JBoss JCA 1.0 User's Guide

Connecting your Enterprise Information Systems

Table of Contents

1

About JBoss JCA

The goal of the JBoss JCA project is to provide an implementation of the Java Connector Architecture 1.6 specification.

The specification can be found here: http://www.jcp.org/en/jsr/detail?id=322.

1.1. The team

Jesper Pedersen acts as the lead for the JBoss JCA project. He can be reached at jesper (dot) pedersen (at) jboss (dot) org.

Jeff Zhang is a core developer on the JBoss JCA project. He can be reached at jizhang (at) redhat (dot) com.

Gurkan Erdogdu is a core developer on the JBoss JCA project. He can be reached at gurkanerdogdu (at) yahoo (dot) com.

Stefano Maestri is a core developer on the JBoss JCA project. He can be reached at stefano.maestri (at) javalinux (dot) it.

1.2. Thanks to

Dimitris Andreadis, Carlo de Wolf, Jason Green, Jonathan Halliday, Søren Hilmer, Vicky Kak, Aslak Knutsen, Sacha Labourey, Alexey Loubyansky, Patrick MacDonald, Andrig Miller, Andrew Lee Rubinger, Anil Saldhana and Scott Stark.

And a special thanks goes to Adrian Brock and Ales Justin for their continuous support of this project.

2

Introduction

The Java Connector Architecture (JCA) defines a standard architecture for connecting the Java EE platform to heterogeneous Enterprise Information Systems (EIS). Examples of EISs include Enterprise Resource Planning (ERP), mainframe transaction processing (TP), database and messaging systems.

The connector architecture defines a set of scalable, secure, and transactional mechanisms that enable the integration of EISs with application servers and enterprise applications.

The connector architecture also defines a Common Client Interface (CCI) for EIS access. The CCI defines a client API for interacting with heterogeneous EISs.

The connector architecture enables an EIS vendor to provide a standard resource adapter for its EIS. A resource adapter is a system-level software driver that is used by a Java application to connect to an EIS. The resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application. The resource adapter serves as a protocol adapter that allows any arbitrary EIS communication protocol to be used for connectivity. An application server vendor extends its system once to support the connector architecture and is then assured of seamless connectivity to multiple EISs. Likewise, an EIS vendor provides one standard resource adapter which has the capability to plug in to any application server that supports the connector architecture.

2.1. What's New

The Java Connector Architecture 1.6 specification adds the following major areas:

- Ease of Development: The use of annotations reduces or completely eliminates the need to deal with a deployment descriptor in many cases. The use of annotations also reduces the need to keep the deployment descriptor synchronized with changes to source code.
- Generic work context contract: A generic contract that enables a resource adapter to control the execution context of a Work instance that it has submitted to the application server for execution.
- Security work context: A standard contract that enables a resource adapter to establish security information while submiting a Work instance for execution to a WorkManager and while delivering messages to message endpoints residing in the application server.
- Standalone Container Environment: A defined set of services that makes up a standalone execution environment for resource adapters.

3

Download

The official JBoss JCA project page is http://www.jboss.org/jca/ where you can download the software.

3.1. Download

The download location is zone: http://www.jboss.org/jca/downloads/

Each release is labelled with a version number and an identifier.

```
jboss-jca-<major>.<minor>.<patch>[.<identifier>]
```

where

- Major: The major version number. Signifies major changes in the implementation.
- Minor: The minor version number. Signifies functional changes to a major version.
- Patch: The patch version number. Signifies a binary compatible change to a minor version.
- Identifier: The identifier. Identifies the level of the quality of the release.
    - None / GA: Stable release
    - CR: Candidate for Release quality. The implementation is functional complete.
    - Beta: Beta quality. The implementation is almost functional complete.
    - Alpha: Alpha quality. The implementation is a snapshot of the development.

An example

```
jboss-jca-1.0.0.tar.gz
```

which is the first stable release of the project.

3.2. SVN Access

If you want to experiment with the latest developments you may checkout the latest code from SVN trunk. Be aware that the information provided in this manual might then not be accurate.

The anonymous SVN repository is located under:

```
svn co http://anonsvn.jboss.org/repos/jbossas/projects/jboss-jca/trunk/ jbossjca-trunk
```

```

```

You can find additional information about this in the developer guide.

4

Installation

Once you have downloaded the distribution you need to install it in a location of your choice.

