Teiid - Scalable Information Integration

1

Teiid Administrator's Guide

7.3

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 7.3 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 7.3* [http://www.jboss.org/teiid/downloads.html]. Unzip the downloaded artifact inside any "profile" in the JBoss AS installation. Teiid 7.3 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. CXF Installation

The usage of CXF is expected for utilizing Salesforce and Web Services connectivity through Teiid. If you do not plan on integrating either of these features, then you may leave JBoss AS with the default "native" web services stack.

1. Download *JBossWS-CXF 3.1.2* [http://www.jboss.org/jbossws/downloads/] and unzip to a temporary location.

2. From the jbossws-cxf-bin-dist directory, save the ant.properties.example file as ant.properties and change the values for jboss510.home, jbossws.integration.target, jboss.server.instance, jboss.bind.address accordingly:

```
...
jboss510.home=<jboss-install>

# The JBoss server under test. This can be [jboss500|jboss501|jboss510|jboss600]
jbossws.integration.target=jboss510

# The JBoss settings
jboss.server.instance=<profile>
jboss.bind.address=<bind address>
```

   The jboss-install location should be the root directory of your AS installation, profile (typically default) should indicate the profile selected for your Teiid installation, and the bind address should be the bind address used when launching JBoss AS (use the value localhost if you do not set the bind address when launching JBoss AS).

3. From the jbossws-cxf-bin-dist directory, install JBossWS-CXF by running the ANT build script:

```
$ant deploy-jboss510
```

4. Optionally run tests to verify that there are no errors with the installation:

```
$ant tests
```

1.3. Directory Structure Explained

Example 1.1. Directory Structure

This shows the contents of the Teiid 7.3 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
```

1.3.1. /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.

1.3.2. /deploy/teiid/connectors

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

1.3.3. /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.

1.3.4. /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.

1.3.5. 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

1.3.6. /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.

1.3.7. lib

This directory contains Teiid client libraries. It has the Teiid JDBC driver jar, "teiid-7.3-client.jar", and also contains "teiid-hibernate-dialect-7.3.jar" that contains Teiid's Hibernate dialect.

1.3.8. teiid-examples

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

1.3.9. 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/removed in Teiid runtime in different ways.

Warning

If *VDB versioning* is not used to give distinct version numbers, overwriting a VDB of the same name will terminate all connections to the old VDB. It is recommended that VDB versioning be used for production systems.

Note

Removing an existing VDB will immediately clean up VDB file resources, but will not automatically terminate existing sessions.

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 models (schemas) defined in "META-INF/vdb.xml" file inside your VDB. 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.

Data source instances may be used by only one VDB, or may be shared with as many VDBs or other applications as makes since for your deployments. Consider sharing connections to sources that have heavy-weight and resource constrained connections.

With the exception of JDBC, each of the data sources supported by has a corresponding .rar (zip format) file in <jboss-install>/server/<profile>/deploy/teiid/connectors. If not using JOPR or other tooling to create your -ds.xml files, you can consult the .rar files META-INF/ra.xml for a full description of how the source can be configured.

Some -ds.xml files may contain passwords or other sensitive information. See the WIKI article *EncryptingDataSourcePasswords* [http://community.jboss.org/wiki/EncryptingDataSourcePasswords] to not store passwords in plain text.

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. JDBC Data Sources

The following is an example highlighting configuring an Oracle data source. The process is nearly identical regardless of the vendor. Typically only the client jar and the setting in the -ds.xml file change.

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

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/<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>
```

2.2.2. File Data Sources

File data sources use a Teiid specific JCA connector. You need to create "-ds.xml" file and copy it to the "<jboss-install>/server/<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>teiid-connector-file.rar</rar-name>
    <connection-definition>javax.resource.cci.ConnectionFactory</connection-definition>
    <config-property name="ParentDirectory">path-to-the-directory-of-data-file</config-property>
  </no-tx-connection-factory>
</connection-factories>
```

2.2.3. Web Service Data Sources

Web service data sources use a Teiid specific JCA connector. You need to create "-ds.xml" file and copy it to the "<jboss-install>/server/<profile>/deploy" directory.

Example 2.2. Template for creating a web service based data source

