Teiid - Scalable Information Integration

1

Teiid Administrator's Guide

7.1

Installation Guide

Starting with the 7.0 release Teiid needs to be installed into an existing JBoss AS installation, which is entirely different from previous versions.

Note

Teiid does not support the "embedded" mode in 7.1 version. ("embedded" will be coming in a future release).

1.1. Installation

Steps to install Teiid

1. Download the *JBoss AS 5.1.0* [http://www.jboss.org/jbossas/downloads.html] application server. Install the server by unzipping into a known location. Ex: /apps/jboss-5.1.0
2. Download *Teiid 7.1* [http://www.jboss.org/teiid/downloads.html]. Unzip the downloaded artifact inside any "profile" in the JBoss AS installation. Teiid 7.1 uses a JBoss AS service called the "profile service" that is only installed in "default" and "all" profiles, so installing into one of these profiles is required. The default profile is the typical installation location, for example "<jboss-install>/server/default". The Teiid runtime directory structure matches JBoss profiles directly - it is just an overlay.
3. Start the JBoss AS server by executing "<jboss-install>/bin/run.sh" if you installed in the "default" profile. Otherwise use "<jboss-install>/bin/run.sh -c <profilename>"
4. That it!. JBoss AS and Teiid are now installed and running. See below instructions to customize various settings.
5. Once VDBs have been deployed, users can now connect their JDBC applications to Teiid. If you need help on connecting your application to the Teiid using JDBC check out the "Client Developer's Guide".

1.2. Directory Structure Explained

Example 1.1. Directory Structure

This shows the contents of the Teiid 7.1 deployment. The directory structure is exactly the same under any JBoss profile.

```
teiid
```

```
/conf
  /props
    teiid-security-roles.properties
    teiid-security-users.properties
  jboss-teiid-log4j.xml
/deploy
  /teiid
     /connectors
     teiid-jboss-beans.xml
     teiid-connector-templates-jboss-beans.xml
  admin-console.war
/deployers
  /teiid.deployer
/lib
/teiid-examples
```

1.2.1. /deploy/teiid/teiid-jboss-beans.xml

Master configuration file for Teiid system. This file contains its own documentation, so refer to the file for all the available properties to configure.

1.2.2. /deploy/teiid/connectors

This directory contains all the translator JAR and connector RAR files that are supplied as part of the Teiid installation.

1.2.3. /conf/props

Relevant Files

- /teiid-security-users.properties
- /teiid-security-roles.properties

These files define the allowed users and their defined roles in Teiid using the default security domain. Edit these files to add uses. If you want to use a different security domain look for details in main configuration file.

1.2.4. /conf/jboss-teiid-log4j.xml

This file contains the Teiid specific logging contexts to be included in the "jboss-log4j.xml" file. If you need to turn ON or OFF specific logging in Teiid, then copy the contents of this file into "jboss-log4j.xml" in the installation directory. See the Developers Guide for more on customizing logging.

1.2.5. admin-console.war

This file has the required files for Teiid JOPR plugin. To see the Teiid's "admin-console", go to http://<host>:<port>/admin-console

1.2.6. /deployers/teiid.deployer

This directory contains Teiid runtime specific configuration files and its libraries. These configuration files define VDB deployers, connector binding deployers etc. Typically user never need to edit any files in this directory.

1.2.7. lib

This directory contains Teiid client libraries. It has the Teiid JDBC driver jar, "teiid-7.1-client.jar", and also contains "teiid-hibernate-dialect-7.1.jar" that contains Teiid's Hibernate dialect.

1.2.8. teiid-examples

This directory contains some examples of how Teiid can be used. Contains artifacts need by the Getting Started Guide. Also contains some sample XML files to define the connectors.

1.2.9. teiid-docs

This directory contains the PDF documents related Teiid and Teiid development.

Deploying VDBs in Teiid 7

A *VDB* [http://www.jboss.org/teiid/basics/virtualdatabases.html]is the primary means to define a Virtual Database in Teiid. A user can create a VDB using *Teiid Designer* [http://www.jboss.org/teiiddesigner.html] or follow the instructions in the Reference Guide to create a "Dynamic VDB" without Teiid Designer.

2.1. Deploying a VDB

Once you have a "VDB" built it can be deployed in Teiid runtime in different ways.

2.1.1. Direct File Deployment

Copy the VDB file into the "<jboss-install>/server/<profile>/deploy" directory. Make sure that there are no other VDB files with the same name. If a VDB already exists with the same name, then this VDB will be replaced with the new VDB. This is the simplest way to deploy a VDB. This is mostly designed for quick deployment during development, when the Teiid server is available locally on the developer machine.

