Developer's Guide

Table of contents

1. Introduction

Dimensions is a multi-channel framework that works on top of Tiles, because the entry-point classes (ChannelFactorySet and ConfigurableFactorySet) implement the FactorySet class (and then the DefinitionsFactory class). This way, Tiles delegates all of the job of building Tiles definitions (in terms of object representations) to Dimensions.

The main purpose of Dimensions is to deliver different Tiles definitions starting from the same request, depending on situations like different client devices, different user roles, etc., that can be decided by the programmer. This is achieved by assigning different Tiles definitions configuration files to different "cases".

During execution, if a "case" is identified, a "Decision" is made by a "Decider" and then a particular Tiles definitions configuration file is used. If a case is identified but no corresponding "decision" can be taken, the best thing that a Decider can do is "degrade" to a feasible decision.

2. The architecture

In this section we will explain the architecture of Dimensions 0.6. If you want to investigate the "old" 0.5 architecture you should go to the old Dimensions website. (???)

Dimensions is composed of three main parts:

- The class `ConfigurableFactorySet`, an implementation of `FactorySet` abstract class. It is the interface between Tiles and the internal part of Dimensions.
- The configurator, that is implemented as a part of `ConfigurableFactorySet`, that is able to build all the needed objects.
- The `Decider` interface: it has the responsibility to inteprete the HTTP request and take a `Decision`. The class `ConfigurableFactorySet` will use this "decision" to deliver the correct definitions set, among which the requested Tiles definition should be picked.

2.1. ConfigurableFactorySet runtime part

The `ConfigurableFactorySet` class is the entry point for Dimensions framework. The Tiles engine will pass all of Tiles definitions names to this class, that (at the end) will answer with an object of class `XmlDefinition`.

This class extends the `FactorySet` class, that permits to store different `DefinitionsFactory`, though it acts as a `DefinitionsFactory` itself! `FactorySet` contains a key-value map, where values are `DefinitionsFactory` objects and the keys can be anything. In our case, the keys are objects of class `Decision`.

The most important methods are:

- `getDefinitionsFactoryKey`: it is responsible to return the key (i.e. the decision) after calling the decider;
- `createFactory`: it is responsible to create a `DefinitionsFactory` for the given key (i.e. decision).

The class `FactorySet` will not recreate an already built DefinitionsFactory, so it has an already made caching artifact!

2.2. ConfigurableFactorySet configurator

This class has also the task of loading the configuration object. This is made by parsing the configuration file using Jakarta Commons Digester and a custom ruleset.

(To be completed...)