Then, call $\mathrm{replay}()$, the method under test, and $\mathrm{verify}()$ to actually run the test. Implement an $\mathrm{@After}$ method that is resetting all mock objects to prevent interference between distinct test methods. In case the method under test does return a value, place an assertion on the expected value before calling $\mathrm{verify}()$. Also, always express your assertions for the mock expectations using EasyMock's $\mathrm{eq}*$ methods. In case you want to validate the content of a complex object, simply expect $\mathrm{anyObject}()$ and implement an $\mathrm{IAnswer}$ that does the validation on $\mathrm{EasyMock.getCurrentArguments}()$. Placing proper assertions and expectations is critical for unit testing; by simply running the test methods, you can increase test coverage easily, but only by checking parameter and return values, you can validate if code does what you expect. In case you want to visualize if your test cases cover most possible executions of your code, you can use the Eclipse plugin EclEmma[4]; run your test case with that, and get untested lines of code highlighted in your Eclipse editor.

[1] http://www.easymock.org/
[2] http://static.springsource.org/spring/docs/3.0.5.RELEASE/reference/testing.html
[3] http://www.junit.org/
[4] http://www.eclemma.org/

15.2 Unit Testing

15.2.1 Testing Spring Wiring and Transaction Handling

Spring misconfiguration is a common cause of error. While these problems will show up immediately during container startup, it is best to catch configuration problems early on with unit tests. This also relieves developers maintaining your code, since configuration breaking changes will be discovered during build time.

The most basic test includes simply do validate that all beans get instantiated by Spring. To this end you have to replace all imported OSGi services, since this would not be available for unit testing. Hence, the mandated configuration file layout is to have a separate bundle-context-osgi.xml file for configuration of all osgi:reference, osgi:list, osgi:set, and osgi:service elements. This allows to reuse all remaining configuration for wiring tests and validate as much as possible from the configuration. There is obviously no way to validate OSGi configuration outside of the container.

Include a dependency on the Spring test JAR in your project's POM and make your test case run with the SpringJunit4ClassRunner by annotating the test class with @RunWith(SpringJUnit4ClassRunner.class). Additionally, provide all configuration files by setting the @ContextConfiguration(locations = "/META-INF/spring/bundle-context.xml", "/osgi-service-ref-mock.xml") annotation to the test class as well. You should include all required configuration from the src/main/resources folder, but the OSGi specific configuration and include a mock configuration in your src/test/resources folder to provide mocked objects for the beans that are imported through the OSGi service registry.

The mock objects could be test classes that you provide an implement in src/test/java, but in case you just want to do basic validation, you can also have EasyMock create mock object for you:

```
<bean id="daoFactory"
  class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
  <property name="staticMethod" value="org.easymock.EasyMock.createMock" />
  <property name="arguments">
    <list>
      <value>
         com.sybase365.mobiliser.money.persistence.dao.factory.api.DaoFactory
      </value>
    </list>
  </property>
</bean>
```

For each bean that is defined in your Spring configuration file, add a class property to your test class, annotate it with @Resource to get it injected, and define assertions in your test method that make sure that these beans are not null.

You can also validate the transactional behavior of your classes. Simply provide an EasyMock PlatformTransactionManager and set the expectations accordingly, in particular make sure that the method transactionManager.getTransaction is invoked at the right places with the right transaction settings.

```
EasyMock.expect(
 transactionManager.getTransaction(EasyMock
  .anyObject(TransactionDefinition.class))).andAnswer(
 new IAnswer<TransactionStatus>() {
     @Override
     public TransactionStatus answer() throws Throwable {
  TransactionDefinition def = (TransactionDefinition) EasyMock
   .getCurrentArguments()[0];
  Assert.assertEquals(
   TransactionDefinition.ISOLATION\_DEFAULT,
   def.getIsolationLevel());
  Assert.assertEquals(
   TransactionDefinition.PROPAGATION\_REQUIRED,
   def.getPropagationBehavior());
  return null;
     }
 });
```

15.2.2 Testing Persistence

The core persistence bundle comes with a set of test cases for the Hibernate implementation of DAO APIs. These test cases use the Hibernate capability to auto-generate the database during initialization. For testing, Hibernate points to an in-memory database, which is created on-the-fly. The test cases insert data into this database, run the DAO methods, and validate the results. Writing new test cases for new DAO methods is strongly recommended; this is an easy way to find simple mistakes on HQL grammar as well as logical flaws in the HQL itself.

15.2.3 Testing Business Logic

The $\mathrm{money/businesslogic/util}$ bundle contains a $\mathrm{TestBusinessLogic}$ test class that can be extended to inherit a test case that provides automatic basic tests for all implementations of the corresponding BL-API. Subclasses of $\mathrm{TestBusinessLogic}$ have to set the two generic types:

LogicInterface: The interface that provides the BL-API.

LogicImplementation: The class implementing the BL-API specified in LogicInterface

If your BL-Implementation has field-dependencies to other BL-Implementations you have also to override the method $\mathrm{getBusinessLogicFieldClasses()}$ and return an array which contains all classes of the BL-Impl. field-dependencies. These classes will automatically be injected into the BL-Implementation under test.

Example:

```
public class TestSecurityLogic extends TestBusinessLogic<ISecurityLogic,
SecurityLogic> {
    @Override
    protected java.lang.Class<?>[] getBusinessLogicFieldClasses() {
        return new Class<?>[] { CustomerLogic.class };
    }
}
```

Additionally, you can use Easymock for more sophisticated semantic unit testing.

15.3 Service Integration Testing

We suggest to use SoapUI for the full service integration testing. SoapUI allows to define Test Cases that invokes a sequence of steps, even with conditional branches and dynamic value replacement.

You can use custom project properties to define properties, such as the Mobiliser URL, service users, passwords or JDBC connection URLs and use these property values in the projects. This allows you to share soapUI projects with other team members even if they are using a different environment setup.

Please see the SoapUI homepage[5] for downloads, tutorials and references.

[5] http://www.soapui.org/

Appendices

Appendix A

Best Practices

This Chapter gives you an overview of some of the best practices we recommend you follow during development.

A.1 Patterns

One general rule of thumb is simply:

Write your code as if you were the one who has to maintain it after not dealing with it for a year.

A.1.1 General Java

Readers of "Effective Java" and the likes will have a few deja vu experiences here, this is only a small subset of the most important things every Java developer should know.

Don't let Methods return null. If the result is empty, have methods return an empty Collection or array instead. In the rare cases where there is no other option comment this behavior.

Why?: Because it does not matter if you do a check on $\mathrm{List.size}{>}0$ or $\mathrm{List}{=}\mathrm{null}$ and "NullPointers" should be avoided wherever possible.

Use $\mathbf{Stringbuffer.append}$ instead of $+=$ when concatenating $\mathbf{Strings}$. A common task where this is done is when building queries where different parts are dynamically added.

Why?: Because String is not mutable, every time you do a $+=$ you create a new String object in memory. The "trash" you generate grows exponentially with the length of the String. Doing this in one spot in the code has no great impact, but across an application it adds up, it is simply unnecessary. $\mathrm{StringBuffer}$ and $\mathrm{StringBuilder}$(Non synchronized!) were created for exactly this purpose.

When working with resources (Files, DB) always close these in a finally block. Why?: Because the finally block is the only piece of code which will be executed no matter what Exception occurs and that's exactly where you want to place the code to release the resources you are holding.

Never throw "$\mathbf{Exception}$", use a more descriptive exception instead If you need to terminate the flow of a program, throw an exception specific to the problem. If the parameters are invalid throw an $\mathrm{IllegalArgumentException}$ for example.

Why?: The specific problem can only be handled correctly if the exception which is thrown by a method is specific enough to indicate the problem.

Prefer Collections over Arrays In 9 out of 10 cases it is possible to use Collections instead of Arrays. Collections may use a little more memory but the benefits usually outweigh the costs.

Why?: Collections are easier to handle due to the Iterateable interface and the Datastructure Design, plus, you don't have to worry about ArrayOutOfBoundsExceptions anymore.

Make sure your method returns the DB connection/transaction to the same state it had before the method call... If you call a commit on an open transaction...do a beginTransaction before exiting the method, the developer calling your method and using the DB Connection afterwards will thank you :-)

Before you write a method for a generic task, check if it has been done by someone else

1. Check the UtilityClasses of the application you are developing,
2. check the bundles in the Utils Component of Mobiliser
3. Depending on the type of task check the Apache Commons libraries.

Why?: You save yourself work. Additionally these libs and methods have been around for a while so they are pretty hardened in means of correctness and error handling.

Never return internal arrays or collections that must not be changed If you return an internal array or collection directly, the caller may change value outside, so this is a possible security issue. Therefore you may use Object.clone to return a copy of the array or return a immutable copy of the collection

A.1.2 Exception Handling

Don't use "Exception" to catch exceptions Catch exceptions as specific as possible and try to handle them as specific as possible.

Why?: Because you will catch all kinds of exceptions from NullPointerExceptions to NumberFormatExceptions, this basically makes it impossible to handle the specific exception correctly.

If you catch an exception then handle it Don't swallow exceptions by leaving them unhandled. If you think you can't handle a certain exception, comment why you can't, and at the very least write some logging about it.

Don't use System.out.* or Error.out.* ...you never know where it may be written to, use proper logging instead.

Don't use Exception.printStackTrace ...use LOG.error("What you have to say about it",exception) instead.

Why?: Basically the same as above, it ensures it is written to the correct logfile.

A.1.3 Architecture

Collect constants in a "Constants" class instead of spreading them all over the code Many of our applications already have such a constants class. For customizations of standard applications you may want to simply create a "CustomConstants" class which inherits the standard Constants class.

Use Preferences for "pseudo constant" values which change close to never but may need to be configured later

Collect all Preferences which have a application wide impact in a configuration class Why?: It makes configuration of variables which are used in the whole Application easier. Work with the preferences manager and you will know what I mean.

Don't have applications use webservices AND direct JDBC to access the same DB... ...only the one or the other.... Using both is very error prone since you will very likely have racing conditions between the two as well as deadlocks on the DB.

A.1.4 Testing

Junit Test cases should be written for all available methods (the ones which are not private at the very least) Remember to write test cases for failure scenarios, not only for the "good" case

The "good" case may be the exception, test how robust your method is by giving it some invalid input or leaving out parameters. A high test coverage will spare you of many headaches in the long run, the initial investment in taking time write a testcase is a good investment. If you have to modify the tested code later you will already have a test harness which will give you some indication if you broke something during your change.

Don't modify a working JUnit test to test to cater for a different scenario JUnit tests are meant to be cumulative! If you want to test something new, write a new test and add it to the test suite.

If you have a bug in your code, write a Testcase for it! If someone finds a Bug in your code then you must have not tested that scenario which means your testcases were not extensive enough. So, as soon as a bug is reported create a testcase which reproduces it. This will help you to determine if you have fixed the bug when you think you have and will also serve as a regression test for changes done later to prevent anyone from "un-fixing" the bug through other changes.

Check your test code coverage If you don't have Sonar checking your code you can also check your coverage locally by using ECLEmma or a similar tool, this shows you which parts of your code you have tested and which parts are not covered by your current tests. Complete Test Code Coverage does not give a guarantee that you wont́ still have bugs but at least you know you have tested it at all.

A.1.5 Comments

Comment everything that is not trivial and understandable without a deeper knowledge of the application.

As written in the header of this Chapter, imagine that you quit working on something today and need to resume work on that in a year. Most of the application details which may be trivial to you now won't be then.