2.1.2. Admin Console Deployment (Web)

Use the JOPR-based web console configuration system. Check out the JOPR plugin at:

```
http://<host>:<port>/admin-console
```

More details for this can be found in the *Admin Console VDB deployment section.*This is the easiest way to deploy a VDB to a remote server.

2.1.3. AdminShell Deployment

Teiid provides a groovy based AdminShell scripting tool, which can be used to deploy a VDB. Check out the "deployVDB" method. Consult the *AdminShell documentation* for more information. Note that using the AdminShell scripting, you can automate the deployment of artifacts in your environment.

2.1.4. Admin API Deployment

The Admin API (look in org.teiid.adminpi.*) provides Java API methods that lets a user connect to a Teiid runtime and deploy a VDB. If you need to programmatically deploy a VDB use this method. This method is preferable for OEM users, who are trying to extend the Teiid's capabilities through their applications.

2.2. Deploying VDB Dependencies

Apart from deploying the VDB, the user is also responsible for providing all the necessary dependent libraries, configuration for creating the data sources that are needed by the Schemas (models) defined in "META-INF/vdb.xml" file inside your VDB. This section shows you a sample VDB configuration needed.

For example, if you are trying to integrate Oracle and File sources in your VDB, then you are responsible for providing the JDBC driver for the Oracle source, and any necessary documents and configuration that are needed by the File Translator.

Once the VDB and its dependencies are deployed, then client applications can connect using the JDBC API. If there are any errors in the deployment, a connection attempt will not be successful and a message will be logged. You can use the *admin-console* tool or check the log files for errors and correct them before proceeding.

2.2.1. Creating An Oracle Data Source

1. Copy the Oracle JDBC JAR file into "<jboss-install>/server/<profile>/lib" directory
2. Create a "data source" to the Oracle instance in the JBoss container. This typically done by creating "xxx-ds.xml" file and copying this file to the "<jboss-install>/server/%lt;profile>/deploy" directory. The following shows a "-ds.xml" file template for Oracle. You can also use admin-console to create this data source.

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <xa-datasource>
    <jndi-name>OracleDS</jndi-name>
    <!-- uncomment to enable interleaving <interleaving/> -->
    <isSameRM-override-value>false</isSameRM-override-value>
    <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasource-class>
    <xa-datasource-property name="URL">jdbc:oracle:oci8:@tc</xa-datasource-property>
    <xa-datasource-property name="User">scott</xa-datasource-property>
    <xa-datasource-property name="Password">tiger</xa-datasource-property>
    <!-- Uses the pingDatabase method to check a connection is still valid before handing it out
 from the pool -->
    <!--valid-connection-checker-class-name>
     org.jboss.resource.adapter.jdbc.vendor.OracleValidConnectionChecker
    </valid-connection-checker-class-name-->

    <!-- Checks the Oracle error codes and messages for fatal errors -->
    <exception-sorter-class-name>
     org.jboss.resource.adapter.jdbc.vendor.OracleExceptionSorter
    </exception-sorter-class-name>
```

```
    <!-- Oracles XA datasource cannot reuse a connection outside a transaction once enlisted
  in a global transaction and vice-versa -->
    <no-tx-separate-pools/>
     <!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml (optional) -->
     <metadata>
       <type-mapping>Oracle9i</type-mapping>
     </metadata>
  </xa-datasource>
</datasources>
```

There are templates for all the data sources in the "<jboss-install>/docs/examples/jca" directory.

2.2.2. Creating A File Data Source

File data source uses Teiid specific JCA connector. You need to create "-ds.xml" file and copy it to the "<jboss-install>/server/%lt;profile>/deploy" directory.

Example 2.1. Template for creating a File based data source