```
<?xml version="1.0" encoding="UTF-8"?>
<connection-factories>
  <no-tx-connection-factory>
    <jndi-name>somewhere-ws-source</jndi-name>
    <rar-name>teiid-connector-ws.rar</rar-name>
    <connection-definition>javax.resource.cci.ConnectionFactory</connection-definition>
    <config-property name="EndPoint">http://somewhere.com</config-property>
  </no-tx-connection-factory>
</connection-factories>
```

2.2.3.1. CXF Configuration

Each web service data source may choose a particular CXF config file and port configuration. The `ConfigFile` config property specifies the Spring XML configuration file for the CXF Bus and port configuration to be used by connections. If no config file is specified then the system default configuration will be used.

Only 1 port configuration can be used by this data source. You may explicitly set the local name of the port QName to use via the `ConfigName` property. The namespace URI for the QName in your config file should be http://teiid.org. See the sections on WS-Security, Logging, etc. for examples of using the CXF configuration file.

See the *CXF documentation* [http://cxf.apache.org/docs/] for all possible configuration options.

Note

The CXF configuration is currently only applicable to non-binary web service calls.

2.2.3.2. WS-Security

To enable the use of WS-Security, the `SecurityType` should be set to WSSecurity. At this time Teiid does not expect a WSDL to describe the service being used. Thus a Spring XML configuration file is not only required, it must instead contain all of the relevant policy configuration. And just as with the general configuration, each data source is limited to specifing only a single port configration to use.

Example 2.3. Example WS-Security enabled data source

```
<?xml version="1.0" encoding="UTF-8"?>
<connection-factories>
  <no-tx-connection-factory>
    <jndi-name>somewhere-ws-source</jndi-name>
    <rar-name>teiid-connector-ws.rar</rar-name>
    <connection-definition>javax.resource.cci.ConnectionFactory</connection-definition>
    <config-property name="EndPoint">http://somewhere.com</config-property>
          <config-property  name="ConfigFile">${jboss.server.home.dir}/server/default/conf/xxx-
jbossws-cxf.xml</config-property>
    <config-property name="ConfigName">port_x</config-property>
    <config-property name="SecurityType">WSSecurity</config-property>
  </no-tx-connection-factory>
</connection-factories>
```

Corresponding xxx-jbossws-cxf.xml file that adds a timestamp to the SOAP header

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jaxws="http://cxf.apache.org/jaxws"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://cxf.apache.org/jaxws
        http://cxf.apache.org/schemas/jaxws.xsd">

   <jaxws:client name="{http://teiid.org}port_x"
      createdFromAPI="true">
      <jaxws:outInterceptors>
        <bean class="org.apache.cxf.binding.soap.saaj.SAAJOutInterceptor"/>
```

```
        <ref bean="Timestamp_Request"/>
      </jaxws:outInterceptors>
   </jaxws:client>

   <bean
      class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor"
      id="Timestamp_Request">
      <constructor-arg>
         <map>
            <entry key="action" value="Timestamp"/>
         <map>
      </constructor-arg>
   </bean>

</beans>
```

Note that the client port configuration is matched to the data source instance by the QName {http://teiid.org}port_x. The configuration may contain other port confiruations with different local names.

For more information on configuring CXF interceptors, please consult the *CXF documentation* [https://cwiki.apache.org/CXF20DOC/ws-security.html] or the *JBossWS-CXF documentation* [http://community.jboss.org/wiki/JBossWS-StackCXFUserGuide#WSSecurity].

2.2.3.3. Logging

The CXF config property may also be used to control the logging of requests and responses for specific or all ports. Logging, when enabled, will be performed at an INFO level to the org.apache.cxf.interceptor context.