Make use of TODO, XXX and FIXME Check your code for them to make sure you haven't forgotten anything that you need to fix before delivering your code.

Comment your methods At the very least write a brief description comment on what they do. JavaDoc comments are the most preferable way of doing this.

A.1.6 Misc

Don't hardcode expressions, which may be seen by end users Use property files instead, you never know if the application you are working on needs to be localized to Russian or Spanish for example. This is also valid for JSPs, localize wherever possible.

When adding non Java files to a project, make sure the build will include the corresponding file types When adding a png image for use in a report for example, an ant or maven build may ignore this file when building the artifact (jar , war etc.) since it will only include file types which are defined in the build. When the artifact is deployed to the server the png will be missing since it was not included.

Remove dead code If it isn't used and/or it is commented out...remove it ASAP. You can always get an older version of the source code from the SVN if you notice you require it later.

Use FindBugs or a similar tool if your code is not covered by Sonar, use FindBugs for static code checks locally, it will give you some hints on what lines of your code may cause problems. Although there are quite a few false alarms, it is definitely worthwhile to analyze what it reports.

Use Java Script in JSPs sparingly Don't use Javascript to generate multiple views on a single page for example.

Remember that it's harder to maintain a 200 line Javascript which shows or hides various elements on the page depending on various roles and rights, than to create and maintain 2 or 3 JSPs.

Note: Reusability is generally a good thing, but placing everything inside a single monolith which changes its face by using lots of code which nobody will understand in a couple of months, really misses the goal here.

A.2 Security

A.2.1 General

Do not log secret data such as PINs and Passwords You never know who may gain access to the log-files...an alternative can be to hash these values. You can then still check if the password matches the value you are expecting, by simply hashing it and comparing the value in the log. Use the utility classes in the $\mathrm{encryptionutils}$ bundle of the utils component for for this task.

Use SecureRandom instead of Random Whenever you require a real random and not only a pseudorandom (for generating pins for example) use the $\mathrm{SecureRandom}$ class.

Check the validity of all incoming parameters before passing them on It is **your** duty to check if the incoming values don't contain harmful (invalid characters or characters used for exploits such as SQL-Injection or XSS) or invalid contents.

Do not insert variables into query strings . . . **always** use the prepared/parametrized statements instead.

This is a requirement which helps to prevent SQL-Injection Attacks. If you use a JPA implementation you get type validation for free.

OWASP Top 10: Know the most common vulnerabilities Check out the OWASP Top-10 List

- SQL-Injection[1]

  As already mentioned above, construct the queries correctly using some form of PreparedStatements with checked parameters in Hibernate and this should be no problem.

- XSS(Wicket)[2]

  How to prevent: Per default XSS protection is activated in Wicket (Markup will be escaped upon OUTPUT) which can only be deactivated by using ".setEscapeModelStrings(false));" so this should never be set to false...it is true by default. It is still **your** duty to validate user input before you accept it into you application. This is even more important in case you need to pass the data on to another application or system.

- Broken Auth & Session Management[3]

  This may be prevented by timing out the session on inactivity on all of our applications, Mobiliser provides this functionality.

- Insecure Direct Object References[4]

  This is an issue which may be observed more often than expected. Whenever possible use indirect references. E.g. If a customer has multiple payment instruments he may use for a transaction and he needs to choose one of them on the next page, then store the payment instruments IDs in a Map and only provide the mapped values

  ```
  1->234782344
  2->234432654
  ```

  to the frontend. This way you can then resolve the correct ID from the index. This eliminates the possibility of this user manipulating a request to use some other PI-ID for the transaction since he can only choose from the list of IDs in the session.

- Check for inconsistencies between the front-end logic and the back-end
  I have seen more than once that Business logic was implemented solely by restricting the view in the front-end. E.g. a customer (or at least parts of his data) must not be edited if he is in BLR 5, this is enforced in the front-end view (by showing all fields as inactive) but is not enforced in the back-end when action is taken. Because of this, a potential attacker can tamper with the request (activate the fields and modify their values) and modify the values in the back-end. To prevent this you definitely need to enforce the business logic that the customer must not be edited, in the back-end by checking if the action is valid (in this case don't change any values if he is in BLR 5). Additionally if somehow possible/feasible: Remove hidden fields for values which may not be edited, if the field is supposed to be "read only" then don't supply a form field for it in the first place...simple as that, thankfully Wicket will take care of this for you.

- Session Fixation[5]

  Call the replaceSession() method of the Wicket session after successful authentication. This will assign a new sessionId to the session of the recently authenticated user. This prevents the attacker from hijacking the session he previously created for use by the victim.

- CSRF[6]

  How to prevent: Use CryptedUrlWebRequestCodingStrategy to encrypt the URLs of the application per User. This makes it very unlikely for the attacker to create a valid and malicious request.

  ```
  @Override
      protected IRequestCycleProcessor newRequestCycleProcessor() {
      return new WebRequestCycleProcessor() {
          @Override
          protected IRequestCodingStrategy newRequestCodingStrategy() {
  ```

[1] https://www.owasp.org/index.php/SQL_Injection
[2] https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
[3] https://www.owasp.org/index.php/Top_10_2010-A3
[4] https://www.owasp.org/index.php/Top_10_2010-A4
[5] https://www.owasp.org/index.php/Session_fixation
[6] https://www.owasp.org/index.php/CSRF

```
            return new CryptedUrlWebRequestCodingStrategy(
                new WebRequestCodingStrategy());
            }
        };
    }
```

A.3 Logging

Mobiliser uses the “Simple Log For Java” (SLF4J) logging framework. It provides simple, fast, reliable, and OSGi enabled logging. SLF4J supports various backend implementations such as Java Util Logging, Log4J and Commons Logging.

A.3.1 Using SLF4J

This section describes how to use the SLF4J API in Java code and which dependencies are needed in the pom.xml.

The only dependency you need in the modules, is on the API bundle:

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>com.springsource.slf4j.api</artifactId>
</dependency>
```

For testing you may want to add the simple logger provided with SLF4J to actually get some log output during test cases:

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>com.springsource.slf4j.simple</artifactId>
  <scope>test</scope>
</dependency>
```

In the Java class implementation you first create a static final Logger instance via LoggerFactory, like you do it in Log4J. SLF4J uses different classes.

```
1 import org.slf4j.Logger;
2 import org.slf4j.LoggerFactory;
3
4 public class MyCustomClass {
5     private static final Logger LOG = LoggerFactory
6         .getLogger(MyCustomClass.class);
7 (...)
```

Listing A.1: Logger Initialization

You can now use the logger instance in the code to log out any log message. This is an simple example how to log a single String:

```
1 (...)
2     LOG.trace("Hello World");
3 (...)
```

Listing A.2: Simple Logging

SLF4J provides means to use placeholders in the log message, which are replaced dynamically.

```
1 (...)
```

```
2 public sayHello(final String firstname, final String lastname)
3     lOG.trace("Hello {} {}! How are you doing?", firstname, lastname);
4 (...)
```

Listing A.3: String Template Logging

If you need to log more than two parameters you can pass an Object[] array

```
1 (...)
2 public void authenticate(SubTransaction transaction, String otpStr,
3         boolean payer) throws AuthenticationException {
4
5     LOG.debug(
6         "#authenticate(subtransaction = {}, authToken = {}, isPayer = {})",
7         new Object[] { transaction, "xxxx",
8             Boolean.valueOf(payer) });
9     }
10 (...)
```

Listing A.4: String Template Logging 2

Note: You do not need to use the pattern $\mathrm{if(LOG.isDebugEnabled())}\ldots$ or similar when the log String is created by SLF4J. SLF4J will create the String only when it is logged. Hence it is also critical that you don't call the $\mathrm{toString}$ method explicitly, but pass in the objects and let the logging framework invoke $\mathrm{toString}$ only when the log message must be generated at all.

A.3.2 Log Level

Always make sure that you select the proper log level for your message. It is very important that the level is not too far off, otherwise you might hide problems from showing up in the log file or alarm people even if there was no reason for.

Level	Purpose
trace	Lengthy messages that are exchanged between systems but do not contain much useful information.
debug	enter method, leave method, result of an operation
info	unexpected result of an operation (but not a fault), also: result of main operation
warn	error condition that might get resolved without manual interaction, unknown behavior or return code when calling third party
error	error condition that most likely needs manual interaction to get resolved, e.g. connection lost permanently, configuration data missing / misconfiguration. Also use error for critical situation that needs immediate attention (e.g. connection to DB lost) [7]

Table A.1: Log Level

Most test systems are set to log level "debug". Most production systems are set to log level "debug" or "info".

A.3.3 Runtime Configuration

In an actual container environment, we use the Pax Logger implementation. Pax Logger implements the OSGi logging service and gives you access to the current container log through the Felix web console (just click the "logging service" tab). More important, it allows configuring the log levels through a log4j properties file and updates the logging framework dynamically on file changes.

The file $\mathrm{org.ops4j.pax.logging.cfg}$ must be located in the $./\mathrm{config}/\mathrm{cfgbackup}$ directory that is monitored by the ARF config admin file poller (an extension of Felix Fileinstall), so that the $\mathrm{ConfigAdmin}$ is initialized and updated properly. The content of this file must be conform to the standard Log4J property file layout.

Web Application Logging

To configure logging for web applications, we use a servlet context listener. Define the filename of the log4j XML or properties file in the web.xml by setting

```
  <listener>
    <display-name>Log4jConfiguration</display-name>
    <listener-class>
      com.sybase365.mobiliser.util.tools.wicketutils.logging.Log4jServletContext
Listener
    </listener-class>
  </listener>

  <context-param>
    <param-name>log4j.config</param-name>
    <param-value>web-ui_log4j.xml</param-value>
  </context-param>
```

The directory that hosts this file must be given by providing the system property to Tomcat/Jetty: -Dcom.sybase365.mobiliser.ui.web.logging.basedir=/tmp

Also, please include a reasonable sample configuration file in the web application's META-INF directory so that deployment teams have a template for configuration. The file should have the actual log file location configurable through system properties like this:

```
  <appender name="FILE" class="org.apache.log4j.FileAppender">
    <param name="File" value="\${log4j.logfiles.path}/web-ui.log" />
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d{ISO8601} [%t] %-5p %c:%L %x - %m%n" />
    </layout>
  </appender>
```

So that this system property available to Tomcat/Jetty can easily control the log file location:

-Dlog4j.logfiles.path=/tmp

A.3.4 Logging Patterns

If you catch an exception log something about it Why?: If it does not appear in the logfile an error is multiple times more harder to trace since the app shows an error but the log does not tell you what happened.

Provide logging of start and end of the method including all readable parameters When writing methods, provide logging of start and end of the method. This log should includes all readable parameters and, depending on the readability of the return type, the return parameter. Usually this should be logged as DEBUG.