```
<?xml version="1.0" encoding="UTF-8"?>
<connection-factories>
  <no-tx-connection-factory>
    <jndi-name>text-source</jndi-name>
    <rar-name>connector-file-{version}.rar</rar-name>
    <connection-definition>javax.resource.cci.ConnectionFactory</connection-definition>
      <config-property name="ParentDirectory" type="java.lang.String">path-to-the-directory-of-
data-file</config-property>
    <max-pool-size>1</max-pool-size>
  </no-tx-connection-factory>
</connection-factories>
```

2.3. VDB Versioning

VDB Versioning is a feature that allows multiple versions of a VDB to be deployed at the same time with additional support to determine which version will be used. When a user connects to Teiid the desired VDB version can be set as a connection property (See the Client Developers Guide). If a specific version is set, then only that VDB may be connected to. If no version is set, then the deployed VDBs are searched for the appropriate version. This feature helps support more fluid migration scenarios.

Setting the version can either be done in the vdb.xml, which is useful for dynamic vdbs, or through a naming convention of the deployment file - vdbname.version.vdb, e.g. marketdata.2.vdb. The

deployer is responsible for choosing an appropriate version number. If the version number is same as an existing VDB existing connections to the previous VDB will remain valid and any new connections will be made to the new VDB - note that the new VDB may be able to use cache entries of the previous VDB.

Once deployed a VDB has an updatable property called connection type, which is used to determine what connections can be made to the VDB. The connection type can be one of:

- *NONE* - disallow new connections.
- *BY_VERSION* - the default setting. Allow connections only if the version is specified or if this is the earliest BY_VERSION vdb and there are no vdbs marked as ANY.
- *ANY* - allow connections with or without a version specified.

The connection type may be changed either through the AdminConsole or the AdminAPI.

2.3.1. Deployment Scenarios

If only a select few applications are to migrate to the new VDB version, then a freshly deployed VDB would be left as BY_VERSION. This ensures that only applications that know the new version may use it.

If only a select few applications are to remain on the current VDB version, then their connection settings would need to be updated to reference the current VDB by its version. Then the newly deployed vdb would have its connection type set to ANY, which allows all new connections to be made against the newer version. If a rollback is needed in this scenario, then the newly deployed vdb would have its connection type set to NONE or BY_VERSION accordingly.

2.4. Migrating VDBs from 6.x

VDBs from prior release contain an older configuration file version that is no longer supported. You can use the migration utility (bin/migrate.sh or bin/migrate.bat) supplied with the *AdminShell* download to update these VDBs for use with Teiid 7. Note - XML and File based sources from previous releases have changed, and require manual changes to the VDB.

Teiid Security

The Teiid system provides a range of built-in and extensible security features to enable the secure access of data.

3.1. Authentication

JDBC clients may use simple passwords to authenticate a user.

Typically a user name is required, however user names may be considered optional if the identity of the user can be discerned by the password credential alone. In any case it is up to the configured security domain to determine whether a user can be authenticated.

Note

By default, access to Teiid is NOT secure. The default login modules are only backed by file based authentication, which has a well known user name and password. The same is true for making connections to the Admin Console application. We DO NOT recommend leaving the default security profile as defined when you are exposing sensitive data.

3.2. Authorization

Authorization covers both administrative activities and data roles. A data role is a collection of permissions (also referred to as entitlements) and a collection of entitled principals or groups. With the deployment of a VDB the deployer can choose which principals and groups have which data roles.

3.3. Encryption

At a transport level Teiid provides built-in support for JDBC over SSL or just sensitive message encryption when SSL is not in use.

Passwords in configuration files however are by default stored in plain text. If you need these values to be encrypted, please see *encrypting passwords* [http://community.jboss.org/wiki/maskingpasswordsinjbossasxmlconfiguration] for instructions on encryption facilities provided by the container.

3.4. LoginModules

LoginModules are an essential part of the JAAS security framework and provide Teiid customizable user authentication and the ability to reuse existing LoginModules defined for JBossAS. See *JBossAS Security* [http://docs.jboss.org/jbossas/admindevel326/html/ch8.chapter.html] for general information on configuring security in JBossAS.

Teiid can be configured with multiple named application policies that group together relevant LoginModules. Each of these application policy (or domains) names can be used to fully qualify user names to authenticate only against that domain. The format for a qualified name is username@domainname.

If a user name is not fully qualified, then the installed domains will be consulted in order until a domain successfully or unsuccessfully authenticates the user.

If no domain can authenticate the user, the logon attempt will fail. Details of the failed attempt including invalid users, which domains were consulted, etc. will be in the server log with appropriate levels of severity.

Note

The security-domain defined for the JDBC connection and Admin connections are separate. The default name of JDBC connection's security-domain is "teiid-security". The default name for Admin connection is "jmx-console". For the Admin connection's security domain, the user is allowed to change which LoginModule that "jmx-console" pointing to, however should not change the name of the domain, as this name is shared between the "admin-console" application.

3.4.1. Built-in LoginModules

JBossAS provides several LoginModules for common authentication needs, such as authenticating from text files or LDAP.

The UsersRolesLoginModule, which utilizes simple text files to authenticate users and to define their groups. The teiid-jboss-beans.xml configuration file contains an example of how to use UsersRolesLoginModule. Note that this is typically not for production use.

See *LDAP LoginModule configuration* [http://community.jboss.org/docs/DOC-11253] for utilizing LDAP based authentication. If you want use a your own Custom Login module, check out the Developer's Guide for instructions.

3.5. Configuring SSL

The Teiid's configuration file `<jboss-install>/server/<profile>/deploy/teiid/teiid-jboss-beans.xml`, contains the properties to configure SSL.

There are two separate connection profiles:

- JDBC Connection - The `JdbcSslConfiguration` bean configuration defines this.
- Admin Connection - The `AdminSslConfiguration` bean configuration defines this.

Example 3.1. Example Configuration