Example 2.4. Example logging data source

```
<?xml version="1.0" encoding="UTF-8"?>
<connection-factories>
  <no-tx-connection-factory>
    <jndi-name>somewhere-ws-source</jndi-name>
    <rar-name>teiid-connector-ws.rar</rar-name>
    <connection-definition>javax.resource.cci.ConnectionFactory</connection-definition>
    <config-property name="EndPoint">http://somewhere.com</config-property>
          <config-property  name="ConfigFile">${jboss.server.home.dir}/server/default/conf/xxx-
jbossws-cxf.xml</config-property>
    <config-property name="ConfigName">port_y</config-property>
  </no-tx-connection-factory>
```

```
</connection-factories>
```

Corresponding xxx-jbossws-cxf.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jaxws="http://cxf.apache.org/jaxws"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd
         http://cxf.apache.org/jaxws
         http://cxf.apache.org/schemas/jaxws.xsd">

    <jaxws:client name="{http://teiid.org}port_y"
        createdFromAPI="true">
        <jaxws:features>
           <bean class="org.apache.cxf.feature.LoggingFeature"/>
        </jaxws:features>
    </jaxws:client>

</beans>
```

2.2.3.4. Transport Settings

The CXF config property may also be used to control low level aspects of the HTTP transport. See the *CXF documentation* [http://cxf.apache.org/docs/client-http-transport-including-ssl-support.html] for all possible options.

Example 2.5. Example Disabling Hostname Verification

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:http-conf="http://cxf.apache.org/transports/http/configuration"
       xsi:schemaLocation="http://cxf.apache.org/transports/http/configuration
                           http://cxf.apache.org/schemas/configuration/http-conf.xsd
                           http://www.springframework.org/schema/beans
                             http://www.springframework.org/schema/beans/spring-beans.xsd">

  <http-conf:conduit name="{http://teiid.org}port_z.http-conduit">
    <!-- WARNING ! disableCNcheck=true should NOT be used in production -->
    <http-conf:tlsClientParameters disableCNcheck="true" />

  </http-conf:conduit>
```

```
</beans>
```

2.2.4. Salesforce Data Sources

Salesforce data sources use a Teiid specific JCA connector. You need to create "-ds.xml" file and copy it to the "<jboss-install>/server/<profile>/deploy" directory.

Example 2.6. Template for creating a Salesforce based data source

```
<?xml version="1.0" encoding="UTF-8"?>
<connection-factories>
  <no-tx-connection-factory>
    <jndi-name>sf-source</jndi-name>
    <rar-name>teiid-connector-salesforce.rar</rar-name>
    <connection-definition>javax.resource.cci.ConnectionFactory</connection-definition>
        <config-property  name="URL">https://test.salesforce.com/services/Soap/u/10.0</config-
property>
    <config-property name="username">username</config-property>
    <config-property name="password">password</config-property>
  </no-tx-connection-factory>
</connection-factories>
```

2.2.5. LDAP Data Sources

LDAP data sources use a Teiid specific JCA connector. You need to create "-ds.xml" file and copy it to the "<jboss-install>/server/<profile>/deploy" directory.

Example 2.7. Template for creating an LDAP based data source

```
<?xml version="1.0" encoding="UTF-8"?>
<connection-factories>
  <no-tx-connection-factory>
    <jndi-name>ldap-source</jndi-name>
    <rar-name>teiid-connector-ldap.rar</rar-name>
    <connection-definition>javax.resource.cci.ConnectionFactory</connection-definition>
    <config-property name="LdapAdminUserDN">cn=x,ou=y,dc=z</config-property>
    <config-property name="LdapAdminUserPassword">password</config-property>
    <config-property name="LdapUrl">ldap://ldapServer:389</config-property>
  </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. If you need authentication, the administrator must configure a LoginModule to be used with Teiid. See below for more information on how configure the Login module in JBoss AS.

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.

3.1.1. Pass-through Authentication

If your client application (web application or Web service) resides in the same JBoss AS instance as Teiid and client application uses a security-domain to handle the security concerns, then you can configure Teiid to use the same security-domain and not force the user to re-authenticate for using Teiid. In this case Teiid looks for a authenticated subject in the calling thread context and uses for its session and authorization purposes. To configure Teiid for this pass-through authentication mechanism, you need change the Teiid's security-domain name to same name as your application's security domain name in the "teiid-jboss-beans.xml" file in the SessionService section. Please note that for this to work, the security-domain must be a JAAS based Login Module and your client application MUST obtain Teiid connection using *Local* Connection, with *PassthroughAuthentication=true* flag set.

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 login 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 and is strongly

recommended that you replace this login module. Please also note that, you can install multiple login modules as part of single security domain configuration and configure them to part of login process. For example, for "teiid-security" domain, you can configure a file based and also LDAP based login modules, and have your user authenticated with either both or single login module.

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

3.4.2. Security at Data Source level

In some use cases, user might need to pass-in different credentials to their data sources based on the logged in user than using the shared credentials for all the logged users. To support this feature, JBoss AS and Teiid provide multiple different login modules to be used in conjunction with Teiid's main security domain. See this *document* [http://community.jboss.org/docs/DOC-9350] for details on configuration. Note that the below directions need to be used in conjunction with this document.

3.4.2.1. CallerIdentity and Trusted Payload

If client wants to pass in simple text password or a certificate or a custom serialized object as token credential to the data source, user can configure "CallerIdentity" login module. Using this login module, user can pass-in same credential that user logged into Teiid security domain to the data source. Here is a sample configuration, this needs to be configured in "teiid-jboss-beans.xml" file.