A.4 Date and Timestamp Handling

SOAP requests will arrive in Mobiliser in the timezone of the client software. Taking the Java Wicket front-end using Sun's JAXB reference implementation and not explicitly setting a timezone on XmlGregorianCalendars, the following table shows how information will arrive:

The Mobiliser server will then correctly convert the dateTime value back into the same Unix timestamp.

xsd:dates represent midnight on a certain day in a certain timezone. We must be especially care when handling these values. For a customer using a registration frontend running in the timezones in the table, when entering this birthdate of 1970-01-01, the following information will be sent to Mobiliser:

Mobiliser will always parse out the Unix timestamp in the last column. So you see a different value is computed based on the time zone of the front end server. If we do not alter that value, hibernate will read the same Unix

TimeZone	UnixTimestamp	xsd:dateTime
America/Los_Angeles	1326155028	2012-01-09 16:23:48-0800
UTC	1326155028	2012-01-10 00:23:48+0000
Pacific/Auckland	1326155028	2012-01-10 13:23:48+1300

Table A.2: Date Example 1

TimeZone	Date	xsd:dateTime	UnixTimestamp
America/Los_Angeles	1970-01-01	1970-01-01-0800	28800
UTC	1970-01-01	1970-01-01+0000	0
Pacific/Auckland	1970-01-01	1970-01-01+1200	-43200

Table A.3: Date Example 2

timestamp value back from the database so that is fine. The problem occurs when we send that value back to the client. Since we do not know any longer which time zone that was, the server will always send back its time zone, so if the server is running in Europe/Berlin, it will send 1970-01-01+0100, which is of course a different Unix timestamp.

The recommendation is to discard the timestamp from incoming dates on the Mobiliser side and always save them in the server time zone. When sending values back out, our timezone will be appended, but recipients should then ignore this value and take it as its face value, simply that date.

A.4.1 Database

All our database columns representing some point in time are $\mathrm{TIMESTAMP}$s in Mobiliser. This data type associates no time zone with the saved data. So taking our Unix timestamp from above 1326155028, if Mobiliser is running with TZ=UTC, the value 2012-01-10 00:23:48 will be written to the database. Or if it is TZ=Pacific/Auckland, then we get 2012-01-10 13:23:48. When reading the value back out again, we will get the same Unix timestamp, but only if the server is still using the same timezone. So it is very important that all processing writing and reading to / from the database be in the same time zone and have their clocks synchronized through something like ntpd.

It is also important for the Mobiliser server to use the same time zone as the database server since there are still a couple of triggers which use $\mathrm{systimestamp}$.

A.4.2 Oracle Specifics

$\mathrm{SYSDATE}$ will return a value dependent on the timezone set in my session. $\mathrm{SYSTIMESTAMP}$ will always return a value in the time zone of the database server (its datatype is $\mathrm{TIMESTAMP\ WITH\ TIMEZONE}$). I think it then best to always use $\mathrm{SYSTIMESTAMP}$ in any triggers or procedures and then ensure the database and application server have the same time zone setting.

A.4.3 Non UTC Timezones

It should be noted that there is a loss of data when using a Mobiliser backend not configured for UTC. During DST crossovers there will be records actually an hour apart, but upon reading the values back from the database, they will be equivalent. When SOAP requests come in querying data, Mobiliser will fetch these timestamps and convert them based on the timezone of the server. Since we do not set a specific time zone on the $\mathrm{XmlGregorianCalendar}$, Mobiliser will use the time zone of the server to convert the Unix timestamp into a string. The receiving end will parse that and will get the same Unix timestamp value.

It is our recommendation that all new Mobiliser installations along with the database be run in UTC.

A.4.4 Display

The Wicket frontend must never format dates with a date format object without first setting a timezone on it, otherwise the date will be formatted in the time zone of the front end server. For this case we have the time zone setting on the customer itself which is returning from the $\mathrm{loginCustomer}$ call. Of course if the customer is currently traveling, he will see timestamps in his home timezone and not where he is currently accessing his account.

The same goes when inputting data. If a customer is searching for transactions between 23 Dec and 29 Dec 2011 and the customer has the timezone America/New_York, those values must be sent to the backend server as 2011-12-23 00:00:00-05:00 and 2011-12-29 23:59:59-05:00. So it is important to parse those fields with the timezone of the current customer.

When processing a customer's birth date, it doesn't matter which timezone you use to parse the field, you just must ensure you set that same timezone on the $\mathrm{XmlGregorianCalendar}$. So if you parse it with the customer's timezone then you must also set that on the $\mathrm{XmlGregorianCalendar}$. If you do not set a timezone when parsing which causes you to parse with the server timezone, do not set a value on the $\mathrm{XmlGregorianCalendar}$ since it will then also get the server timezone by default.

When displaying date to the customer, you must respect the timezone value on the $\mathrm{XmlGregorianCalendar}$, otherwise your date could shift. This is because the dates are sent as 2011-12-23-05:00. So either do not convert into a Unix timestamp(java date) and back (so just simply use the text) or if you must convert to a java date, you must reuse the time zone the server sent you.

Appendix B

Troubleshooting

This chapter describes some common error situations and how to analyze and resolve them.

B.1 Mobiliser Container

B.1.1 Imports/Exports

This section describes general errors that con occur in any bundle, mainly due to incorrect bundle or blueprint configuration.

BundleException “Unresolved constraint”

Felix.out:

```
Auto-deploy start:  org.osgi.framework.BundleException:  Unresolved constraint in bundle
com.sybase365.mobiliser.custom.project.services.endpoint [156]:
Unable to resolve 156.0:  missing requirement [156.0] osgi.wiring.package;
(&(osgi.wiring.package=com.sybase365.mobiliser.custom.project.businesslogic)
(version>=1.0.0) (!(version>=2.0.0)))
```

This error indicates that the container fails to resolve an Import-Package statement on the bundle that is currently starting. In this case the bundle
com.sybase365.mobiliser.custom.project.services.endpoint
has an import on the package
com.sybase365.mobiliser.custom.project.businesslogic
that cannot be fulfilled.

The package should come from another bundle The bundle exporting the “com.sybase365.mobiliser.custom.project.services.endpoint” is missing from the container configuration. Or the other bundle has a missing Export-Package declaration.

The package is in the bundle itself Make sure that the package is either listed in the Export-Packages or Private-Packages of the project’s POM, otherwise, the build process will not include the package into the artifact and instead try to resolve it from another external bundle.

The package should not be part of the required imports The '*' wildcard in the Maven bundle plugin configuration for Import-Packages tries to resolve any packages being used not only in the Java import declarations of classes, but also inspects the Spring XML configuration files of your bundle. In case you have a typo in a bean declaration and reference a non-existing package/class, this will also result into the package being listed in the bundle's Import-Packages.

B.1.2 Blueprint

ApplicationContextException

bundle-context.xml

```
2012-01-30 13:54:26,228 [Spring DM Context Creation Timer] WARN org.eclipse.gemini.blueprint.extender.internal.
dependencies.startup.DependencyWaiterApplicationContextExecutor - Timeout occurred before finding service
dependencies for [OsgiBundleXmlApplicationContext(bundle=
com.sybase365.mobiliser.custom.project.services.endpoint , config=osgibundle:/META-INF/spring/*.xml)]
2012-01-30 13:54:26,228 [Spring DM Context Creation Timer] ERROR org.eclipse.gemini.blueprint.extender.internal.
activator.ContextLoaderListener - Application context refresh failed (OsgiBundleXmlApplicationContext(bundle=
com.sybase365.mobiliser.custom.project.services.endpoint ,
config=osgibundle:/META-INF/spring/*.xml)) org.springframework.context.ApplicationContextException:
Application context initialization for
' com.sybase365.mobiliser.custom.project.services .endpoint '
has timed out waiting for (&(objectClass=
com.sybase365.mobiliser.money.businesslogic.DummyAction ))
at org.eclipse.gemini.blueprint.extender.internal.dependencies.startup.DependencyWaiterApplicationContext
Executor.timeout (DependencyWaiterApplicationContextExecutor.java:489)
```

Means that a blueprint osgi:reference could not be fulfilled within the given timeout. A Bundle with unfulfilled osgi:references will not start and will not provide any osgi:services by itself, causing all dependent bundles to be in a pending state as well. Unfortunately this does not show up in the system/console, all bundles are still in status "active". This error will only show **after 15(!) minutes** in the mobiliser.log file.

In the example above, the bundle
com.sybase365.mobiliser.custom.project.services.endpoint
was not able to find an OSGi service implementing the interface
com.sybase365.mobiliser.money.businesslogic.DummyAction .

Either the service bundle is missing, the OSGi service is not exported or there is a filer criteria that can not be fulfilled. Check bundle-config.xml of the client bundle (osgi:reference) and the bundle-config.xml of the service bundle (osgi:service). If everything is correct, the service bundle might not have been started correctly (look for other errors / exceptions in the log files).

B.1.3 Service Endpoint

NullPointerException

```
2012-01-29 17:02:23,445 [qtp739590180-44] WARN  org.eclipse.jetty.util.log - /mo
biliser/xcustom/XCustom.wsdl
org.springframework.xml.xsd.commons.CommonsXsdSchemaException: Schema [class pat
h resource [com/sybase365/mobiliser/money/contract/transaction/xsd/requests-5-0
.xsd]] could not be loaded; nested exception is java.lang.RuntimeException: java
.lang.NullPointerException
        at org.springframework.xml.xsd.commons.CommonsXsdSchemaCollection.afterP
ropertiesSet(CommonsXsdSchemaCollection.java:156)
        at com.sybase365.mobiliser.framework.gateway.soap.ws.wsdl.DynamicWsdl11D
efinition.getSchemaCollection(DynamicWsdl11Definition.java:271)
        at com.sybase365.mobiliser.framework.gateway.soap.ws.wsdl.DynamicWsdl11D
efinition.instantiate(DynamicWsdl11Definition.java:321)
        at com.sybase365.mobiliser.framework.gateway.soap.ws.wsdl.DynamicWsdl11D
efinition.getSource(DynamicWsdl11Definition.java:292)
```

The problem is most likely that not all required XSDs are listed in the Endpoint configuration in the bundle-context*.xml file. Please make sure that all XSD files are listed (in the correct order) under “schemaPaths”. If there are cyclic dependencies, they must be resolved in the XSDs! The package names of the generates classes must also be listed under “contextPaths”. In addition the dependencies must be entered in the pom.xml and the XSD and JAXB package names must be included in the Import-Packages section of the maven-bundle-plugin. Please investigate all import statements in the XSD documents to compile the full list of XSDs **and the order in which they are loaded**.

```
1  <bean id="custom1Endpoint"
2    class="com.sybase365.mobiliser.ws.api.impl.EndpointInformationImpl">
3    <property name="endpoint" ref="customImplComplete"/>
4    <property name="schemaPaths">
5      <list>
6        <value>/com/sybase365/mobiliser/framework/contract/xsd/base-5-0.xsd</
             value>
7        <value>/com/sybase365/mobiliser/money/contract/custom/xsd/beans-5-0.xsd</
             value>
8        <value>/com/sybase365/mobiliser/money/contract/custom/xsd/custom1-
             requests-5-0.xsd</value>
9      </list>
10   </property>
11   <property name="allowedMessageElements">
12     <list>
13       <value>^.*Custom1.*$</value>
14     </list>
15   </property>
16   <property name="contextPaths">
17     <list>
18       <value>com.sybase365.mobiliser.framework.contract.v5_0.base</value>
19       <value>com.sybase365.mobiliser.money.contract.v5_0.custom</value>
20       <value>com.sybase365.mobiliser.money.contract.v5_0.custom.beans</value>
21     </list>
22   </property>
23 </bean>
```

Listing B.1: “OSGI Configuration” OSGI Context Configuration

B.1 is a typical example: The custom1-requests-5-0.xsd imports beans-5-0.xsd and both import the base definition from base-5-0.xsd

CommonsXsdSchemaException

Mobiliser.log.<context>:

```
2012-01-30 11:45:16,020 [qtp2008927154-49] DEBUG com.sybase365.mobiliser.framewo
rk.gateway.servlet.OSGiDispatcherServlet S:() - Could not complete request
org.springframework.xml.xsd.commons.CommonsXsdSchemaException: Schema [class pat
h resource [com/sybase365/mobiliser/money/contract/customer/xsd/beans-5-0.xsd]]
could not be loaded; nested exception is org.apache.ws.commons.schema.XmlSchemaE
xception: Schema name conflict in collection. Namespace: http://mobiliser.sybase
365.com/money/contract/v5_0/customer/beans
        at org.springframework.xml.xsd.commons.CommonsXsdSchemaCollection.afterP
ropertiesSet(CommonsXsdSchemaCollection.java:156)
        at com.sybase365.mobiliser.framework.gateway.soap.ws.wsdl.DynamicWsdl11D
efinition.getSchemaCollection(DynamicWsdl11Definition.java:271)
```

The problem is most likely that the order of the XSDs in the Endpoint configuration in the bundle-context*.xml file is not correct. Please make sure that all XSD files are listed (in the correct order) under “schemaPaths”. If there are cyclic dependencies, they must be resolved in the XSDs! Figuring out the correct order requires to look at each XSD individually, if you find a n import statements to another XSD; the imported one must be listed first!