4.1. Compressed Tape Archive (.tar.gz)

Extract the distribution using

```
tar xzf jboss-jca-1.0.0.tar.gz
```

The distribution will be located in a directory named

```
jboss-jca-1.0.0
```

4.2. Zip Archive (.zip)

Extract the distribution using

```
unzip jboss-jca-1.0.0.tar.gz
```

or any program capable of handling Zip archives such as WinZip and WinRar.

The distribution will be located in a directory named

```
jboss-jca-1.0.0
```

4.3. Directory structure

The JBoss JCA container has the following directory structure:

- bin: Contains the scripts that starts the container.
- config: Contains the configuration of the container.
- deploy: Contains user deployments.

- doc: Contains the documentation.
- lib: Contains all the libraries needed by the container.
- log: Contains the log files for the container.
- system: Contains system deployments.
- tmp: Contains temporary files.

5

Configuration

The configuration for the JBoss JCA container is located in the config/ directory.

5.1. Logging service

The JBoss JCA container uses JBoss Logging framework as the implementation.

The configuration is done in the

```
config/logging.properties
```

file.

Consult the JBoss Logging documentation [http://www.jboss.org/community/wiki/JBossBootLogging] on how the service can be configured.

5.2. Transaction service

The JBoss JCA container uses the JBoss Transaction Manager as its transaction implementation.

The configuration is done in the

```
config/transaction.xml
```

file.

Consult the JBoss Transaction documentation on how the service can be configured.

5.3. JCA

5.3.1. Deployer

The JBoss JCA deployer is configured in the

```
config/bootstrap/jca.xml
```

```
```

file, as the

```
<bean name="RADeployer"
      interface="com.github.fungal.spi.deployers.Deployer"
      class="org.jboss.jca.deployers.fungal.RADeployer">
  <depends>BeanValidation</depends>
  <depends>WorkManager</depends>
</bean>
```

bean.

Table 5.1. JCA Deployer

Property	Type	Description
`ArchiveValidation`	`boolean`	Toggle archive validation for the deployment units. Default: `true`
`ArchiveValidationFailOnWarn`	`boolean`	Should an archive validation warning report fail the deployment. Default: `false`
`ArchiveValidationFailOnError`	`boolean`	Should an archive validation error report fail the deployment. Default: `true`
`BeanValidation`	`boolean`	Toggle bean validation (JSR-303) for the deployment units. Default: `true`
`DefaultBootstrapContext`	`org.jboss.jca.core.api.CloneableBootstrapContext`	Specifies the default bootstrap context for resource adapters
`BootstrapContexts`	`Map<String, org.jboss.jca.core.api.CloneableBootstrapContext>`	Bootstrap context map (unique name to a cloneable bootstrap context) which allows developers to bind (through `jboss-ra.xml`) their resource adapter to a specific bootstrap context instance.
`PrintStream`	`java.io.PrintStream`	Specifies which print stream that should be used to handle the `LogWriters`
`ScopeDeployment`	`boolean`	Should each deployment be scoped (isolated) from the container. This feature allows deployment of libraries of a different version than used in the container

Property	Type	Description
		environment. Default: `false`

5.3.2. Security

The Java EE Connector Architecture 1.6 specification allows units of `javax.resource.spi.Work` to be executed in a specific security context.

This is done through the use of Java Authentication Service Provider Interface for Containers (JSR-196) call backs using the `javax.security.auth.callback.Callback` interface.

The support is activated by letting the work instance implement the

```
javax.resource.spi.work.WorkContextProvider
```

interface and returning an instance of `javax.resource.spi.work.SecurityContext`.

There is currently support for injecting a user/roles setup based on the files

```
config/users.properties
config/roles.properties
```

The format of the `users.properties` file is

```
username1=password1
username2=password2
```

The format of the `roles.properties` file is

```
username1=role1,role2
username2=role3,role4
```

The user/roles setup can be configured through the `UsersRoles` bean in the `config/bootstrap/jca.xml` file.

```
<!-- Users / roles -->
<bean name="UsersRoles"
      interface="org.jboss.jca.core.spi.security.Callback"
      class="org.jboss.jca.core.security.UsersRoles">
  <property name="UsersProperties">${jboss.jca.home}/config/users.properties</property>
  <property name="RolesProperties">${jboss.jca.home}/config/roles.properties</property>
</bean>
```

```
```

5.4. Web server

The JBoss JCA project features a web server which is used to serve web archive deployments.

The configuration is done in the

```
system/web.xml
```

file.

