Terracotta Integration Modules Manual

About Terracotta Documentation

This documentation is about Terracotta DSO, an advanced distributed-computing technology aimed at meeting special clustering requirements.

Terracotta products without the overhead and complexity of DSO meet the needs of almost all use cases and clustering requirements. To learn how to migrate from Terracotta DSO to standard Terracotta products, see Migrating From Terracotta DSO. To find documentation on non-DSO (standard) Terracotta products, see Terracotta Documentation. Terracotta release information, such as release notes and platform compatibility, is found in Product Information.

Terracotta Integration Modules Manual

Before you start

- Questions? Find answers, insights, and inspiration at the Terracotta forums.
- Promote your integration project to an official Terracotta project at the Terracotta Forge and learn about becoming a contributor.
- Share your integration project with a community of Terracotta DSO users.

Introduction

Terracotta Integration Modules (TIMs) are sets of configuration elements and supporting Java classes packaged together as a single, includable module within the Terracotta configuration. A TIM allows you to integrate Terracotta DSO with the framework or platform that your application is based on.

While a Terracotta configuration file resides with a Terracotta server instance, TIMs are never installed on Terracotta server instances. This is because applications are never integrated with Terracotta server instances, only with Terracotta clients. Terracotta clients get their TIM configurations when they fetch the Terracotta configuration file from a Terracotta server instance or by having their own Terracotta configuration files.

The catalog of available TIMs continues to grow as new technologies are integrated with Terracotta DSO. Community-developed TIMs are also being added. For the complete list of currently supported modules, see the Terracotta Forge at http://forge.terracotta.org.

If a TIM isn't available for your environment, this document shows you how to build one.

While there are many approaches and tools that could be used to design and build a TIM, we recommend Apache Maven. Except as noted below, this document focuses on using Maven2 for managing a TIM project.

Integration Module Versioning

Versioning a TIM to fit with the Terracotta development scheme is crucial for a smooth integration path. The Terracotta development scheme takes the Apache Maven approach of appending "SNAPSHOT" to build versions still in flux. A typical TIM filename looks like the following:

```
clustered-hibernate-3.2.5-2.6.0.jar
```

where a descriptive name (clustered-hibernate) is followed by the named application's version (3.2.5), and finally the TIM version (2.6.0).

To facilitate a consistent and functional version-control process, an official management policy for TIM publishing is in place. If you pursue creating and publishing a TIM, follow these guidelines:

- There is a single Maven2 repository that hosts both releases and snapshots – using it is strongly recommended.
- Whenever a change is made to a TIM, the version number must be incremented as appropriate, in *both* the `MANIFEST.MF` file for the module and the module's own `pom.xml`.
- A TIM must remain a `-SNAPSHOT` version until it becomes an official release.
- If a TIM depends on `-SNAPSHOT` versions of other integration modules, then the dependencies for that TIM must explicitly declare those versions.
- A released TIM cannot depend on `-SNAPSHOT` versions of other integration modules.
- At product release time, TIMs are blessed as release versions with an incremented version number.
- Release versions of all current TIMs must be published to the Terracotta Maven repository.

Create Your Own Terracotta Integration Module

TIMs are very similar to OSGi bundles or Eclipse plugins. Logically, a TIM is a specification loaded at runtime that tells the Terracotta software how to instrument certain Java objects. Structurally, a TIM is a `jar` file with special headers in the manifest. While a TIM can contain code that modifies bytecode directly, normally it's nothing more than an XML file containing a subtree of the `tc-config.xml` file.

To meet Terracotta integration specification, the TIM requires at least one of the following:

a) A `terracotta.xml` file (located in the root of the `jar`)
b) An OSGi `BundleActivator`

Case b) supports a special use case where custom bytecode manipulation is required. Case b) is not likely to be encountered and is not covered here.

Java code vs. terracotta.xml

If your project requires using a `BundleActivator` and you need help, post a question on the developer mailing list.

You can build your TIM using Apache Maven and the Maven TIM archetype, or manually. The procedures for each of these methods are given below.

For either procedure, be sure you are familiar with Terracotta DSO and have a working `tc-config.xml` file.

Procedure Using the Maven TIM Archetype

This procedure assumes that you have Maven installed and are familiar with Maven projects.