IllegalArgumentException

Mobiliser.log:

```
2012-01-30 12:12:36,774 [qtp939655043-38] WARN  org.eclipse.jetty.util.log - /mo
biliser/custom/Custom.wsdl
java.lang.IllegalArgumentException: class path resource [com/sybase365/mobiliser
/custom/project/services/contract/blacklist/xsd/beans-5-0.xsd] does not exit
        at org.springframework.util.Assert.isTrue(Assert.java:65)
        at org.springframework.xml.xsd.commons.CommonsXsdSchemaCollection.afterP
ropertiesSet(CommonsXsdSchemaCollection.java:144)
        at com.sybase365.mobiliser.framework.gateway.soap.ws.wsdl.DynamicWsdl11D
efinition.getSchemaCollection(DynamicWsdl11Definition.java:271)
        at com.sybase365.mobiliser.framework.gateway.soap.ws.wsdl.DynamicWsdl11D
efinition.instantiate(DynamicWsdl11Definition.java:321)
        at com.sybase365.mobiliser.framework.gateway.soap.ws.wsdl.DynamicWsdl11D
efinition.getSource(DynamicWsdl11Definition.java:292)
```

The XSD location (package) is most likely missing or wrong in the Import-Package section of the pom.xml file. The XSD file is not found when trying to construct the WSDL. Example:

```
1       <plugin>
2         <groupId>org.apache.felix</groupId>
3         <artifactId>maven-bundle-plugin</artifactId>
4 ...
5             <Import-Package>
6 ...
7               com.sybase365.mobiliser.framework.contract.xsd,
8               com.sybase365.mobiliser.custom.project.services.contract.blacklist.
                    xsd,
9 ...
```

Listing B.2: pom.xml

B.1.4 Events

Unique Constraint Violation

These log messages show up when there are more than one container running on the same database. The database is used to synchronize event processing information with multiple containers running. Some events get seen by both handlers. Whoever acquires the event first processes it, the other one does get this error. Unfortunately, Hibernate logs this on error level, but this is not an error condition and there is nothing we can do about it.

```
[EvntActnPrfrmr-105] WARN  org.hibernate.util.JDBCExceptionReporter - SQL Error: 1, SQLState: 23000
[EvntActnPrfrmr-105] ERROR org.hibernate.util.JDBCExceptionReporter - ORA-00001: unique constraint
      (MOBR5.IDX_ENTRY_HANDLER) violated
[EvntActnPrfrmr-105] ERROR org.hibernate.event.def.AbstractFlushingEventListener - Could not synchronize
      database state with session
org.hibernate.exception.ConstraintViolationException: Could not execute JDBC batch update
    at org.hibernate.exception.SQLStateConverter.convert(SQLStateConverter.java:94)
    ...
    at com.sybase365.mobiliser.framework.event.persistence.impl.DBStoreProvider.getHandlerProcessLock
      (DBStoreProvider.java:343)
    ...
    at com.sybase365.mobiliser.framework.event.registration.Listener.doAction(Listener.java:150)
    at com.sybase365.mobiliser.framework.event.process.EventActionPerformer.doThreadAction
      (EventActionPerformer.java:49)
    at com.sybase365.mobiliser.framework.event.process.pool.PoolThread.run(PoolThread.java:107)
Caused by: java.sql.BatchUpdateException: ORA-00001: unique constraint (MOBR5.IDX_ENTRY_HANDLER) violated
    at oracle.jdbc.driver.OraclePreparedStatement.executeBatch(OraclePreparedStatement.java:10296)
    at oracle.jdbc.driver.OracleStatementWrapper.executeBatch(OracleStatementWrapper.java:216)
    at com.jolbox.bonecp.StatementHandle.executeBatch(StatementHandle.java:469)
    at org.hibernate.jdbc.BatchingBatcher.doExecuteBatch(BatchingBatcher.java:70)
```

```
    at org.hibernate.jdbc.AbstractBatcher.executeBatch(AbstractBatcher.java:268)
    ... 84 more
[EvntActnPrfrmr-105] WARN  com.sybase365.mobiliser.framework.event.registration.Listener -
      doAction([ClearingEvent,63152] [Delay: 2000 millisecond(s) @ Wed May 02 15:01:02 MDT 2012]
      [BatchRunIds=500056268]) -> Couldn't get a handler process lock -
      another instance maybe got it ahead of me
```

B.2 Gogo Shell

Felix GoGo Shell allows command line access to the OSGi framework.

The GoGo shell is installed in the standard vanilla container. It is accessible through telnet on port 5356. Telnet doesn't use readline so you have no history and can't use the arrow keys, Ctrl-A, Ctrl-E to navigate which makes it somewhat of a pain. You can use readline wrap to alleviate this and also get command history along with it for free.[1]

Note: Do not use $\mathrm{exit}\ 0$ to leave the console. That will send the exit command to the framework (shutdown command). Just close the telnet session (Ctrl-D)

B.2.1 Tips and Tricks

You can do many things in the console like starting, stopping bundles, inspecting manifests etc.

List commands

```
g! help
```

Show help for command

```
g! help <command>
```

Find bundles named authentication

```
g! lb | grep -i authentication
  149|Active     |    15|AIMS Mobiliser :: Framework :: Gateway Au
thentication Dao (5.0.0.SNAPSHOT)
  156|Active     |    15|AIMS Mobiliser :: Money :: Business Logic Authentication
   API (5.0.0.SNAPSHOT)
  157|Active     |    15|AIMS Mobiliser :: Money :: Business Logic Authentication
   Handler No-Op (5.0.0.SNAPSHOT)
  158|Active     |    15|AIMS Mobiliser :: Money :: Business Logic Authentication
   Handler OTP (5.0.0.SNAPSHOT)
  159|Active     |    15|AIMS Mobiliser :: Money :: Business Logic Authentication
   Handler PIN (5.0.0.SNAPSHOT)
  160|Active     |    15|AIMS Mobiliser :: Money :: Business Logic Authentication
   Handler SMS AOC (5.0.0.SNAPSHOT)
  161|Active     |    15|AIMS Mobiliser :: Money :: Business Logic Authentication
   Impl (5.0.0.SNAPSHOT)
  219|Active     |    15|AIMS Mobiliser :: Money :: Service Endpoint Authenticati
  on Continue (5.0.0.SNAPSHOT)
```

Find bundles with authentication in the jar name

```
g! lb -l | grep -i authentication
  149|Active     |    15|file:/home/aclemons/workspace\_m5/vanilla/postgresql\_st
andalone/container/bundles/15-mobiliser-money-businesslogic-services/com.sybase3
65.mobiliser.framework.gateway.security.authentication.dao-5.0.0-SNAPSHOT.jar
  150|Active     |    15|file:/home/aclemons/workspace\_m5/vanilla/postgresql\_st
andalone/container/bundles/15-mobiliser-money-businesslogic-services/com.sybase3
65.mobiliser.framework.gateway.security.authentication.webconsole-5.0.0-SNAPSHOT
```

[1] Linux Users: A quick alias in your .bashrc makes things simple:
`alias gogo="rlwrap -c telnet localhost 5356"`

```
.jar
  156|Active     |    15|file:/home/aclemons/workspace\_m5/vanilla/postgresql\_st
andalone/container/bundles/15-mobiliser-money-businesslogic-services/com.sybase3
65.mobiliser.money.businesslogic.authentication.api-5.0.0-SNAPSHOT.jar
  157|Active     |    15|file:/home/aclemons/workspace\_m5/vanilla/postgresql\_st
andalone/container/bundles/15-mobiliser-money-businesslogic-services/com.sybase3
65.mobiliser.money.businesslogic.authentication.auth-handlers.noop-5.0.0-SNAPSHO
T.jar
  158|Active     |    15|file:/home/aclemons/workspace\_m5/vanilla/postgresql\_st
andalone/container/bundles/15-mobiliser-money-businesslogic-services/com.sybase3
65.mobiliser.money.businesslogic.authentication.auth-handlers.otp-5.0.0-SNAPSHOT
.jar
  159|Active     |    15|file:/home/aclemons/workspace\_m5/vanilla/postgresql\_st
andalone/container/bundles/15-mobiliser-money-businesslogic-services/com.sybase3
65.mobiliser.money.businesslogic.authentication.auth-handlers.pin-5.0.0-SNAPSHOT
.jar
  160|Active     |    15|file:/home/aclemons/workspace\_m5/vanilla/postgresql\_st
andalone/container/bundles/15-mobiliser-money-businesslogic-services/com.sybase3
65.mobiliser.money.businesslogic.authentication.auth-handlers.smsaoc-5.0.0-SNAPS
HOT.jar
  161|Active     |    15|file:/home/aclemons/workspace\_m5/vanilla/postgresql\_st
andalone/container/bundles/15-mobiliser-money-businesslogic-services/com.sybase3
65.mobiliser.money.businesslogic.authentication.impl-5.0.0-SNAPSHOT.jar
  219|Active     |    15|file:/home/aclemons/workspace\_m5/vanilla/postgresql\_st
andalone/container/bundles/15-mobiliser-money-businesslogic-services/com.sybase3
65.mobiliser.money.services.authenticationcontinue-5.0.0-SNAPSHOT.jar
  304|Active     |    11|file:/home/aclemons/workspace\_m5/vanilla/postgresql\_st
andalone/container/bundles/11-mobiliser-framework/com.sybase365.mobiliser.framew
ork.gateway.security.authentication.jmx-5.0.0-SNAPSHOT.jar
```

Find bundles which are not active

```
g! lb | grep -v Active
START LEVEL 50
   ID|State      |Level|Name
  110|Resolved   |    18|AIMS Mobiliser :: 3rd Party :: Barcode Library (1.0.1)
  271|Resolved   |     1|JBoss Hibernate Annotations (3.4.0.GA-A)
  272|Resolved   |     1|AIMS Mobiliser :: Framework :: Persistence Hibernate-ANT
LR Bridge (5.0.0.SNAPSHOT)
  273|Resolved   |     1|AIMS Mobiliser :: Framework :: Persistence Hibernate-Dat
abase Driver Bridge (5.0.0.SNAPSHOT)
  274|Resolved   |     1|AIMS Mobiliser :: Framework :: Persistence BoneCP-Databa
se Driver Bridge (5.0.0.SNAPSHOT)
  275|Resolved   |     1|AIMS Mobiliser :: Framework :: Service Configuration (5.
0.0.SNAPSHOT)
  276|Resolved   |     1|AIMS Mobiliser :: Utilities :: Tools - Logging (5.0.0.SN
APSHOT)
  277|Resolved   |     1|AIMS Mobiliser :: Utilities :: Tools - Spring Extender C
onfiguration Fragment (5.0.0.SNAPSHOT)
  278|Resolved   |     1|Spring Web Services Support (2.0.0.RELEASE)
  322|Resolved   |    10|AIMS Mobiliser :: Framework :: Persistence Hibernate DB
Dialect (5.0.0.SNAPSHOT)
```