```
  <application-policy xmlns="urn:jboss:security-beans:1.0" name="teiid-security">
     <authentication>

      <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule" flag="required">
           <module-option name = "password-stacking">useFirstPass</module-option>
         <module-option name="usersProperties">props/teiid-security-users.properties</module-
option>
           <module-option name="rolesProperties">props/teiid-security-roles.properties</module-
option>
        </login-module>

                    <login-module   code="org.jboss.resource.security.CallerIdentityLoginModule"
 flag="required">
           <module-option name = "password-stacking">useFirstPass</module-option>
                                                                    <module-option
 name = "managedConnectionFactoryName">jboss.jca:service=LocalTxCM,name=DefaultDS</
module-option>
        </login-module>
```

```
    </authentication>
  </application-policy>
```

In the -ds.xml file that is defined as the "managedConnectionFactoryName" in the above configuration, you need to add the following element

```
        <security-domain>teiid-security</security-domain>
```

In the above configuration example, in the primary login module "UsersRolesLoginModule" is setup to hold the passwords in the file, and when user logs in with password, the same password will be also set on the logged in Subject after authentication. This credentials can be extracted by the data source by asking for Subject's private credentials.

To use a certificate or serialized object instead of plain password as the token, simply replace the simple text password with Base64 encoded contents of the serialized object. Please note that, encoding and decoding of this object is strictly up to the user as JBoss AS and Teiid will only act like carrier of the information from login module to connection factory. Using this CallerIdentity module, the connection pool for data source is segmented by Subject.

3.4.2.2. Role Based Credential Map

In some use cases, the users are divided by their functionality and they have varied level of security access to data sources. These types of users are identified by their roles as to what they have access to. In the above "CallerIdentity" login scenario, that may be too fine-grained security at data sources, that can lead resource exhaustion as every user has their own separate connection. Using Role based security gives a balance, where the users with same role are treated equally for authentication purposes at the data source. Teiid provides a login module called "RoleBasedCredentialMap" for this purposes, where administrator can define a role based authentication module, where given the role of the user from the primary login module, this module will hold credentail to that role. So, it is container of credentials that map to different roles. If a user has multiple roles, the first role that has the credential will be chosen. Below find the sample configuration.

```
<application-policy xmlns="urn:jboss:security-beans:1.0" name="teiid-security">
    <authentication>

      <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule" flag="required">
          <module-option name = "password-stacking">useFirstPass</module-option>
```

```
        <module-option name="usersProperties">props/teiid-security-users.properties</module-option>
        <module-option name="rolesProperties">props/teiid-security-roles.properties</module-option>
      </login-module>

        <login-module code="org.teiid.jboss.RoleBasedCredentialMapIdentityLoginModule" flag="required">
        <module-option name = "password-stacking">useFirstPass</module-option>
        <module-option name="credentialMap">props/teiid-credentialmap.properties</module-option>
                                                                    <module-option name = "managedConnectionFactoryName">jboss.jca:service=LocalTxCM,name=DefaultDS</module-option>
      </login-module>

    </authentication>
  </application-policy>
```

In the -ds.xml file that is defined as the "managedConnectionFactoryName" in the above configuration, you need to add the following element

```
        <security-domain>teiid-security</security-domain>
```

In the above configuration example, in the primary login module "UsersRolesLoginModule" is setup for logging in the primary user and assign some roles. The "RoleBasedCredentialMap" login module is configured to hold role to password information in the file defined by "credentialMap" property. When user logs in, the role information from the primary login module is taken, and extracts the role's passsword and attaches as a private credential to the Subject. If you want use this for role based trusted token, you can configure the Base64 based endcoding/decoded object as defined above.

You can also encrypt the password instead of plain text password using this module. Just include the encrypted password in the file defined by the "credentialMap" property, and define following properties in the "RoleBasedCredentialMap" login module.

