OpenArm Agent Configuration

1. OpenArm Programmatic Configuration (Dependency Injection, IoC)

OpenArm is best configured programmatically, via an instance of the OpenArmConfiguration interface. This interface has only one method:

```
Map getMediatorConfigurations()
```

You write a class that implements this interface, and pass an instance of this class to the constructor of OpenArmTransactionSimpleFacade. Typically, we tend to write an anonymous inner class that implements the OpenArmConfiguration interface, and pass it, in place, to the constructor.

Implementations must return a Map containing Mediator configurations, where the fully qualified class name of the Mediator is the key of each entry in the Map. For example:

```
return new OpenArmConfiguration() {
    public Map getMediatorConfigurations() {
        final Map result = new HashMap();
        result.put("net.m2technologies.open_arm.transport.transaction.snmp.SnmpMediator
                new SnmpMediatorConfiguration("127.0.0.1",
                162,
                "1.3.6.1.2.1.2.0",
                "1.3.6.1.2.1.2.0.0.0.42",
                2000,
                10000,
                true));
        result.put("net.m2technologies.open_arm.transport.transaction.logging.LoggingMe
                new LoggingMediatorConfiguration());
        return result;
    }
};
```

This code creates a configuration using both the LoggingMediator and the SnmpMediator, and configures the SnmpMediator (the LoggingMediator's configuration has no attributes).

> **Note:**
> Note the use of a "configuration class" for each Mediator. All of the Mediators delivered with OpenArm conform to this pattern. Although you're free to solve the problem any way you like with Mediators that you write yourself, you may find this approach the simplest. The Mediator Configuration classes that accompany each OpenArm Mediator should provide enough of an example.

Note:

Mediators are additive. The example above sets up the agent to use **both** the logging and the SNMP Mediators -- such a setup, would, for example, both send traps and log events to log4J.

Note that this style of configuration is significantly more flexible -- you can not only specify which Mediators to use; you can also immediately configure them. With the older, file based configuration approach, you can only specify which Mediators to use -- if a given Mediator requires configuration itself, you must provide for this separately, typically via a further configuration file, specific to the Mediator. The SnmpTrapMediator can be configured this way, for example. The drawback to this technique is the same problem one is confronted with in the real estate business -- location. Where are the configuration files, at runtime? Where should they be kept? This turns out to be an especially royal pain in the ass when running within the context of a container of some sort, such as a Servlet container, or a J2EE application server. The mounting complexity of this approach has led to the (rebirth of the) idea of "dependency injection", also referred to as "inversion of control". The programmatic configuration, introduced in v. 0.009, allows us to ride on the coattails of the "dependency injection" hype sufficiently to sidestep the thorny issues involved in specifying a "home grown" configuration mechanism (as any OpenArm-specific, file based, configuration mechanism is and must be, logically).

The assumption here is that you, the application programmer, have some mechanism in place, somewhere in your application, that loads all relevant configuration data from some sort of repository, presumably at startup, and that you simply use this data to initialize OpenArm. Where and/or how you do this is irrelevant -- perhaps you use JNDI inside a J2EE application server, or perhaps you've rolled your own mechanism. It simply doesn't interest OpenArm. Give OpenArm an instance of OpenArmConfiguration, containing a Map of the various Mediator configurations you want to use, and OpenArm is happy.

This approach also seems to fit better with the "OpenArm is a library for use within your application" philosophy. Libraries don't have external configurations, in this way of thinking -- applications do. And applications use their knowledge to instantiate their libraries in a particular way.

We therefore **strongly** recommend using the programmatic, "dependency injection" technique for configuring OpenArm -- we will probably be deprecating (and eventually removing) the file-based mechanism in the near future.

2. OpenArm Agent Configuration File

The configuration file for the OpenArm Agent is very simple: each line of the file must contain the fully qualified class name (FQN) of a class implementing the TransportMediator

interface. Example:

```
net.m2technologies.open_arm.transport.transaction.snmp.SnmpMediator
net.m2technologies.open_arm.transport.transaction.logging.LoggingMediat
```

This would configure the OpenArmAgent to use the SnmpMediator and the LoggingMediator.

If you write your own Mediator, all you have to do to use it is include its FQN in your configuration file.