Show services exported by bundle and who's using them

```
g! inspect 'capability' 'service' 205
com.sybase365.mobiliser.money.jobs.event.handler.transaction-notification [205]
provides:
------------------------------------------------------------------------------
----------
service; org.eclipse.gemini.blueprint.context.DelegatedExecutionOsgiBundleApplic
ationContext, org.eclipse.gemini.blueprint.context.ConfigurableOsgiBundleApplica
tionContext, org.springframework.context.ConfigurableApplicationContext, org.spr
ingframework.context.ApplicationContext, org.springframework.context.Lifecycle,
org.springframework.beans.factory.ListableBeanFactory, org.springframework.beans
.factory.HierarchicalBeanFactory, org.springframework.context.MessageSource, org
.springframework.context.ApplicationEventPublisher, org.springframework.core.io.
support.ResourcePatternResolver, org.springframework.beans.factory.BeanFactory,
org.springframework.core.io.ResourceLoader, org.springframework.beans.factory.Di
sposableBean with properties:
   Bundle-SymbolicName = com.sybase365.mobiliser.money.jobs.event.handler.transa
ction-notification
   Bundle-Version = 5.0.0.SNAPSHOT
```

```
   org.eclipse.gemini.blueprint.context.service.name = com.sybase365.mobiliser.m
oney.jobs.event.handler.transaction-notification
   org.springframework.context.service.name = com.sybase365.mobiliser.money.jobs
.event.handler.transaction-notification
   service.id = 553
service; com.sybase365.mobiliser.framework.event.model.Handler with properties:
   Bundle-SymbolicName = com.sybase365.mobiliser.money.jobs.event.handler.transa
ction-notification
   Bundle-Version = 5.0.0.SNAPSHOT
   org.eclipse.gemini.blueprint.bean.name = transactionNotificationEventHandler
   org.springframework.osgi.bean.name = transactionNotificationEventHandler
   osgi.service.blueprint.compname = transactionNotificationEventHandler
   service.id = 699
   Used by:
      com.sybase365.mobiliser.framework.event.core [279]
```

Show services required by bundle and who exported them

```
g! inspect 'requirement' 'service' 161
com.sybase365.mobiliser.money.businesslogic.authentication.impl [161] requires:
--------------------------------------------------------------------------------
service; org.osgi.service.packageadmin.PackageAdmin provided by:
   org.apache.felix.framework [0]
service; org.xml.sax.EntityResolver provided by:
   org.eclipse.gemini.blueprint.extender [72]
service; org.springframework.beans.factory.xml.NamespaceHandlerResolver provided
 by:
   org.eclipse.gemini.blueprint.extender [72]
service; com.sybase365.mobiliser.money.businesslogic.authentication.api.IAuthenti
cationHandler provided by:
   com.sybase365.mobiliser.money.businesslogic.authentication.auth-handlers.noop
[157]
service; com.sybase365.mobiliser.money.businesslogic.authentication.api.IAuthent
icationHandler provided by:
   com.sybase365.mobiliser.money.businesslogic.authentication.auth-handlers.smsa
oc [160]
service; com.sybase365.mobiliser.money.businesslogic.authentication.api.IAuthent
icationHandler provided by:
   com.sybase365.mobiliser.money.businesslogic.authentication.auth-handlers.otp
[158]
service; com.sybase365.mobiliser.money.businesslogic.authentication.api.IAuthent
icationHandler provided by:
   com.sybase365.mobiliser.money.businesslogic.authentication.auth-handlers.pin
[159]
```

Appendix C

Use Third Party Libraries in OSGi Bundles

Customization often have third party libraries dependencies. This chapter describes how to include those third party dependencies in various ways. It starts with the straight forward approach when this library is available as OSGi version. The second section shows you how to add this as embedded library. The last section covers the process how to build an OSGi bundle out of an existing library.

C.1 Search for OSGi Libraries

OSGi is support is rapidly growing and widely used. More and more libraries are published OSGi enabled. Therefore you will easily find a new OSGi enabled version of common libraries.

Other common libraries are already been wrapped into OSGi bundle libraries by other project teams and provided free. Those library can be found via search services.

SpringSource Enterprize Bundle Repository The SpringSource Enterprise Bundle Repository[1] provides a huge reference for OSGi bundles.

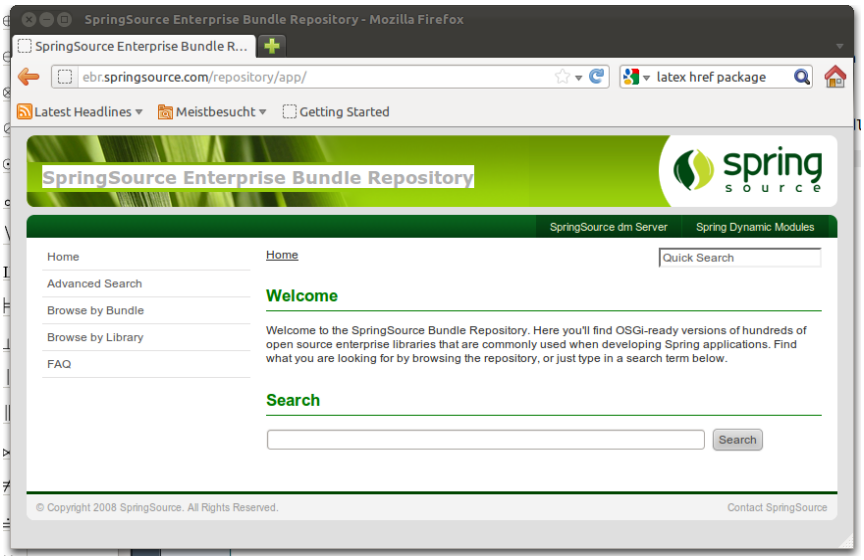

Figure C.1: SpringSource Enterprise Bundle Repository Search Page

[1] http://ebr.springsource.com/repository/app/

Simply type in the library name you are looking for and SpringSource will list all available version of it. It also provides a link to already OSGi enabled libraries, that wraps non OSGi libraries.

Apache ServiceMix 4.0 Apache ServiceMix 4.0[2] also provides a list of OSGi enabled bundles. Even if the list is shorter, you will find here more recent libraries compared to SpringSource.

C.2 Embed Third Party Libraries

If you can not find a OSGi version of an third party library, you may embed this to the bundle.

Note: Embedding third party libraries this way is not what the OSGi framework intends: Modular bundles. It also is limit to libraries that are used in this bundle and not exported in any way. Last, but not least you may end up to add additional libraries when they are used by the 3rd party library.

The following information is copied from the Apache Felix plug-in for Maven[3].

The Maven Bundle plug-in supports embedding of selected project dependencies inside the bundle by using the <Embed-Dependency> instruction:

```
1 <Embed-Dependency>dependencies</Embed-Dependency>
```

Listing C.1: POM Tag For Embedded Dependency Definition

where:

```
1 dependencies ::= clause ( ',' clause ) *
2 clause ::= MATCH ( ';' attr '=' MATCH | ';inline=' inline )
3 attr ::= 'groupId' | 'artifactId' | 'version' | 'scope' | 'type' | 'classifier' |
      'optional'
4 inline ::= 'true' | 'false' | PATH ( '|' PATH ) *
5 MATCH ::= <globbed regular expression>
6 PATH ::= <Ant-style path expression>
```

Listing C.2: Definition of the Embed-Dependency Tag

The plugin uses the <Embed-Dependency> instruction to transform the project dependencies into <Include-Resource> and <Bundle-ClassPath> clauses, which are then appended to the current set of instructions and passed onto BND. If you want the embedded dependencies to be at the start or middle of <Include-Resource> or <Bundle-ClassPath> then you can use {maven-dependencies}, which will automatically expand to the relevant clauses.

The MATCH section accepts alternatives, separated by |, and can be negated by using ! at the beginning of the MATCH. Use * to represent zero or more unknown characters and ? to represent zero or one character. You can also use standard Java regexp constructs. There is no need to escape the . character inside MATCH. The first MATCH in a clause will filter against the artifactId.

some examples:

```
1 <!-- embed all compile and runtime scope dependencies -->
2 <Embed-Dependency>*;scope=compile|runtime</Embed-Dependency>
3
4 <!-- embed any dependencies with artifactId junit and scope runtime -->
5 <Embed-Dependency>junit;scope=runtime</Embed-Dependency>
6
7 <!-- inline all non-pom dependencies, except those with scope runtime -->
8 <Embed-Dependency>*;scope=!runtime;type=!pom;inline=true</Embed-Dependency>
9
```

[2] http://servicemix.apache.org/SMX4/bundles-repository.html
[3] http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html

```
<!-- embed all compile and runtime scope dependencies, except those with
     artifactIds in the given list -->
<Embed-Dependency>*;scope=compile|runtime;inline=false;artifactId=!cli|lang|
    runtime|tidy|jsch</Embed-Dependency>

<!-- inline contents of selected folders from all dependencies -->
<Embed-Dependency>*;inline=images/**|icons/**</Embed-Dependency>
```

Listing C.3: Embed-Dependency Examples

Examples of using{maven-dependencies}:

```
<Include-Resource>
  {maven-resources}, {maven-dependencies},
  org/foo/Example.class=target/classes/org/foo/Example.class
</Include-Resource>

<Bundle-ClassPath>.,{maven-dependencies},some.jar</Bundle-ClassPath>
```

Listing C.4: Embed-Dependency Examples

By default matched dependencies are embedded in the bundle as artifactId-version.jar. This behavior can be modified using the following instructions:

```
    <Embed-StripVersion>true</Embed-StripVersion> - removes the version from the
        file (ie. artifactId.jar)
    <Embed-StripGroup>false</Embed-StripGroup> - adds the groupId as a
        subdirectory (ie. groupId/artifactId-version.jar)
    <Embed-Directory>directory</Embed-Directory> - adds a subdirectory (ie.
        directory/artifactId-version.jar)
```

Listing C.5: Embed-Dependency Strip Version Example

Normally the plugin only checks direct dependencies, but this can be changed to include the complete set of transitive dependencies with the following option:

```
<Embed-Transitive>true</Embed-Transitive>
```

Listing C.6: Transitive Embed-Dependency

If you want a dependency inlined instead of embedded add the inline=true. For example to inline all compile and runtime scoped dependencies use:

```
<Embed-Dependency>*;scope=compile|runtime;inline=true</Embed-Dependency>
```

Listing C.7: Inline Embed-Dependency Examples

Embed-Dependency and Export-Package If you embed a dependency with <Embed-Dependency>, and your <Export-Package> or <Private-Package> instructions match packages inside the embedded jar, you will see some duplication inside the bundle. This is because the <Export-Package> and <Private-Package> instructions will result in classes being inlined in the bundle, even though they also exist inside the embedded jar. If you want to export packages from an embedded dependency without such duplication then you can either inline the dependency, or use a new BND instruction called <_exportcontents>.