```
<bean name="JdbcSslConfiguration" class="org.teiid.transport.SSLConfiguration">
   <property name="sslEnabled">false</property>
   <property name="keystoreFilename">cert.keystore</property>
   <property name="keystorePassword">passwd</property>
   <property name="keystoreType">JKS</property>
   <property name="sslProtocol">SSLv3</property>
   <property name="keymanagementAlgorithm">false</property>
   <property name="truststoreFilename">cert.truststore</property>
   <property name="truststorePassword">passwd</property>
   <!--  1-way, 2-way, anonymous -->
   <property name="authenticationMode">1-way</property>
   <property name="clientEncryptionEnabled">true</property>
</bean>
```

Properties

1. sslEnabled - true|false, SSL usage either turned ON or OFF
2. sslProtocol- Type of SSL protocol to be used. Default is SSLv3
3. keystoreType - Keystore type created by the keytool. Default "JKS" is used.
4. authenticationMode - anonymous|1-way|2-way, Type of SSL mode, see above about different *SSL modes* available.
5. keymanagementAlgorithm - Type of key algorithm used. Default is based upon the VM, e.g. "SunX509"
6. keystoreFilename - The file name of the keystore, which contains the private key of the Server. This must be available in the classpath of Teiid Server
7. keystorePassword - password for the keystore.
8. truststoreFilename - if "authenticationMode" is chosen as "2-way", then this property must be provided. This is the truststore that contains the public key for the client. Depending upon how you created the keystore and truststores, this may be same file as defined under "keystoreFilename" property.
9. truststorePassword - password for the truststore.

Logging

4.1. General Logging

The Teiid system provides a wealth of information via logging. To control logging level, contexts, and log locations, you should be familiar with *log4j* [http://logging.apache.org/log4j/] and the container's jboss-log4j.xml configuration file. Teiid also provides a <profile>/conf/jboss-teiid-log4j.xml containing much of information from chapter.

All the logs produced by Teiid are prefixed by "org.teiid". This makes it extremely easy to control of of Teiid logging from a single context. Note however that changes to the log configuration file require a restart to take affect

4.1.1. Logging Contexts

While all of Teiid's logs are prefixed with "org.teiid", there are more specific contexts depending on the functional area of the system. Note that logs originating from third-party code, including integrated org.jboss components, will be logged through their respective contexts and not through org.teiid. See the table below for information on contexts relevant to Teiid. See the container's jboss-log4j.xml for a more complete listing of logging contexts used in the container.

Context	Description
com.arjuna	Third-party transaction manager. This will include information about all transactions, not just those for Teiid.
org.teiid	Root context for all Teiid logs. Note: there are potentially other contexts used under org.teiid than are shown in this table.
org.teiid.PROCESSOR	Query processing logs. See also org.teiid.PLANNER for query planning logs.
org.teiid.PLANNER	Query planning logs.
org.teiid.SECURITY	Session/Authentication events - see also AUDIT logging
org.teiid.TRANSPORT	Events related to the socket transport.
org.teiid.RUNTIME	Events related to work management and system start/stop.
org.teiid.CONNECTOR	Connector logs.
org.teiid.BUFFER_MGR	Buffer and storage management logs.
org.teiid.TXN_LOG	Detail log of all transaction operations.
org.teiid.COMMAND_LOG	See *command logging*
org.teiid.AUDIT_LOG	See *audit logging*

Context	Description
org.teiid.ADMIN_API	Admin API logs.

4.2. Command Logging

Command logging captures executing commands in the Teiid System. Both user commands (that have been submitted to Teiid) and data source commands (that are being executed by the connectors) are tracked through command logging.

To enable command logging to the default log location, simply enable the DETAIL level of logging for the org.teiid.COMMAND_LOG context.

To enable command logging to an alternative file location, configure a separate file appender for the DETAIL logging of the org.teiid.COMMAND_LOG context. An example of this is shown below and can also be found in the jboss-log4j.xml distributed with Teiid.