Table 5.2. Web server

Property	Type	Description
`Port`	`int`	Set the port for the web server Default: `8080`

The web server can be removed from the environment by removing the `web.xml` file in

```
system/
```

Furthermore all .WAR files in the same directory should be removed too.

6

Deployment

The JBoss JCA distribution contains a deploy/ directory where all deployments should be deployed to.

6.1. Deploying resource adapters

Resource adapters (.rar) are deployed by copying the resource adapters into the deploy/ directory

```
cp example.rar jboss-jca-1.0.0/deploy
```

on a Un*x based system or

```
copy example.rar jboss-jca-1.0.0\deploy
```

on Windows.

7

Running

7.1. Starting the container

The JBoss JCA container is started by entering the bin/ directory

```
cd jboss-jca-1.0.0/bin
```

and executing

```
./run.sh
```

on a Un*x based system or

```
run.bat
```

on Windows.

The command takes an optional -b argument to define the binding address of the naming server

```
./run.sh -b 192.168.0.199
```

Once the container has started you should see a log entry like

```
13:33:10,999 INFO  [Main] Server started in 941ms
```

in the console where the command was executed.

After the container has started you can browse to

```
http://localhost:8080
```

to view the project documentation and use the administration console.

7.2. Command line interface

The JBoss JCA container can be controlled by a command line interface.

7.2.1. Deploy

You can deploy a resource adapter archive (.rar) using

```
java -jar fungal-cli.jar deploy <file>
```

where `file` specifies the resource adapter archive.

7.2.2. Undeploy

You can undeploy a resource adapter archive (.rar) using

```
java -jar fungal-cli.jar undeploy <file>
```

where `file` specifies the resource adapter archive.

7.2.3. Shutdown

You can shutdown the JBoss JCA environment by

```
java -jar fungal-cli.jar shutdown
```

8

Validator

8.1. Introduction

The JBoss JCA container features a validator which checks resource adapter archives against the Java Connector Architecture (JCA) specification.

The validator is doing a static analysis of the resource adapter classes and checks them against the rules defined in the validator.

The validator is used in the deployer chain of the JCA container, and is available as a standalone tool and an Apache Ant too.

8.2. Reports

The validator works by scanning the resource adapter in question and output a report which lists which rules have been violated.

An example could be

```
Severity: ERROR
Section: 19.4.2
Description: A ResourceAdapter must implement a "public int hashCode()" method.
Code: com.mycompany.myproject.ResourceAdapterImpl

Severity: ERROR
Section: 19.4.2
Description: A ResourceAdapter must implement a "public boolean equals(Object)" method.
Code: com.mycompany.myproject.ResourceAdapterImpl
```

which means that `com.mycompany.myproject.ResourceAdapterImpl` is missing an `equals` and `hashCode` implementation.

Table 8.1. Validator report

Key	Desciption
`Severity`	Specifies the severity of the rule. • ERROR: Critical error which must be fixed in order for the resource adapter to operate correctly.

Key	Desciption
	• WARN: Error which should be fixed in order for the resource adapter to operate correctly.
`Section`	A reference to a section in the Java Connector Architecture specification where the requirement is defined.
`Descrption`	A short description of the rule.
`Code`	The class which triggered the rule.

8.3. Running the standalone validator

The validator can be run on the command line by

```
java -jar jboss-jca-validator.jar <file>
```

The reports will be generated into the current directory under the name of `<file>.log`.

8.4. Apache Ant integration

The validator integrates with Apache Ant such that you can generate the reports directly from your build environment before deploying the resoruce adapter into the JBoss JCA container.

First you have to define the `taskdef` for the task

```
<taskdef name="validator"
         classname="org.jboss.jca.validator.ant.ValidatorTask"
         classpathref="jbossjca.lib.path.id"/>
```

See the Apache Ant documentation for additional instructions on installation.

8.4.1. Usage

```
<validator rarFile="${myArchive.rar}" outputDir="${report.dir}"/>
```

Table 8.2. Apache Ant: validator

Key	Value
`rarFile`	The resource adapter file
`outputDir`	The directory where the reports should be generated

Key	Value
`classpath`	A classpath to resolve additional dependencies against

9

Code generator

9.1. Introduction

The JBoss JCA project includes a resource adapter code generator which can generate a complete code skeleton that will help developers get started with their development tasks.