<_exportcontents> behaves just like Export-Package, except it doesn't change the content of the bundle, just what content should be exported.

C.3 Create an OSGi Version of an Third Party Library

This section describes the elegant way to bundle a third party non OSGi enabled library into a OSGi bundle. The section is an example how to build a OSGi version of the Sybase jConnect JDBC driver.

The actual process uses the Apache Felix Plug In for Maven[4]. This time it is used to construct a OSGi Manifest for a third party library with Peter Kriens BND tool[5].

C.3.1 Define Import and Export Packages

The actual build process uses the BND tool integrated in the Apache Felix Plug In for Maven. Each OSGi Manifest file defines a list of package exports, imports and internal libraries. These Manifest list are:

Export-Package: This is the list of packages that are exported by the OSGi bundle and available for other bundle. The attribute uses defines a list of packages that are required for this package.

Import-Package: This is a list of packages that are needed by the library. The attribute optional marks this package as optional and the bundle is loaded by the OSGi framework even if the required bundle does not exist.

Private-Package: This is a list of internal packages that are neither imported nor exported.

This is the Authentication API bundle Manifest for Example:

```
1  Manifest-Version: 1.0
2  Bnd-LastModified: 1330696374646
3  Build-Jdk: 1.7.0_02
4  Built-By: dirkg
5  Bundle-Category: businesslogic
6  Bundle-Description: Parent Project for all Mobiliser Projects
7  Bundle-ManifestVersion: 2
8  Bundle-Name: AIMS Mobiliser :: Money :: Business Logic Authentication AP
9   I
10 Bundle-SymbolicName: com.sybase365.mobiliser.money.businesslogic.authent
11  ication.api
12 Bundle-Version: 5.0.0.SNAPSHOT
13 Created-By: Apache Maven Bundle Plugin
14 Export-Package: com.sybase365.mobiliser.money.businesslogic.authenticati
15  on.api;uses:="com.sybase365.mobiliser.money.businesslogic.authenticatio
16  n.api.exception,com.sybase365.mobiliser.money.persistence.model.transac
17  tion,com.sybase365.mobiliser.framework.service.api,com.sybase365.mobili
18  ser.money.businesslogic.authentication.api.beans";version="5.0.0.SNAPSH
19  OT",com.sybase365.mobiliser.money.businesslogic.authentication.api.bean
20  s;version="5.0.0.SNAPSHOT",com.sybase365.mobiliser.money.businesslogic.
21  authentication.api.exception;uses:="com.sybase365.mobiliser.framework.s
22  ervice.api";version="5.0.0.SNAPSHOT"
23 Import-Package: com.sybase365.mobiliser.framework.service.api;version="[
24  5.0,6)",com.sybase365.mobiliser.money.persistence.model.transaction;ver
25  sion="[5.0,6)"
```

Listing C.8: MANIFEST.MF of the Authentication API

So the first task is to examine the library and define the imports and exports. The project POM will load this from the text file osgi.bnd, which is now explained in details.

[4] http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html
[5] http://www.aqute.biz/Bnd/Bnd

Bundle Import

The Import-Package lists all required packages of an OSGi bundle. This will also includes Java base bundles that are not part of the Java Runtime library.

The jConnect JDBC driver therefore also lists the the JNDI packages javax.naming.* packages in the import. If a package is used optional, it can be marked as optional and the OSGi container will load and start the bundle even if this package is not available.

This is the bundle import definition for the jConnect library:

```
1  Import-Package: com.sybase.jdbcx, \
2      com.sybase.jdbc4.jdbc, \
3      com.sybase.jdbc4.jdbc.resource, \
4      com.sybase.jdbc4.a.b, \
5      com.sybase.jdbc4.a.a, \
6      com.sybase.jdbc4.a.b.a, \
7      com.sybase.jdbc4.tds, \
8      com.sybase.jdbc4.timedio, \
9      com.sybase.jdbc4.security.asn1;resolution:=optional, \
10     javax.naming, \
11     javax.naming.directory, \
12     javax.naming.spi, \
13     javax.crypto, \
14     javax.security.auth, \
15     javax.transaction.xa, \
16     javax.sql, \
17     javax.net, \
18     javax.net.ssl;resolution:=optional, \
19     org.ietf.jgss;resolution:=optional, \
20     sun.io;resolution:=optional, \
21     foo.bar;resolution:=optional, \
22     xxxxxxxxxxxxxxxxxxxxxxxxx;resolution:=optional
```

Listing C.9: jConnect Import Packages

The last two lines are markers only which will never be resolved by the OSGi container. The com.sybase* packages are self imports because they are exported too.

Bundle Export

The bundle export list defines the visible/exported packages by the library. Each package also defines which packages are used by the packages. These packages must be part of the bundle export definition.

This is the bundle export definition for the jConnect library. You will notice that some packages uses other exported packages.

```
1  Export-Package: com.sybase.jdbcx;uses:="javax.naming,javax.sql", \
2      com.sybase.jdbc4.jdbc;uses:="com.sybase.jdbcx,javax.naming,com.sybase.jdbc4.
           utils,org.ietf.jgss,javax.naming.spi,javax.naming.directory,javax. sql,
           com.sybase.jdbc4.tds,javax.transaction.xa",com.sybase.jdbc4.utils;uses:="
           com.sybase.jdbcx,com.sybase.jdbc4.jdbc,com.sybase.jdbc4.timedio", \
3      com.sybase.jdbc4.utils.resource, \
4      com.sybase.jdbc4.a.b;uses:="com.sybase.jdbc4.a.a,com.sybase.jdbc4.security.
           asn1", \
5      com.sybase.jdbc4.a.a, \
6      com.sybase.jdbc4.a.b.a;uses:="com.sybase.jdbc4.a.b", \
7      com.sybase.jdbc4.tds;uses:="com.sybase.jdbc4.timedio,com.sybase.jdbc4.jdbc,
           com.sybase.jdbc4.utils,sun.io,javax.net.ssl,com.sybase.jdbcx,com.sybase.
           jdbc4.a.b,com.sybase.jdbc4.a.a,com.sybase.jdbc4.a.b.a,javax.security.auth
           ,org.ietf.jgss,javax.crypto,foo.bar,javax.net", \
8      com.sybase.jdbc4.jdbc.resource, \
```

```
9     com.sybase.jdbc4.timedio;uses:="com.sybase.jdbcx,com.sybase.jdbc4.tds,com.
          sybase.jdbc4.jdbc,com.sybase.jdbc4.utils"
```

Listing C.10: jConnect Export Packages

Private Bundles

The private bundle lists all internal bundles that are neither exported not imported. The Maven plug in makes the life easier and creates this list by its own. You simply need to specify the following private package list and the rest is done by the plug in:

```
1 Private-Package: !*, .
```

Listing C.11: jConnect Private Packages

C.3.2 Maven Module

This section focuses on the Maven module set up that is needed to create an OSGi enabled version of a third party library after creating the BND definition.

This is the most complicated part, because you only need two files. This is the module layout:

```
module
├── pom.xml
└── osgi.bnd
```

Figure C.2: Maven module for OSGi third party libraries

This is the full pom.xml for the jConnect library:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
       /2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM
       /4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
3
4      <modelVersion>4.0.0</modelVersion>
5
6      <groupId>com.sybase365.arf.thirdparty</groupId>
7      <artifactId>jconnect-osgi</artifactId>
8      <version>7.0.5-SNAPSHOT</version>
9      <packaging>bundle</packaging>
10
11     <name>ARF :: 3rd Party Bundle :: Sybase jConnect for JDBC</name>
12
13     <properties>
14
15         <bundle.symbolicName>${pom.artifactId}</bundle.symbolicName>
16
17         <jconnect.version>7.0.5</jconnect.version>
18
19     </properties>
20
21     <build>
22         <plugins>
23
24             <!-- Create an OSGi Bundle Manifest -->
25             <plugin>
26                 <groupId>org.apache.felix</groupId>
27                 <artifactId>maven-bundle-plugin</artifactId>
28                 <extensions>true</extensions>
```

```
                <version>2.0.0</version>
                <configuration>
                    <obrRepository>${project.build.directory}/repository.xml</
                        obrRepository>
                    <instructions>
                        <manifestLocation>META-INF</manifestLocation>
                        <Bundle-SymbolicName>${bundle.symbolicName}</Bundle-
                            SymbolicName>
                        <Bundle-Version>${project.version}</Bundle-Version>
                        <Embed-Dependency>*;scope=provided;type=!pom;inline=true<
                            /Embed-Dependency>
                        <_include>-osgi.bnd</_include>
                    </instructions>
                </configuration>
            </plugin>
            <!-- Target JDK6 -->
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.0.2</version>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <!-- Not externally available; uploaded jar manually to our repository --
            >
        <dependency>
            <groupId>com.sybase.jdbcx</groupId>
            <artifactId>jconnect</artifactId>
            <version>${jconnect.version}</version>
            <scope>provided</scope>
        </dependency>

        <!-- Not externally available; uploaded jar manually to our repository --
            >
        <dependency>
            <groupId>com.sybase.jdbcx</groupId>
            <artifactId>jtds</artifactId>
            <version>${jconnect.version}</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

Listing C.12: jConnect pom.xml

And last, but not least the full osgi.bnd instruction file for BND tool:

```
1
2 -failok: true
3
4 Export-Package: com.sybase.jdbcx;uses:="javax.naming,javax.sql", \
5     com.sybase.jdbc4.jdbc;uses:="com.sybase.jdbcx,javax.naming,com.sybase.jdbc4.
          utils,org.ietf.jgss,javax.naming.spi,javax.naming.directory,javax. sql,
          com.sybase.jdbc4.tds,javax.transaction.xa",com.sybase.jdbc4.utils;uses:="
          com.sybase.jdbcx,com.sybase.jdbc4.jdbc,com.sybase.jdbc4.timedio", \
6     com.sybase.jdbc4.utils.resource, \
7     com.sybase.jdbc4.a.b;uses:="com.sybase.jdbc4.a.a,com.sybase.jdbc4.security.
          asn1", \
8     com.sybase.jdbc4.a.a, \
```

```
 9      com.sybase.jdbc4.a.b.a;uses:="com.sybase.jdbc4.a.b", \
10      com.sybase.jdbc4.tds;uses:="com.sybase.jdbc4.timedio,com.sybase.jdbc4.jdbc,
            com.sybase.jdbc4.utils,sun.io,javax.net.ssl,com.sybase.jdbcx,com.sybase.
            jdbc4.a.b,com.sybase.jdbc4.a.a,com.sybase.jdbc4.a.b.a,javax.security.auth
            ,org.ietf.jgss,javax.crypto,foo.bar,javax.net", \
11      com.sybase.jdbc4.jdbc.resource, \
12      com.sybase.jdbc4.timedio;uses:="com.sybase.jdbcx,com.sybase.jdbc4.tds,com.
            sybase.jdbc4.jdbc,com.sybase.jdbc4.utils", \
13
14  Private-Package: !*, .
15
16  Import-Package: com.sybase.jdbcx, \
17      com.sybase.jdbc4.jdbc, \
18      com.sybase.jdbc4.jdbc.resource, \
19      com.sybase.jdbc4.a.b, \
20      com.sybase.jdbc4.a.a, \
21      com.sybase.jdbc4.a.b.a, \
22      com.sybase.jdbc4.tds, \
23      com.sybase.jdbc4.timedio, \
24      com.sybase.jdbc4.security.asn1;resolution:=optional, \
25      javax.naming, \
26      javax.naming.directory, \
27      javax.naming.spi, \
28      javax.crypto, \
29      javax.security.auth, \
30      javax.transaction.xa, \
31      javax.sql, \
32      javax.net, \
33      javax.net.ssl;resolution:=optional, \
34      org.ietf.jgss;resolution:=optional, \
35      sun.io;resolution:=optional, \
36      foo.bar;resolution:=optional, \
37      xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx;
            resolution:=optional
```

Listing C.13: jConnect osgi.bnd

Appendix D

Mobiliser Error Codes

This chapter provides a comprehensive list of all currently available error codes for Mobiliser. The 4-digit error code space is reserved for Mobiliser standard error codes. Customizations must use 5-digit error codes.

