01 Getting Started

Registry_UI_registered

THIS PAGE IS NOT YET READY

Getting started with DistributeMe

This page will teach you how to write your very first service with DistributeMe. This involves some simple steps. You can do it yourself or checkout the distributeme helloworld example from our svn:

```
svn://svn.anotheria.net/opensource/distributeme-examples/trunk/helloworld
```


Coding the first service.

Our first service will be pretty straight forward. It does almost nothing, but prints something into console.

Interface

DistributeMe is all about interfaces, and therefore we first have to define the interface:

HelloWorldService.java

```
package org.distributeme.examples.helloworld;

import net.anotheria.anoprise.metafactory.Service;
import org.distributeme.annotation.DistributeMe;

@DistributeMe
public interface HelloWorldService extends Service{
        void printMessage(String message);
}
```

There are actually two things that differs from any other normal interface of a service:

1. *@DistributeMe* annotation above the interface declaration which tell the preprocessor to generate java code for this service and which can contain additional generation parameters we will care about later.

2. **extends Service{** - which is needed by DistributeMe internal DI management. It's an empty interface, which signalizes that this component is a service and not of further interest now.

Implementation

Of course we have to implement the interface to make it work. The implementation is absolutely straight-forward:

HelloWorldServiceImpl.java

```
package org.distributeme.examples.helloworld;

public class HelloWorldServiceImpl implements HelloWorldService{
  @Override
  public void printMessage(String message) {
    System.out.println(message);
  }
}
```

As you can see there is absolutely nothing special about this implementation.

The client

Finally we need something that calls the service.

RemoteClient.java

```
package org.distributeme.examples.helloworld;
import java.util.Date;
import org.distributeme.core.ServiceLocator;


public class RemoteClient {
  public static void main(String a[]){
    HelloWorldService service = ServiceLocator.getRemote(HelloWorldService.class);
    String message = "Hello world at "+new Date(System.currentTimeMillis());
    System.out.println("Server should print out following message now: "+message);
    service.printMessage(message);
  }
}
```

The client is called remote client because it calls explicitly the remote instance. Usually you would only know the interface and let some DI mechanism decide which instance to return (for example: MetaFactory). However, for the test its easier to call the remote service explicitly.

So the only unusual thing in this client is the following call:

```
HelloWorldService service = ServiceLocator.getRemote(HelloWorldService.class);
```

This is an utility provided by distributeme, which knows the name of the class, you are looking for (but you can call new RemoteHelloWorldServiceStub() directly instead).

Now you have all you need to build and run the example.

Building

You can build the project with whatever tool you want, we supply a pom.xml for building with maven 2 or 3. However any tool that calls apt (future javax.annotations.Processor) will do (ant and nearly everything else).

pom.xml

```
<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">
  <parent>
```

```
      <groupId>net.anotheria</groupId>
      <artifactId>parent</artifactId>
      <version>1.8</version>
</parent>
<modelVersion>4.0.0</modelVersion>
<groupId>net.anotheria</groupId>
<artifactId>helloworld</artifactId>
<version>1.0.0-SNAPSHOT</version>
<name>helloworld example</name>

<build>
  <plugins>
        <plugin>
        <groupId>org.codehaus.mojo</groupId>
          <artifactId>apt-maven-plugin</artifactId>
          <dependencies>
                  <dependency>
                          <groupId>org.jfrog.maven.annomojo</groupId>
                          <artifactId>maven-plugin-tools-anno</artifactId>
                          <version>1.3.1</version>
                          <exclusions>
                                  <exclusion>
                                          <groupId>com.sun</groupId>
                                          <artifactId>tools</artifactId>
                                  </exclusion>
                          </exclusions>
                  </dependency>
              <dependency>
                    <groupId>org.codehaus.mojo</groupId>
              <artifactId>cobertura-maven-plugin</artifactId>
              <version>2.4</version>
              </dependency>
          </dependencies>
    <executions>
      <execution>
          <id>process</id>
              <goals>
                 <goal>process</goal>
              </goals>
              <phase>generate-sources</phase>
              <configuration>
                      <factory>org.distributeme.processors.GeneratorProcessorFactory</factory>
                      <encoding>UTF-8</encoding>
                              <verbose>true</verbose>
                              <outputDirectory>${project.basedir}/generated/java</outputDirectory>
                    </configuration>
      </execution>
    </executions>
      </plugin>
      <plugin>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>2.2</version>
    <configuration>
      <descriptorRefs>
        <descriptorRef>jar-with-dependencies</descriptorRef>
      </descriptorRefs>
    </configuration>
    <executions>
      <execution>
        <id>make-assembly</id> <!-- this is used for inheritance merges -->
        <phase>package</phase> <!-- append to the packaging phase. -->
        <goals>
          <goal>single</goal> <!-- goals == mojos -->
        </goals>
      </execution>
    </executions>
  </plugin>
 </plugins>
</build>

  <reporting>
```

```
          </reporting>
  <dependencies>
        <dependency>
                <groupId>net.anotheria</groupId>
                <artifactId>distributeme-support</artifactId>
                <version>1.2.1</version>
        </dependency>
        <dependency>
                <groupId>net.anotheria</groupId>
                <artifactId>distributeme-generator</artifactId>
                <version>1.2.1</version>
        </dependency>
        <dependency>
                <groupId>net.anotheria</groupId>
                <artifactId>distributeme-agents</artifactId>
                <version>1.2.1</version>
        </dependency>
        <dependency>
                <groupId>net.anotheria</groupId>
                <artifactId>ano-prise</artifactId>
                <version>1.0.4</version>
        </dependency>
  </dependencies>
                <repositories>
                        <repository>
                                <id>anotheria</id>
                                <url>http://nexus.anotheria.net/nexus/content/groups/public</url>
                                <releases>
                                        <enabled>true</enabled>
                                </releases>
                                <snapshots>
                                        <enabled>true</enabled>
                                </snapshots>
                        </repository>
                </repositories>
</project>
```

As usually with maven the pom is longer than everything else in the project 😅

We added the repositories directly in the pom, you'll probably not doing that, but edit your settings instead. For details on that talk to your local maven handler.