9.2. Functionality

The code generator will generate a resource adapter code skeleton based on the user input. The code generator supports

- Resource adapter using JCA 1.6 annotations
- Resource adapter using JCA 1.6 metadata
- Resource adapter using JCA 1.5
- Resource adapter using JCA 1.0
- Apache Ant build environment

9.3. Running the tool

The code generator can be run on the command line by

```
./codegenerator.sh
```

from the `doc/codegenerator` directory.

The code generator supports the following arguments

Table 9.1. Code generator arguments

Argument	Desciption
`-o`	Specifies the output directory for the code skeleton.

The developer must then answer various questions regarding the properties of the resource adapter.

9.4. Generated code

The generated code will consist of the classes making up the resource adapter and a test suite environment based on the embedded distribution.

The following targets are supported in the Apache Ant build environment

Table 9.2. Apache Ant build environment

Target	Desciption
`compile`	Compiles all the files
`rar`	Builds the resource adapter archive
`prepare-test`	Prepares the test environment
`test`	Executes the tests
`docs`	Generates the documentation

10

Embedded

10.1. Overview

The JBoss JCA embedded configuration provides a way of running a JCA container in-VM.

The configuration is useful when you want a

- JCA container within your environment
- JCA container when doing unit testing

Especially the ability to unit test your resource adapter archives before deploying them into a testing or a production environment will benefit developers.

10.2. Deployment

Currently you will need all the JAR files located in the

```
$JBOSS_JCA_HOME/lib
```

directory as well as sub-directories on your application class loader - f.ex.

```
java -classpath allthejarfiles.jar yourapp
```

in order to use the embedded configuration.

10.3. Usage

JBoss JCA Embedded supports both a simple and an advanced usage model, using pre-assembled resource adapter archives (.rar) or dynamic resource adapter archives based on ShrinkWrap.

10.3.1. Simple usage

The code sample below shows a simple usage of deploying a pre-assembled resource adapter archive into the JBoss JCA Embedded environment.

```
import org.jboss.jca.embedded.EmbeddedJCA;

import java.net.URL;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

public class MyTestCase
{
   /** Embedded */
   private static EmbeddedJCA embedded;

   /** JNDI prefix */
   private static final String JNDI_PREFIX = "java:/eis/";

   /**
    * Simple test to verify deployment of myresourceadapter.rar
    * @throws Throwable throwable exception
    */
   @Test
   public void testDeployment() throws Throwable
   {
      URL archive = getURL("myresourceadapter.rar");

      Context context = null;

      try
      {
         embedded.deploy(archive);

         context = new InitialContext();
         Object o = context.lookup(JNDI_PREFIX + "myresourceadapter");
         assertNotNull(o);
      }
      catch (Throwable t)
      {
         fail(t.getMessage());
      }
      finally
      {
         embedded.undeploy(archive);

         if (context != null)
         {
            try
            {
               context.close();
            }
            catch (NamingException ne)
            {
               // Ignore
            }
         }
      }
   }

   @BeforeClass
   public static void beforeClass() throws Throwable
   {
```

```
      // Create an embedded JCA instance
      embedded = new EmbeddedJCA();

      // Startup
      embedded.startup();
   }

   @AfterClass
   public static void afterClass() throws Throwable
   {
      // Shutdown
      embedded.shutdown();
   }
}
```

Note

Note that, the url for the archive must end with the `.rar` extension - either representing a file or a directory.

See the JBoss JCA Embedded API documentation for additional functionality.

10.3.2. Advanced usage

The JBoss JCA Embedded container environment supports the following open source testing projects:

1. ShrinkWrap [http://www.jboss.org/community/wiki/ShrinkWrap]
2. Arquillian [http://community.jboss.org/en/arquillian]

These extensions allow the developer to use the embedded platform with greater ease as there doesn't have to be a physical representation of the resource adapter archive located to the disk.

The Arquillian integration furthermore allows the developer to leave all the embedded container setup to the integration instead.