Error Code	Description	Info Level
0	indicates no error occurred	INFO
51	The request's repeat flag is missing.	INFO
52	The request is still being processed.	INFO
53	The user sending a repeat request is invalid.	INFO
54	No message found for a repeated request.	INFO
55	No response was cached for a request.	INFO
101	The error code indicating that a customer is set up invalid or incomplete.	INFO
191	The error code indicating that some license issues prevent execution.	INFO
199	The error code indicating that some assumptions about the database are wrong, the database is inconsistent or mis configured.	INFO
201	No entity found for the given identifier.	INFO
202	There already exists an entity with the same identifier	INFO
203	Invalid data, an identifier or a whole entity is missing.	INFO
204	Invalid data was passed to the unit of work.	INFO
205	The calling user has not teh required access privilege.	INFO
206	An entity could not be deleted, because it is still referenced..	INFO
250	This error code depicts that any process is being suspended and waiting for further approval.	INFO
301	The customer is inactive.	INFO
302	The customer is blacklisted.	INFO
303	The requested customer was not found.	INFO
321	The customer's credential is expired.	INFO
322	The customer's credential is blocked.	INFO
323	The credential has an invalid pattern.	INFO
324	The credential contains a weak block.	INFO
325	The credential is invalid.	INFO
326	The credential was not found.	INFO
327	The credential is too long.	INFO
328	The credential is too short.	INFO
329	The credential wrong.	INFO
330	The credential is wrong and the customer was blacklisted.	INFO
331	The credential is ok, but expired.	INFO
332	The credential is ok, but temporary.	INFO
333	Configuration issue, no security policy found.	INFO
334	The credential can not be set, because it is still being retained.	INFO
351	The many active sessions.	INFO
352	The session was already closed.	INFO

353	The session is expired.	INFO
354	Session not supported.	INFO
355	No active session found.	INFO
361	To many logins.	INFO
362	The login has been stolen.	INFO
363	The login expired.	INFO
364	Login not supported.	INFO
399	The customer is blacklisted.	INFO
401	The customer data is missing	INFO
402	The customer has no active MSISDN.	INFO
403	The customer has no active email.	INFO
1001	The error code indicating that a request is incomplete or invalid; usually this refers to artifacts that cannot be covered by XML schemas. It also indicates a invalid logic in the request fields, such as minimum amount greater than max amount	INFO
1002	The error code indicating that an invalid MSISDN is specified.	INFO
1003	The error code indicating that an unknown transaction is specified.	INFO
1004	The error code indicating that a transaction is specified that cannot be considered in the request context.	INFO
1005	The error code indicating that a transaction is specified that cannot be considered due to its status.	INFO
1006	The error code indicating that a customer is specified whom is inactive.	INFO
1007	The error code indicating that either a live customer tried to participate in a test transaction or vice versa.	INFO
1008	The error code indicating that a customer is specified whom is blacklisted.	INFO
1009	The error code indicating that a request with the same origin and trace number was executed within the last 24 hours.	INFO
1012	The error code indicating that an unknown customer is specified.	INFO
1050	The error code indicating that the transaction is locked by another process and cannot be handled right now.	INFO
1101	The error code indicating that authentication data is missing.	INFO
1102	The error code indicating that authentication data is invalid; e.g. protocol violations or unknown/invalid encryption is used.	INFO
1111	The error code indicating that the authentication failed because the user does not exist.	INFO
1112	The error code indicating that the authentication failed because the user is in a status prohibiting to execute anything.	INFO
1113	The error code indicating that the authentication failed because the credentials are incorrect.	INFO
1114	The error code indicating that the authentication failed because the user is blocked due to the number of times wrong credentials have been specified.	INFO
1115	The error code indicating that the authentication failed because the user is blacklisted.	INFO
1116	The error code indicating that the authentication failed because the credentials are incorrect and the customer is blocked as of now.	INFO
1117	The error code indicating that the authentication failed because the credentials are incorrect and the customer is blacklisted as of now.	INFO
1121	The error code indicating that the credentials have expired and need to be updated.	INFO
1201	The error code indicating that the authorization failed because no agent/user data is given.	INFO
1202	The error code indicating that the authorization failed because the agent/user is in a status prohibiting to execute anything.	INFO
1203	The error code indicating that the identified customer/agent is not allowed to create a session or login. One reason could be that it is an internal customer	INFO
1211	The error code indicating that the authorization failed because the agent misses at least one privilege.	INFO
1221	The error code indicating that a basically authorized access to an unauthorized resource was detected.	INFO

1231	The transactions use case requires auto capture but the transaction is not	INFO
1232	The transactions use case permits auto capture but the transaction is an auto-capture one	INFO
1250	The transaction was pending in initial state too long and invalidated automatically.	INFO
1301	The error code indicating that a login failed because there is still an active session.	INFO
1302	The error code indicating that the customer has reached his limit of parallel sessions	INFO
1391	The error code indicating that the used application needs to be updated.	INFO
2401	The error code indicating that the connection to the gw–msg failed.	INFO
2402	The error code indicating that the connection to the gw–msg timed out.	INFO
2403	The error code indicating that reading data from the gw–msg timed out.	INFO
2404	The error code indicating that the gw–msg send data that is unexpected or not fitting the protocol.	INFO
2405	The error code indicating that sending a message via gw–msg failed: communication with gw–msg was ok, but gw–msg indicated an error.	INFO
2601	The error code indicating that the payer payment instrument is inactive.	INFO
2602	The error code indicating that the communication with the payment instrument's processor failed.	INFO
2603	The error code indicating that there is an inconsistency between the payer payment instrument's processor and Money Mobiliser.	INFO
2604	The payer payment instrument is in a wrong status for the current request.	INFO
2605	The payer payment instrument is not allowed for this use case.	INFO
2606	The error code indicating that the payer payment instrument's weekly limit is hit.	INFO
2607	The error code indicating that the payer payment instrument's daily limit is hit.	INFO
2608	The error code indicating that the payer payment instrument's monthly limit is hit.	INFO
2609	The error code indicating that the payer payment instrument's absolute limit is hit.	INFO
2610	The error code indicating that the payer payment instrument's single txn limit is hit.	INFO
2611	The error code indicating that the payer payment instrument has no funds available.	INFO
2612	The error code indicating that the payer payment instrument's weekly limit is hit.	INFO
2613	The error code indicating that the payer payment instrument's daily limit is hit.	INFO
2614	The error code indicating that the payer payment instrument's monthly limit is hit.	INFO
2615	The error code indicating that the payer payment instrument's absolute limit is hit.	INFO
2616	The error code indicating that the payer payment instrument's single minimum limit is hit.	INFO
2621	The error code indicating that the authorization of the payment instrument's processor has expired.	INFO
2622	The error code indicating that a capture cancel with this payment instrument's processor can only cancel the full capture amount.	INFO
2623	The error code indicating that the capture of the payer payment instrument's processor has "'frozen"' and can no longer be canceled.	INFO
2631	The error code indicating that the payer payment instrument is blocked on the issuer side (blacklisted).	INFO
2641	The error code indicating that the CVD validation of the payment instrument failed.	INFO
2650	The error code indicating that the transaction has no pickup code attached.	INFO
2651	The error code indicating that the maximum number of tries has been exceeded.	INFO
2652	The error code indicating that the pickup code is invalid.	INFO

2653	The error code indicating that the pickup code attribute attached to the transaction is not unique.	INFO
2654	The error code indicating that no internal voucher associated with the transaction can be found.	INFO
2655	The error code indicating that no internal voucher transaction associated with the transactionId can be found.	INFO
2656	The error code indicating that no internal voucher transaction associated with the transactionId can be found.	INFO
2660	Generic error code indicating that the wallet entry is invalid	INFO
2661	The error code indicating that a wallet entry exists for the same customer with a different payer payment instrument	INFO
2662	The error code indicating that a wallet entry exists for the same customer having the same debit or credit priority	INFO
2691	The error code indicating that the payer payment instrument does not support the Balance Inquiry operation.	INFO
2699	The error code indicating that the payer payment instrument processor returned an unknown error.	INFO
2701	The error code indicating that the payee payment instrument is inactive.	INFO
2702	The error code indicating that the communication with the payment instrument's processor failed.	INFO
2703	The error code indicating that there is an inconsistency between the payee payment instrument's processor and Money Mobiliser.	INFO
2704	The payee payment instrument is in a wrong status for the current request.	INFO
2705	The payee payment instrument is not allowed for this use case.	INFO
2706	The error code indicating that the payee payment instrument's weekly limit is hit.	INFO
2707	The error code indicating that the payee payment instrument's daily limit is hit.	INFO
2708	The error code indicating that the payee payment instrument's monthly limit is hit.	INFO
2709	The error code indicating that the payee payment instrument's absolute limit is hit.	INFO
2710	The error code indicating that the payee payment instrument's single txn limit is hit.	INFO
2711	The error code indicating that the payee payment instrument has no funds available.	INFO
2712	The error code indicating that the payee payment instrument's weekly limit is hit.	INFO
2713	The error code indicating that the payee payment instrument's daily limit is hit.	INFO
2714	The error code indicating that the payee payment instrument's monthly limit is hit.	INFO
2715	The error code indicating that the payee payment instrument's absolute limit is hit.	INFO
2716	The error code indicating that the payee payment instrument's single minimum limit is hit.	INFO
2721	The error code indicating that the authorization of the payment instrument's processor has expired.	INFO
2722	The error code indicating that a capture cancel with this payment instrument's processor can only cancel the full capture amount.	INFO
2723	The error code indicating that the capture of the payee payment instrument's processor has "'frozen"' and can no longer be canceled.	INFO
2731	The error code indicating that the payee payment instrument is blocked on the issuer side (blacklisted).	INFO
2741	The error code indicating that the CVD validation of the payment instrument failed.	INFO
2760	Generic error code indicating that the payee wallet entry is invalid	INFO
2761	The error code indicating that a payee wallet entry exists for the same customer with a different payee payment instrument	INFO
2762	The error code indicating that a payee wallet entry exists for the same customer having the same debit or credit priority	INFO