We refer to our parent (net.anotheria.parent) which defines stuff like testing or compiling already, to save some place in this pom.

If you use this pom, you'll probably want to check if the versions are still uptodate, check Change Log for details on current versions.

Now run maven with:

```
mvn clean install
```

This should produce something like this:

```
...

[INFO] --- maven-install-plugin:2.3.1:install (default-install) @ helloworld ---
[INFO] Installing /Users/yourname/projects/distributeme-examples/helloworld/target/helloworld-1.0.0-SNAPSHOT.
jar to /Users/yourname/.m2/repository/net/anotheria/helloworld/1.0.0-SNAPSHOT/helloworld-1.0.0-SNAPSHOT.jar
[INFO] Installing /Users/your name/projects/distributeme-examples/helloworld/pom.xml to /Users/yourname/.m2
/repository/net/anotheria/helloworld/1.0.0-SNAPSHOT/helloworld-1.0.0-SNAPSHOT.pom
[INFO] Installing /Users/yourname/projects/distributeme-examples/helloworld/target/helloworld-1.0.0-SNAPSHOT-
jar-with-dependencies.jar to /Users/yourname/.m2/repository/net/anotheria/helloworld/1.0.0-SNAPSHOT/helloworld-
1.0.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 12.071s
[INFO] Finished at: Mon Aug 13 23:35:34 CEST 2012
[INFO] Final Memory: 14M/265M
[INFO] ------------------------------------------------------------------------
```

Background

For running this example you also need to launch DistributeMe Registry. This process is described on Building & running page.

Leaving Tomcat with DistributeMe Registry application running you can complete an example with follow-up actions.

Running an example

Go back to our example *helloworld* project directory.

Along with project sources and *target* there are three shell scripts: *start.sh, startServer.sh, startRemoteClient.sh*. Open one new terminal (do not stop Tomcat) for running the server, and another one new for running the client.

If you have opened new terminal in helloworld directory, then it's your current working directory (CWD). To avoid some errors while running an example classes, create directory named */logs* in CWD before the next actions.

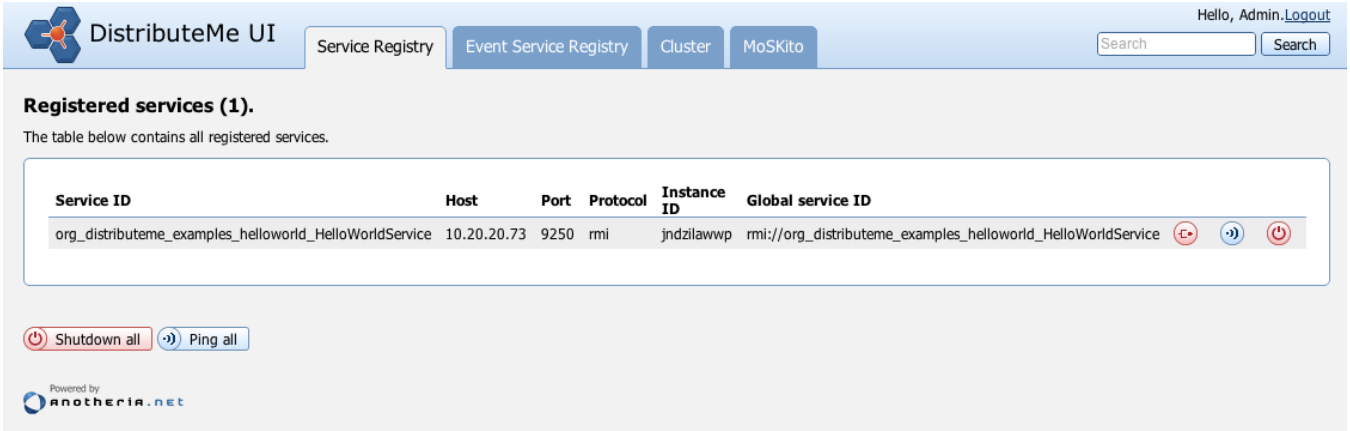
Starting server

Execute *startServer.sh script:*

```
bezuhlyi:helloworld bezuhlyi$ ./startServer.sh
CLASSPATH: etc/appdata:target/helloworld-1.0.0-SNAPSHOT-jar-with-dependencies.jar
Server org_distributeme_examples_helloworld_HelloWorldService is up and ready.
```

Process wil be kept running.

We can see the result, that service has been successfully registered in DistributeMe Registry:



So service is available for remote invocations of its methods now.

Starting client

And in another terminal execute *startRemoteClient.sh script:*

```
bezuhlyi:helloworld bezuhlyi$ ./startRemoteClient.sh
CLASSPATH: etc/appdata:target/helloworld-1.0.0-SNAPSHOT-jar-with-dependencies.jar
Server should print out following message now: Hello world at Fri Oct 26 20:06:59 EEST 2012
bezuhlyi:helloworld bezuhlyi$
```

The result

After the execution look at the server's process terminal:

```
bezuhlyi:helloworld bezuhlyi$ ./startServer.sh
CLASSPATH: etc/appdata:target/helloworld-1.0.0-SNAPSHOT-jar-with-dependencies.jar
Server org_distributeme_examples_helloworld_HelloWorldService is up and ready.
Hello world at Fri Oct 26 20:06:59 EEST 2012
```

Method thad had been called by client was executed on server.