```
<appender name="COMMAND" class="org.apache.log4j.RollingFileAppender">
     <param name="File" value="log/command.log"/>
    <param name="MaxFileSize" value="1000KB"/>
    <param name="MaxBackupIndex" value="25"/>
     <layout class="org.apache.log4j.PatternLayout">
       <param name="ConversionPattern" value="%d %p [%t] %c - %m%n"/>
     </layout>
  </appender>

  <category name="org.teiid.COMMAND_LOG">
    <priority value="INFO"/>
    <appender-ref ref="COMMAND"/>
  </category>
```

See the Developer's Guide to develop a custom logging solution if file based, or any other built-in Log4j, logging is not sufficient.

4.3. Audit Logging

Audit logging captures important security events. This includes the enforcement of permissions, authentication success/failures, etc.

To enable audit logging to the default log location, simply enable the DETAIL level of logging for the org.teiid.AUDIT_LOG context.

To enable audit logging to an alternative file location, configure a separate file appender for the DETAIL logging of the org.teiid.AUDIT_LOG context. An example of this is already in the log4j.xml distributed with Teiid. See the Developer's Guide to develop a custom logging solution if file based, or any other built-in Log4j, logging is not sufficient.

Teiid Admin Console

The Teiid Admin Console is a web based administrative and monitoring tool for Teiid. Teiid's Admin Console is built using the *Embedded JOPR* [http://www.jboss.org/embjopr] library and adds a additional plugin into the Embeeded JOPR program already available in the *JBoss AS* [http://www.jboss.org/jbossas].

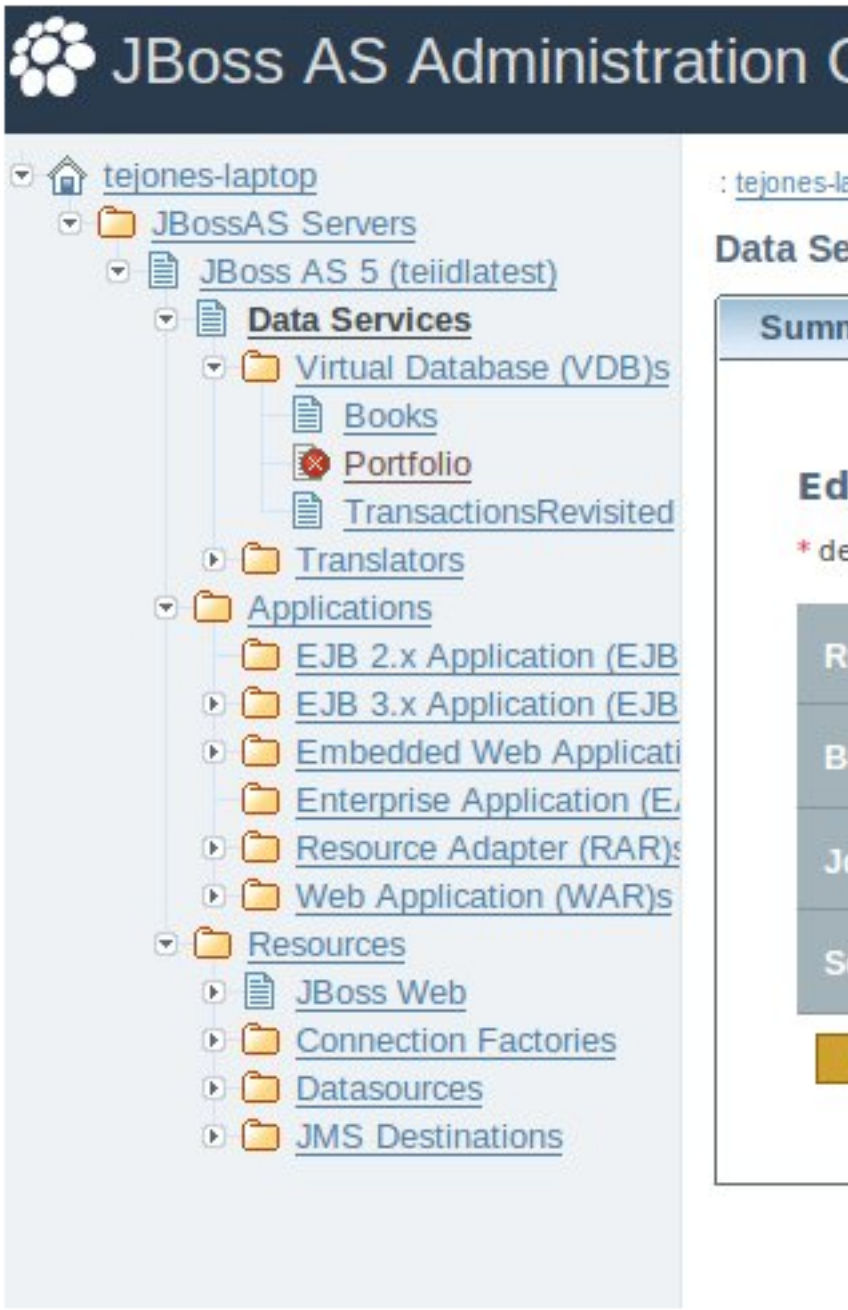
5.1. What can be monitored and/or configured?

Here are the steps to follow to install Teiid

1. *The Teiid Runtime Engine* (Data Services node in the tree)
2. *VDBs* - Virtual databases
   a. *Models*
      i. *Source*- these are physical sources
      ii. *Multi-source* - these are multiple sourced models
      iii. *Logical* - these are virtual sources
   b. *Translator instances*- any Translator instances defined for use by this VDB
3. Translators - These are the extensions to supported datasources that come with Teiid out-of-the-box.

Note

The creation/modification of the datasource is managed by the JBossAS plugin.



5.1.1. Configuration

1. Runtime Engine properties
2. Buffer Service
3. Jdbc Socket configuration
4. Session Service

5.1.2. Metrics

1. Long Running Query count
2. Active Query count
3. Active Session count

5.1.3. Control (Operations)

1. View Long Running Queries
2. View Current Sessions
3. Terminate Session
4. View Current Requests
5. Terminate requests
6. View Current Transactions
7. Terminate Transaction

5.1.4. Deploying the VDB

VDB archive files created it the Designer Tool or Dynamic VDBs can be deployed into Teiid server using the Admin Console.

1. Select the Virtual Database node in the Admin Console tree and click the Add New Resource button.
2. Select the VDB archive file from the file system and click continue.
3. The VDB will deploy if no fatal errors are found in the archive. The status of the VDB will be UP if no errors are found with the models in the VDB.
4. If there are model errors, the VDB will be deployed with a status of DOWN and the errors will be listed on the configuration tab of the VDB. VDBs that are not UP will be marked with a red X in the tree.

Only Model's "connection-jndi-name" can be modified using this tool by clicking on the "configuration" tab, all other proeprties are read-only.

AdminShell

6.1. Introduction