```
        <login-module code="org.teiid.jboss.RoleBasedCredentialMapIdentityLoginModule" flag="required">
```

```
            <module-option name = "password-stacking">useFirstPass</module-option>
            <module-option name="credentialMap">props/teiid-credentialmap.properties</module-
option>
                                                                        <module-option
 name = "managedConnectionFactoryName">jboss.jca:service=LocalTxCM,name=DefaultDS</
module-option>

            <!-- below properties are only required when passwords are encrypted -->
            <module-option name = "pbealgo">PBEWithMD5AndDES</module-option>
            <module-option name = "pbepass">testPBEIdentityLoginModule</module-option>
            <module-option name = "salt">abcdefgh</module-option>
            <module-option name = "iterationCount">19</module-option>
          </login-module>
```

For full details about encryption of the password, please follow this *document* [http://community.jboss.org/docs/DOC-9703]'s "A KeyStore based login module for encrypting a datasource password" section. Be sure to give the same configuration elements in the above configuration, as they are used to encrypt the password.

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 per socket transport.

There are three socket transports, each with it's own SSL configuration:

- JDBC Connections - uses the `JdbcSslConfiguration` bean configuration. Defaults to only encrypt login traffic, none of the other properties are used.
- Admin Connections - uses the `AdminSslConfiguration` bean configuration. Defaults to encrypting all traffic with anonymous SSL, none of the other properties are used.
- ODBC Connections - uses the `OdbcSslConfiguration` bean configuration. Defaults to no SSL.

Example 3.1. Example Configuration