1. Download and install the Maven TIM archetype.
2. Generate a Maven TIM project archetype by following **these** instructions.
   Generating the project creates the necessary directory structure, a stub configuration file, a manifest file, and a Maven `pom.xml`.
3. that you can use to build the integration module jar.
4. Edit `terracotta.xml` to support the library or framework your TIM is intended for.
   See terracotta.xml for details.
5. Add the appropriate headers to the `MANIFEST.MF` file.
   See MANIFEST.MF for details.

Manual Procedure

1. Create a `terracotta.xml`.
2. Edit `terracotta.xml` to support the library or framework your TIM is intended for.
   See terracotta.xml for details.
3. Create a `jar` manifest file called `MANIFEST.MF`.
4. Add the appropriate headers to the `MANIFEST.MF` file.
   See MANIFEST.MF for details.
5. Package the `terrracotta.xml` and `MANIFEST.MF` files into a `jar` file.
   For example:

   ```
   jar cvfm MyApp-cluster-config-1.0-1.0.0-SNAPSHOT.jar META-INF/MANIFEST.MF terracotta.xml
   ```

   Be sure to name your `jar` file using conventions that facilitate your integration project. See the versioning guidelines above and these usage guidelines for more information.

terracotta.xml

The `terracotta.xml` file contains a subtree of the `application/dso` element of the `<tc-config.xml>` file, wrapped in an `<xml-fragment>` element. If the content of your `tc-config.xml` file is similar to the following:

```
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">
  <!-- ...stuff... -->
  <application>
    <dso>
      <instrumented-classes>
        <!-- ...other stuff... -->
      </instrumented-classes>
      <locks>
        <!-- ...yet more stuff... -->
      </locks>
    </dso>
  </application>
  <!-- ...trailing stuff... -->
</tc:tc-config>
```

then the content of your `terracotta.xml` file should be similar to the following:

```
<xml-fragment>
  <instrumented-classes>
    <!-- ...other stuff... -->
  </instrumented-classes>
  <locks>
    <!-- ...yet more stuff... -->
  </locks>
</xml-fragment>
```

META-INF/MANIFEST.MF

The manifest file has one required OSGi header: `Bundle-SymbolicName`. Optional headers can be added to make the manifest file more useful.

> **Generate TIM Manifest using Terracotta Maven Plugin**
>
> If Maven is used to build the TIM project, the TIM manifest file can be generated using the Maven Plugin for Terracotta.

The following is an example of set of headers in a `MANIFEST.MF` file:

```
Bundle-Description: MyApp/MyFramework Cluster Configuration
Bundle-DocURL: http://www.myorg.org/doc/terracotta-integration-module/
Bundle-Name: Terracotta integration module for MyApp/MyFramework
Bundle-SymbolicName: org.myorg.myapp.integration_module
Bundle-Vendor: MyOrg, Inc.
Bundle-Version: 1.0
```

See the OSGi R4 documentation for more information on headers. Eclipse offers a manifest editor that makes it easier to work with manifest files.

Using Your TIM

To use your TIM, save the TIM `jar` file to a module repository and add the following element to clients/modules in your `tc-config.xml` file:

```
<module name="MyApp-cluster-config" version="1.0"/>
```

Adding this element effectively embeds the `terracotta.xml` in `tc-config.xml`.

> **Module Versions Are Optional**
>
> Since the tim-get script finds the optimal version for the current installation of the Terracotta kit, module versions are optional.

See these usage guidelines for a more detailed procedure.

Using a TIM to Replace a Shared Class

Under certain circumstances, you may want to replace a class with your own version. This can be done using a TIM. The TIM's contents should be similar to the following:

```
public class FooModule extends TerracottaConfiguratorModule {
     protected void addInstrumentation(BundleContext context) {
         super.addInstrumentation(context);
         Bundle bundle = getExportedBundle(context, "org.foo.tim-foo");
         addClassReplacement(bundle, "org.foo.Impl", "org.foo.ClusteredImpl");
     }
 }
```

where:

- FooModule is the class name of the TIM.
- org.foo.Impl is the class to replace.
- org.foo.ClusteredImpl is the new class.
- org.foo.tim-foo is the value of the TIM's *Bundle-SymbolicName* manifest header. For example, the ehcache-1.3 TIM has the *Bundle-SymbolicName* manifest header *org.terracotta.modules.clustered-ehcache-1.3*.

Additional Reading

See Juris Galang's blog entry on using the TIM archetype.