10.3.2.1. ShrinkWrap integration

The code sample below shows an advanced usage of deploying a dynamic ShrinkWrap resource adapter archive into the JBoss JCA Embedded environment.

```


import org.jboss.jca.embedded.EmbeddedJCA;
import org.jboss.jca.embedded.rars.simple.MessageListener;
import org.jboss.jca.embedded.rars.simple.TestActivationSpec;
import org.jboss.jca.embedded.rars.simple.TestConnection;
import org.jboss.jca.embedded.rars.simple.TestConnectionInterface;
import org.jboss.jca.embedded.rars.simple.TestManagedConnection;
import org.jboss.jca.embedded.rars.simple.TestManagedConnectionFactory;
import org.jboss.jca.embedded.rars.simple.TestResourceAdapter;

import java.util.UUID;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
```

```
import org.jboss.logging.Logger;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.spec.JavaArchive;
import org.jboss.shrinkwrap.api.spec.ResourceAdapterArchive;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

public class ShrinkWrapTestCase
{
   private static Logger log = Logger.getLogger(ShrinkWrapTestCase.class);

   /** Embedded */
   private static EmbeddedJCA embedded;

   /** JNDI prefix */
   private static final String JNDI_PREFIX = "java:/eis/";

   /**
    * Basic ShrinkWrap ResourceAdapterArchive test case
    * @exception Throwable Thrown if case of an error
    */
   @Test
   public void testBasic() throws Throwable
   {
      String deploymentName = UUID.randomUUID().toString();

      ResourceAdapterArchive raa = ShrinkWrap.create(ResourceAdapterArchive.class,
                                                     deploymentName + ".rar");

      JavaArchive ja =
         ShrinkWrap.create(JavaArchive.class, UUID.randomUUID().toString() + ".jar");

      ja.addClasses(MessageListener.class, TestActivationSpec.class,
                    TestConnection.class, TestConnectionInterface.class,
                    TestManagedConnection.class, TestManagedConnectionFactory.class,
                    TestResourceAdapter.class);

      raa.addLibrary(ja);
      raa.addManifestResource("simple.rar/META-INF/ra.xml", "ra.xml");

      Context context = null;

      try
      {
         embedded.deploy(raa);

         context = new InitialContext();
         Object o = context.lookup(JNDI_PREFIX + deploymentName);
         assertNotNull(o);
      }
      catch (Throwable t)
      {
         log.error(t.getMessage(), t);
         fail(t.getMessage());
      }
      finally
      {
         embedded.undeploy(raa);

         if (context != null)
         {
            try
            {
```

```
               context.close();
            }
            catch (NamingException ne)
            {
               // Ignore
            }
         }
      }
   }

   /**
    * Lifecycle start, before the suite is executed
    * @exception Throwable Thrown if case of an error
    */
   @BeforeClass
   public static void beforeClass() throws Throwable
   {
      // Create and set an embedded JCA instance
      embedded = new EmbeddedJCA();

      // Startup
      embedded.startup();
   }

   /**
    * Lifecycle stop, after the suite is executed
    * @exception Throwable Thrown if case of an error
    */
   @AfterClass
   public static void afterClass() throws Throwable
   {
      // Shutdown embedded
      embedded.shutdown();

      // Set embedded to null
      embedded = null;
   }
}
```

Note

Note that, the name for the `ResourceAdapterArchive` must end with the `.rar` extension.

See the ShrinkWrap [http://www.jboss.org/community/wiki/ShrinkWrap] web site for a full description of the project and additional documentation.

10.3.2.2. Arquillian integration

The code sample below shows an advanced usage of deploying a dynamic ShrinkWrap resource adapter archive into the JBoss JCA Embedded environment using Arquillian.