2791	The error code indicating that the payee payment instrument does not support the Balance Inquiry operation.	INFO
2799	The error code indicating that the payee payment instrument processor returned an unknown error.	INFO
2501	The error code indicating that the payee is unknown.	INFO
2502	The error code indicating that the payee is invalid / missing privileges	INFO
2503	The error code indicating that the payee has no PI.	INFO
2504	The error code indicating that the payee's PI does not match to the transaction's currency.	INFO
2505	The error code indicating that the payee PI does not support Credit .	INFO
2507	The error code indicating that the payee PI is unknown.	INFO
2508	The error code indicating that the caller cannot act as a delegate for the payee.	INFO
2511	The error code indicating that the payer is unknown.	INFO
2512	The error code indicating that the payer is invalid/missing privilege.	INFO
2513	The error code indicating that the payer has no PI.	INFO
2514	The error code indicating that the payer's PI does not match to the transaction's currency.	INFO
2515	The error code indicating that the payer PI does not support Debit .	INFO
2516	The error code indicating that the payer PI is not supported.	INFO
2517	The error code indicating that the payer PI is unknown.	INFO
2518	The error code indicating that the caller cannot act as a delegate for the payer.	INFO
2519	The error code indicating that an authorization expired.	INFO
2521	The error code indicating that an transaction authentication is required for the transaction.	INFO
2522	The error code indicating that a wrong OTP was specified.	INFO
2524	The error code indicating that a wrong OTP was specified and the transaction failed permanently.	INFO
2525	Indicates that the transaction processing has been stopped, because a manual decision is required	INFO
2531	The error code indicating that the capture amount exceeds the reservation amount.	INFO
2532	The error code indicating that the capture cancel amount exceeds the capture amount.	INFO
2533	The error code indicating that the transaction is not allowed because payee and payer are the same customer.	INFO
2534	The error code indicating that the sub transaction amount differs from the original transaction amount.	INFO
2535	The error code indicating that the payee has another open transaction and thus cannot open a new one.	INFO
2536	The error code indicating that the payer has another open transaction and thus cannot open a new one.	INFO
2537	The error code indicating that no open transaction was found.	INFO
2538	The error code indicating that the customer has multiple open transactions	INFO
2539	The error code indicating that the backstop authorization failed	INFO
2540	The error code indicating that the payee's wallet absolute limit count is hit.	INFO
2541	The error code indicating that the payer denied the transaction during the authorization process.	INFO
2542	The error code indicating that the system wasn't able to communicate with the payer during the authorization process.	INFO
2543	The error code indicating that the payee's wallet weekly limit count is hit.	INFO
2544	The error code indicating that the payer's wallet weekly limit count is hit.	INFO
2545	The error code indicating that the payee's wallet monthly limit count is hit.	INFO
2546	The error code indicating that the payer's wallet monthly limit count is hit.	INFO
2547	The error code indicating that the payer's wallet absolute limit count is hit.	INFO
2548	The error code indicating that the payee's wallet daily limit count is hit.	INFO
2549	The error code indicating that the payer's wallet daily limit count is hit.	INFO
2550	The error code indicating that the subscription is unknown.	INFO
2551	The error code indicating that the payee's wallet single transaction minimum underrun is hit.	INFO

2552	The error code indicating that the payer's wallet single transaction minimum underrun is hit.	INFO
2553	The error code indicating that the payee's weekly limit count is hit.	INFO
2554	The error code indicating that the payer's weekly limit count is hit.	INFO
2555	The error code indicating that there is no activation transaction for the PI tried to activate.	INFO
2556	The error code indicating that the activation amounts were wrong (at least one of them).	INFO
2557	The error code indicating that the activation amounts were wrong (at least one of them) and the activation transaction is invalidated.	INFO
2559	The error code indicating that the payee's absolute limit count is hit.	INFO
2560	The error code indicating that the payer's absolute limit count is hit.	INFO
2561	The error code indicating that the payee's absolute limit is hit.	INFO
2562	The error code indicating that the payer's absolute limit is hit.	INFO
2563	The error code indicating that the payee's monthly limit is hit.	INFO
2564	The error code indicating that the payer's monthly limit is hit.	INFO
2565	The error code indicating that the payee's daily limit is hit.	INFO
2566	The error code indicating that the payer's daily limit is hit.	INFO
2567	The error code indicating that the payee's weekly limit is hit.	INFO
2568	The error code indicating that the payer's weekly limit is hit.	INFO
2569	The error code indicating that the payee's single transaction limit is hit.	INFO
2570	The error code indicating that the payer's single transaction limit is hit.	INFO
2571	The error code indicating that the payee's wallet absolute limit is hit.	INFO
2572	The error code indicating that the payer's wallet absolute limit is hit.	INFO
2573	The error code indicating that the payee's wallet monthly limit is hit.	INFO
2574	The error code indicating that the payer's wallet monthly limit is hit.	INFO
2575	The error code indicating that the payee's wallet daily limit is hit.	INFO
2576	The error code indicating that the payer's wallet daily limit is hit.	INFO
2577	The error code indicating that the payee's wallet weekly limit is hit.	INFO
2578	The error code indicating that the payer's wallet weekly limit is hit.	INFO
2579	The error code indicating that the payee's wallet single limit is hit.	INFO
2580	The error code indicating that the payer's wallet single limit is hit.	INFO
2581	The error code indicating that there is a transaction restriction for the payee concerning the transaction amount.	INFO
2582	The error code indicating that there is a transaction restriction for the payer concerning the transaction amount.	INFO
2583	The error code indicating that there is a transaction restriction for the payee concerning the number of transactions.	INFO
2584	The error code indicating that there is a transaction restriction for the payer concerning the number of transactions.	INFO
2585	The error code indicating that there is a transaction restriction for the payee concerning the sum of transaction amounts.	INFO
2586	The error code indicating that there is a transaction restriction for the payer concerning the sum of transaction amounts.	INFO
2587	The error code indicating that there is a transaction restriction for the payee concerning the transaction amount.	INFO
2588	The error code indicating that there is a transaction restriction for the payer concerning the transaction amount.	INFO
2589	The error code indicating that there is an individual restriction for this txn type & customers.	INFO
2590	The error code indicating that the payer has insufficient KYC level	INFO
2591	The error code indicating that the payee has insufficient KYC level	INFO
2592	The error code indicating that the payee's single transaction minimum underrun is hit.	INFO
2593	The error code indicating that the payer's single transaction minimum underrun is hit.	INFO
2594	The error code indicating that the payee's monthly limit count is hit.	INFO
2595	The error code indicating that the payer's monthly limit count is hit.	INFO
2596	The error code indicating that the payee's daily limit count is hit.	INFO

2597	The error code indicating that the payer's daily limit count is hit.	INFO
2599	The error code indicating that transaction processing requires manual intervention.	INFO
2801	The error code indicating that connecting to the IVR failed.	INFO
2802	The error code indicating that an IVR service timeout occurred.	INFO
2803	The error code indicating that an IVR service timeout occurred.	INFO
2804	The error code indicating that the IVR user did not gave any input	INFO
2805	The error code indicating that a user did not pickup the phone.	INFO
2806	The error code indicating that a phone was busy.	INFO
2811	The error code indicating that a user ended an IVR call by hanging up.	INFO
2899	The error code indicating that an unexpected IVR error occurred.	INFO
2901	Indicates that a task in a workflow definition failed whie executing.	ERROR
2902	Indicates that a task in a workflow definition was rejected.	ERROR
5001	The specified Invoice Type reference is unknown to the Invoice issuer	DEBUG
5002	The specified Customer reference is unknown to the Invoice issuer	DEBUG
5003	The specified Customer reference is unknown to the Invoice issuer	DEBUG
5004	The specified Invoice has expired	DEBUG
5005	The specified Invoice has already been paid	DEBUG
5006	The specified Invoice has already been canceled	DEBUG
5007	The specified Invoice is in a wrong status for the operation	DEBUG
5008	The specified amount is lower than the Invoice amount	DEBUG
5009	The specified amount is higher than the Invoice amount	DEBUG
5010	The specified amount does not match with the Invoice amount	DEBUG
5099	The Invoice issuer reported an unknown error	DEBUG
5101	The Invoice Configuration can not be deleted because there are still active Invoices	DEBUG
5999	There was an unknown error with the Invoice processing	DEBUG
9935	Indicates that an unexpected database error occurred.	INFO
9999	Indicates that an unexpected error occurred.	INFO

Table D.1: Mobiliser Error Codes

Appendix E

Cron Expression Reference

Cron expressions provide the ability to specify complex time combinations such as “At 8:00am every Monday through Friday” or “At 1:30am every last Friday of the month”. Cron expressions are comprised of 6 required fields and one optional field separated by white space. The fields respectively are described as follows:

Field Name	Allowed Values	Allowed Special Characters
Seconds	0-59	,-*/
Minutes	0-59	,-*/
Hours	0-23	,-*/
Day-of-month	1-31	,-*?/LW
Month	1-12 or JAN-DEC	,-*/
Day-of-Week	1-7 or SUN-SAT	,-*?/L#
Year (Optional)	empty, 1970-2199	,-*/

Table E.1: Cron Pattern Fields

- The “*” character is used to specify all values. For example, “*” in the minute field means “every minute”.
- The “?” character is allowed for the day-of-month and day-of-week fields. It is used to specify “no specific week value”. This is useful when you need to specify something in one of the two fields, but not the other.
- The “-” character is used to specify ranges. For example “10-12” in the hour field means “the hours 10, 11 and 12”.
- The “,” character is used to specify additional values. For example “MON,WED,FRI” in the day of-week field means “the days Monday, Wednesday, and Friday”.
- The “/” character is used to specify increments. For example “0/15” in the seconds field means “the seconds 0, 15, 30, and 45”. And “5/15” in the seconds field means “the seconds 5, 20, 35, and 50”. Specifying “*” before the “/” is equivalent to specifying 0 is the value to start with. Essentially, for each field in the expression, there is a set of numbers that can be turned on or off. For seconds and minutes, the numbers range from 0 to 59. For hours 0 to 23, for days of the month 0 to 31, and for months 1 to 12. The “/” character simply helps you turn on every "nth" value in the given set. Thus “7/6” in the month field only turns on month “7”, it does NOT mean every 6th month, please note that subtlety.
- The “L” character is allowed for the day-of-month and day-of-week fields. This character is short-hand for “last”, but it has different meaning in each of the two fields. For example, the value “L” in the day-of-month field means “the last day of the month” - day 31 for January, day 28 for February on non-leap years. If used leap in the day-of-week field by itself, it simply means “7” or “SAT”. But if used in the day week field after day-of-week another value, it means “the last xxx day of the month” - for example “6L” means “the last friday of the month”. When using the “L” option, it is important not to specify lists, or ranges of values, as you’ll get confusing results.

- The “W” character is allowed for the day month field. This character is used to specify the weekday day-of-month (Monday-Friday) nearest the given day. As an example, if you were to specify “15W” as the value for the day-of-month field, the meaning is: “the nearest weekday to the 15th of the month”. So if the 15th is a Saturday, the trigger will fire on Friday the 14th. If the 15th is a Sunday, the trigger will fire on Monday the 16th. If the 15th is a Tuesday, then it will fire on Tuesday the 15th. However if you specify “1W” as the value, for day-of-month, and the 1st is a Saturday, the trigger will fire on Monday the 3rd, as it will not ’jump’ over the boundary of a month’s days. The “W” character can only be specified when the day-of-month is a single day, not a range or list of days.
- The “L” and “W” characters can also be combined for the day month expression to yield “LW”, which translates to “last weekday of the month”.
- The “#” character is allowed for the day week field. This character is used to specify “the nth" XXX day of the month”. For example, the value of “6#3” in the day week field means the third Friday of the month (day 6 = Friday and “#3” = the 3rd one in the month). Other examples: “2#1” = the first Monday of the month and “4#5” = the fifth Wednesday of the month. Note that if you specify “#5” and there is not 5 of the given day-of-week in the month, then no firing will occur that month. If the “#” character is used, there can only be one expression in the day-of-week field (“3#1,6#3” is not valid, since there are two expressions).
- The legal characters and the names of months and days of the week are not case sensitive.