The AdminShell tooling provides scripting based programming environments that enable user to access, monitor and control a Teiid Server. Both the command line and graphical console tools are built on functionality provide by the Groovy (*http://groovy.codehaus.org/*) project. The AdminShell tools can be used in ad-hoc scripting mode or to run pre-defined scripts.

AdminShell features:

1. fully functional programming environment with resource flow control and exception management. See *Groovy* [http://groovy.codehaus.org/] docs for the full power of the language.
2. quick administrative tool. The user can connect to a running Teiid Server and invoke any of the AdminAPI methods, such as "deployVDB" or "stopConnectionFactory", to control the Teiid System. Since this can be script driven, these tasks can be automated and re-run at a later time.
3. simplified data access tool. The user can connect to a VDB, issue any SQL commands, and view the results of the query via *Groovy Sql* [http://groovy.codehaus.org/Database+features] extensions.
4. migration tool. This can be used to develop scripts like moving the Virtual Databases (VDB), Connection Factories, and Configuration from one development environment to another. This will enable users to test and automate their migration scripts before production deployments.
5. testing tool. The JUnit (*http://junit.org* [http://junit.org/]) test framework is built in, see *Groovy Unit Tests* [http://groovy.codehaus.org/Unit+Testing]. User can write regression tests for checking system health, or data integrity that can be used to validate a system functionality automatically instead of manual verification by QA personnel.

6.1.1. Download

AdminShell is distributed along with other Teiid downloads under "teiid-7.1-adminshell-dist.zip" name. Download and unzip this file to any directory. Once you have unzipped the file, in root directory you will find "adminshell" and "adminshell-console" executable scripts to launch the command line and graphical tools respectively.

Windows: Double click or execute "adminshell.cmd"

*nix: Execute the "adminshell.sh" script

6.2. Getting Started

To learn the basics of *Groovy* [http://groovy.codehaus.org/] take a look at their documents and tutorials on their website.

Basic knowledge of the Java programming language and types is required in order to effectively design and develop scripts using the AdminShell. To learn Java language find learning resources at *http://java.sun.com* [http://java.sun.com/].

You can learn about the Teiid AdminAPI either using “adminHelp()” function or by using the JavaDocs.

AdminShell is a specialized version of Groovy which works in several different modes: interactive shell, graphical console, or script run mode. In interactive shell mode (launched via adminshell), the user can invoke connect to a live Teiid system and issue any ad-hoc commands to control the system. The interactive buffer can be used to develop a script and the interactive session input and output can be captured into a log file, more on this later in the document.

In graphical mode (lanched via adminshell-console), the user can develop and run scripts using a text editor that supports syntax highlighting.

In the script run mode, the user can execute/play back previously developed scripts. This mode especially useful to automate any testing or to perform any repeated configurations/migrations changes to a Teiid system.

6.2.1. Essential Rules

To use AdminShell successfully, there are some basic syntactical rules to keep in mind.

1. In interactive shell mode, most commands (as seen by the help command) are used to control shell behavior and are not general Groovy scripting constructs. Admin methods will typically be called using functional notation:

```
connectAsAdmin()
```

2. All commands and functions are case sensitive.
3. An ending semicolon is optional for Groovy statements.
4. If a function requires input parameter(s), they should be declared inside "(" and ")". A function may have more than one parameter. String parameters can be wrapped in double or single quotes. Example:

```
connectAsAdmin("mm://localhost:34413", "user", "password")
```

5. Other Java methods and classes can be used from your scripts, if the required Java class libraries are already in class path. You may place additional jars in the lib directory to have be automatically part of the class path. An example showing an import:

```
import my.package.*;
myObject = new MyClass();
myObject.doSomething();
```

To execute the commands and arbitrary script in interactive mode you enter them first and press enter to execute, then enter the next line, so on.

To exit the tool in the interactive mode, first disconnect if you are connected to the Teiid system by executing “disconnect();” then type "exit". In the script mode, when execution of the script finishes the tool will exit automatically, however you still have to disconnect from Teiid system in the script.

Note: If SSL is turned on the Teiid server, you would need to adjust the connection URL and the client SSL settings as necessary (typically this will only be needed for 2-way SSL).

6.2.2. Help

The adminHelp() methods lists all the available administrative API methods in the AdminShell. Please note that none of the Groovy Shell commands or other available function calls will be shown in this list

```
adminHelp();
```

To get a specific definition about a method and it's required input parameters, use adminHelp("method")

```
adminHelp("deployVDB");

/*
 *Deploy a VDB from file
 */
void deployVDB(
    String /* file name */)
  throws AdminException
  throws FileNotFoundException
```

The sqlHelp() methods lists all Sql extension methods.

```
sqlHelp();
```

To get a specific definition about a method and it's required input parameters, use sqlHelp("method")

6.2.3. Basic Commands

The list below contains some common commands used in AdminShell.

```
println "xxx";   // print something to console

adminHelp();     // shows all the available admin commands;

sql = connect(); // get an extended Groovy Sql connection using connection.properties file

sql.execute(<SQL>); // run any SQL command.

connectAsAdmin(); // connect as admin; no need have the vdb name. SQL commands will not
 work under this connection

println getConnectionName();  // returns the current connection name

useConnection(<connection name>); // switches to using the given connection settings

disconnect();   // disconnects the current connection in the context
```

6.3. Executing a script file

To execute a script file "foo.groovy" in a directory "some/directory" in the interactive comamnd line tool, execute as following

```
. some/directory/foo.groovy
```

"foo.groovy" is read into current context of the shell as if you typed in the whole document. If your script only contained method calls, you can explicitly invoke the call to execute.

Full execute syntax may also be used, and is required outside of the interactive command line tool:

```
evaluate("some/directory/foo.groovy" as File)
```

To execute the same file without entering interactive mode, run

```
./adminshell.sh . some/directory/foo.groovy
```

Parameters can be passed in as Java System properties. For example

```
./adminshell.sh -Dparam=value . some/directory/foo.groovy
```

Inside the script file, you can access these properties using System.getProperty

```
value = System.getProperty(“param”); // will return "value"
```

6.4. Log File and Recorded Script file

During the interactive mode, input is recorded in a history file. This file can be accessed via the up arrow in the interactive shell.

User can also capture the commands entered during a interactive session to their own script file by using “startRecording” and “stopRecording” commands. For example,

```
record start directory/filename.txt
<commands and script ..>
record stop
```

All input and output between the start and stop are captured in the “directory/filename.txt” file. This gives the user an option to capture only certain portions of the interactive session and to later refine a script out of recorded file.

6.5. Default Connection Properties

The file "connection.properties" in the installation directory of the AdminShell defines the default connection properties with which user can connect to Teiid system. The following properties can be defined using this file