This setup allows the developer to skip the entire JBoss JCA Embedded container setup and handling of its lifecycle methods.

```
package org.jboss.jca.embedded.unit;

import org.jboss.jca.embedded.rars.simple.MessageListener;
import org.jboss.jca.embedded.rars.simple.TestActivationSpec;
```

```
import org.jboss.jca.embedded.rars.simple.TestConnection;
import org.jboss.jca.embedded.rars.simple.TestConnectionFactory;
import org.jboss.jca.embedded.rars.simple.TestConnectionInterface;
import org.jboss.jca.embedded.rars.simple.TestConnectionManager;
import org.jboss.jca.embedded.rars.simple.TestManagedConnection;
import org.jboss.jca.embedded.rars.simple.TestManagedConnectionFactory;
import org.jboss.jca.embedded.rars.simple.TestResourceAdapter;

import java.util.UUID;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.jboss.arquillian.api.Deployment;
import org.jboss.arquillian.junit.Arquillian;
import org.jboss.logging.Logger;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.spec.JavaArchive;
import org.jboss.shrinkwrap.api.spec.ResourceAdapterArchive;

import org.junit.Test;
import org.junit.runner.RunWith;
import static org.junit.Assert.*;

/**
 * Unit test for Arquillian integration
 *
 * @author <a href="mailto:jesper.pedersen@jboss.org">Jesper Pedersen</a>
 */
@RunWith(Arquillian.class)
public class ArquillianTestCase
{
   // -------------------------------------------------------------------------------||
   // Class Members -----------------------------------------------------------------||
   // -------------------------------------------------------------------------------||

   private static Logger log = Logger.getLogger(ArquillianTestCase.class);

   private static final String JNDI_PREFIX = "java:/eis/";
   private static String deploymentName = null;

   /**
    * Define the deployment
    * @return The deployment archive
    */
   @Deployment
   public static ResourceAdapterArchive createDeployment()
   {
      deploymentName = UUID.randomUUID().toString();

      ResourceAdapterArchive raa =
         ShrinkWrap.create(ResourceAdapterArchive.class, deploymentName + ".rar");

      JavaArchive ja = ShrinkWrap.create(JavaArchive.class,
                                         UUID.randomUUID().toString() + ".jar");
      ja.addClasses(MessageListener.class, TestActivationSpec.class, TestConnection.class,
                    TestConnectionFactory.class, TestConnectionManager.class,
                    TestConnectionInterface.class, TestManagedConnection.class,
                    TestManagedConnectionFactory.class, TestResourceAdapter.class);

      raa.addLibrary(ja);
      raa.addManifestResource("simple.rar/META-INF/ra.xml", "ra.xml");

      return raa;
   }
```

```
   //------------------------------------------------------------------------------------||
   // Tests -----------------------------------------------------------------------------||
   //------------------------------------------------------------------------------------||

   /**
    * Basic
    * @exception Throwable Thrown if case of an error
    */
   @Test
   public void testBasic() throws Throwable
   {
      Context context = null;

      try
      {
         context = new InitialContext();
         Object o = context.lookup(JNDI_PREFIX + deploymentName);
         assertNotNull(o);
      }
      catch (Throwable t)
      {
         log.error(t.getMessage(), t);
         fail(t.getMessage());
      }
      finally
      {
         if (context != null)
         {
            try
            {
               context.close();
            }
            catch (NamingException ne)
            {
               // Ignore
            }
         }
      }
   }
}
```

Note

Note that, the name for the `ResourceAdapterArchive` must end with the `.rar` extension.

See the Arquillian [http://community.jboss.org/en/arquillian] web site for a full description of the project and additional documentation.

11

Community

11.1. Website

The website contains the latest information about the project and links to important information.

The website is located at http://www.jboss.org/jca/

11.2. User forum

The user forum is where we discuss matters about the usage of the JBoss JCA project.

Our forum is located at http://www.jboss.org/index.html?module=bb&op=viewforum&f=136

11.3. Developer forum

The developer forum is where we discuss the implementation of the JBoss JCA project. This means the internals of the project and not how the project is used.

User questions doesn't belong here - they should go in the user forum instead.

The forum is located at http://www.jboss.org/index.html?module=bb&op=viewforum&f=165

11.4. Issue tracking

We are using JIRA to manage our issues in the project.

These are divided into the following categories

- Feature Request: A feature that you would like see implemented.
- Bug: A software defect.

For all of these you should post your request to our user forum first.

The rest of the categories are for team use only.

Our issue tracking system located at https://jira.jboss.org/jira/browse/JBJCA

12

Troubleshooting

12.1. I think I have found a bug

If you think you have found a bug you should verify this by posting to our forum first.

Our forum is located at http://www.jboss.org/index.html?module=bb&op=viewforum&f=136

You can also search our issue tracking system located at https://jira.jboss.org/jira/browse/JBJCA

12.2. I would like to implement a feature

So you have found an area where you are missing a feature and would like to submit a patch for it, great !

There are a couple of steps to get a feature included

First, you should create a new thread in our development forum where you describe the feature, its design and implementation.

Once there is an agreement on the feature and the design you should proceed with creating the patch.

To maximize your chances of getting the feature in the official build as soon as possible make sure that you run through the following steps:

```
ant clean test
ant clean checkstyle
ant clean findbugs
ant clean cobertura
```

All these should show that,

1. All your test cases for the feature is passing
2. Your code is correctly formatted accoording to project rules
3. There isn't any bug reports from the Findbugs environment
4. There is full code coverage based on the Cobertura report

when done, create a JIRA task (Patch) in our JIRA environment. See the developer guide for additional details.

Happy Coding !

12.3. How do I ?

We can't cover every single issue in this guide, so feel free to drop by our forums to see if a solution has already been provided. Otherwise feel free to ask your question there.

Our forum is located at http://www.jboss.org/index.html?module=bb&op=viewforum&f=136