```
admin.url = <server host name or ip address>
admin.name = <user name>
admin.password = <password>

jdbc.url = <server host name or ip address>
```

```
jdbc.user = <user name>
jdbc.password = <password>
```

A call to "connect()" or "connectionAsAdmin()" without any input parameters, will connect to the Teiid system using the properties defined in properties file. However, a user can always pass in parameters in the connect method to connect to a same or different server than one mentioned in the “connection.properties”. Look all the all the different connect methods using the “adminHelp()” method.

Note that it is not secure to leave the passwords in clear text, as defined above. Please take necessary measures to secure the properties file, or do not use this feature and always pass in password interactively or some other secure way.

Note: At any given time user can be actively connected to more than one system or have more than one connection to same system. To manage the connections correctly each connection is created given a unique connection name. To learn more about this look at *Handling Multiple Connections*.

6.6. Handling Multiple Connections

Using AdminShell, a user can actively manage more than one connection to a single or multiple Teiid systems. For example, two separate connections can be maintained, one to the development server and one to the integration server at the same time. This is possible because AdminShell supports a feature called named connections.

Every time a connection is made, the connection has an explicit or an implicitly assigned name. If another connect command is executed then a new connection is made with a unique name and execution will be switched to use the new connection. The previous connection will be held as it is in its current state, and will not be closed.

You can use the following command to find out the current connection's name

```
name = getConnectionName();
```

Knowing the names of the connection that user is working with is important to switch the active connection to a previous connection. To switch the active connection, use the following command and supply the name of the connection to be used

```
```

```
useConnection("name");
```

If user supplies the same name as the active connection as they are currently participating in, then this operation will simply return with out any modifications. There is no limitation the number of simultaneous connections.

The following shows an example of using and switching between two connections.

```
// creates a connection
connectAsAdmin();

//capture the connection name
conn1 = getConnectionName();

deployVDB("file.vdb")

// creates a second connection
connectAsAdmin();

conn2 = getConnectionName();

deployVDB("file.vdb")

// switch the connection to "conn1"
useConnection(conn1);

// close the connection in the "conn1"
disconnectAll();
```

Appendix A. AdminShell Frequently Asked Questions

A.1. Why won't the adminhelp command work in the Console tool?

The Console environment does not understand Shell commands (load, help, adminhelp, etc.), since they are not directly supported by Groovy. In the Console you should use the equivalent functional form / Groovy, e.g. instead of adminhelp, adminHelp()

A.2. Are there any pre-built scripts available?

Currently no, but we will provide samples in subsequent releases.

A.3. I have written a very useful script to do XYZ, I would like this to be part of the distribution?

Yes, we would love to hear from users. Please submit the script through the *Teiid JIRA* [https://jira.jboss.org/jira/browse/TEIID], and if this script popular, we will include the script in the scripts library in the following releases.

A.4. What is different between "connectAsAdmin()" and "connect()"?

The connectAsAdmin methods create a contextual connection to the AdminAPI of the Teiid Server. The connect methods return an extension of the Groovy Sql object to be used for Sql calls to the Teiid Server.

A.5. Is IDE support available for writing the scripts?

The Admin Console tool is a light-weight IDE. Full IDE support is available for Groovy, but requires manual manipulation of the class path and script imports. See *using AdminShell methods in other environments*.

A.6. Is debugging support available?

The interactive shell and console do have built-in support for inspection of the current state. Performing line based debugging is beyond the scope of this document.

Appendix B. Other Scripting Environments

The AdminShell methods (named contextual connections, AdminAPI wrapper, and help system) have no direct dependencies on Groovy and can be used in other scripting languages.

To use the AdminShell methods in another language, simply import the static methods and Admin classes into your script. You will also need to ensure that the <adminshell dist>/lib/teiid-7.1-client.jar and <adminshell dist>/lib/teiid-adminshell-7.1.jar are in your class path. The snippet below show import statements that would work in Java, BeanShell, Groovy, etc.

```
import static org.teiid.adminshell.AdminShell.*;
import org.teiid.adminapi.*;
```

Note that the provided shell and console executables automatically have the proper class path set and inject the proper imports into running scripts. If you wish to use scripts in a non-Teiid Groovy environment, you will need to manually add these imports and add the admin/client jars to the class path.

Appendix C. System Properties

Some of Teiid's low-level behavior can be configured via system properties, rather than through configuration files. A typical place to set system properties for JBoss AS launches is in the <jboss-install>/bin/run.conf. A property setting has the format -Dproperty=value

- *org.teiid.allowNanInfinity* - defaults to false. Set to true to allow numeric functions to return NaN (Not A Number) and +-Infinity. Note that these values are not covered by the SQL specification.
- *org.teiid.useValueCache* - defaults to true. Set to false to disable the value cache. Value caching is used dynamically when buffer memory is running low to reuse identical values, but at a computational cost. If there is memory available, you should first increase the buffer memory rather than disabling value caching.
- *org.teiid.ansiQuotedIdentifiers* - defaults to true. Set to false to emulate Teiid 6.x and prior behavior of treating double quoted values without leading identifier parts as string literals, which is not expected by the SQL specification.