```
<bean name="JdbcSslConfiguration" class="org.teiid.transport.SSLConfiguration">
   <property name="mode">login</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>
</bean>
```

Properties

- mode - diabled|login|enabled, disabled = no transport or message level security will be used. login = only the login traffic will be encrypted at a message level using 128 bit AES with an ephemerial DH key exchange. No other config values are needed in this mode. enabled = traffic will be secured using the other configuration properties.
- sslProtocol- Type of SSL protocol to be used. Default is TLSv1
- keystoreType - Keystore type created by the keytool. Default "JKS" is used.
- authenticationMode - anonymous|1-way|2-way, Type of *SSL Authentication Mode*.
- keymanagementAlgorithm - Type of key algorithm used. Default is based upon the VM, e.g. "SunX509"
- 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.
- keystorePassword - password for the keystore.
- 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.
- truststorePassword - password for the truststore.

3.5.1. SSL Authentication Modes

SSL supports multiple authentication modes. In most secure intranet environments, anonymous is suitable to just bulk encrypt traffic without the need to setup SSL certificates.

- *anonymous* - no certificates are exchanged, settings are not needed for the keystore and truststore properties. Client must have `org.teiid.ssl.allowAnon` set to true (the default) to connect to an anonymous server.
- *1-way* - the server will present a certificate, which is obtained from the keystore related properties. The client should have a truststore configured to accept the server certificate.
- *2-way* - the server will present a certificate, which is obtained from the keystore related properties. The client should have a truststore configured to accept the server certificate. The

client is also expected to present a certificate, which is obtained from its keystore. The client certificate should be accepted by the trust store configured by the truststore related properties.

3.5.2. Encryption Strength

Both anonymous SSL and login only encryption are configured to use 128 bit AES encryption. 1-way and 2-way SSL allow for cipher suite negotiation based upon the default cipher suites supported by the respective Java platforms of the client and server.

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.

Clustering in Teiid

Since Teiid is installed in JBoss AS, there is no separate configuration needed on the part of the user to cluster the Teiid instances. To cluster JBoss AS instances use these *instructions* [http://www.jboss.org/jbossas/docs/5-x.html] then Teiid instances are clustered as well. Just make sure that you installed Teiid in every JBoss AS node before starting the cluster. There is one specific configuration that needs to be done for enabling the replicated (distributed) cache in Teiid. To enable distributed caching, rename the "<jboss-as>/server/<profile>/deploy/teiid/teiid-cache-manager-jboss-beans-rename-me.xml" file to "<jboss-as>/server/<profile>/deploy/teiid/teiid-cache-manager-jboss-beans.xml".

Typically users create clusters to improve the performance of the system through:

1. Load Balancing: Take look at the Client developers guide on how to use load balancing between multiple nodes.
2. Fail Over: Take look at the Client developers guide on how to use fail over between multiple nodes.
3. Distributed Caching: This is automatically done for you once you configure it as specified above.

If would like a clustered deployment of the VDB and data-source artifacts, i.e. deploy artifacts to a central location and let the system propagate deployments every where, then look into *JBoss Farm Deployment* [http://community.jboss.org/wiki/JBossFarmDeployment]. Note that this only supports hot deployments. Take look at some commonly asked questions *here* [http://community.jboss.org/wiki/JoinTheClusterBeforeUpdatingTheFarmDirectory]. If you need more fine grained control, you can use script based deployment, where you control the deployment of artifacts into each node, or JBoss AS "deploy" folder can be configured as a *shared* folder among all the clustered JBoss AS nodes to achieve farming.

Performance Tuning

6.1. Memory Management

The BufferManager is responsible for tracking both memory and disk usage by Teiid. Configuring the BufferManager properly is one of the most important parts of ensuring high performance. See the `<jboss-install>/server/<profile>/deploy/teiid/teiid-jboss-beans.xml` file for all BufferManager settings.

The Teiid engine uses batching to reduce the number of memory rows processed at a given time. The batch sizes may be adjusted to larger values if few clients will be accessing the Teiid server simultaneously.

The `maxReserveBatchColumns` setting determines the total number of batches (with a max of `processorBatchSize` rows) multiplied by their column width that can be held in memory directly by the BufferManager. This number does not include persistent batches held by soft (such as index pages) or weak references. When your installation can dedicate more memory to Teiid, consider increasing this value in proportion to the number of gigabytes you wish Teiid to use - e.g. 2GB on a 32 bit VM would double the value to 32768. For 64 bit VMs you should use a value of approximately 11000 per GB. The BufferManager automatically triggers the use of a canonical value cache when more than 25% of the reserve is in use. This can dramatically cut the memory usage in situations where similar value sets are being read through Teiid, but does introduce a lookup cost. If you are processing large (100s of MBs) of highly unique datasets through Teiid, you should consider *disabling value caching* since it will not significantly reduce memory consumption.

Each intermediate result buffer, temporary LOB, and temporary table is stored in its own set of buffer files, where an individual file is limited to `maxFileSize` megabytes. Consider increasing the storage space available to all such files `maxBufferSpace` if your installation makes use of internal materialization, makes heavy use of SQL/XML, or processes large row counts.

6.2. Threading

Socket threads are configured for each *transport*. They handle NIO non-blocking IO operations as well as directly servicing any operation that can run without blocking.

For longer running operations, the socket threads queue with work the query engine. The query engine has two settings that determine its thread utilization. `maxThreads` sets the total number of threads available for query engine work (processing plans, transaction control operations, processing source queries, etc.). You should consider increasing the maximum threads on systems with a large number of available processors and/or when it's common to issue non-transactional queries with that issue a large number of concurrent source requests. `maxActivePlans`, which should always be smaller than maxThreads, sets the number of the maxThreads that should be used for user query processing. Increasing the maxActivePlans should be considered for workloads with a high number of long running queries and/or systems with a large number of available processors. If memory issues arise from increasing the max threads

and the max active plans, then consider decreasing the processor/connector batch sizes to limit the base number of memory rows consumed by each plan.

6.3. Cache Tuning

Caching can be tuned for cached result (including user query results and procedure results) and prepared plans (including user and stored procedure plans). Even though it is possible to disable or otherwise severely constrain these caches, this would probably never be done in practice as it would lead to poor performance.

Cache statistics can be obtained through the Admin Console or Adminshell. The statistics can be used to help tune cache parameters and ensure a hit ratio.

Plans are currently fully held in memory and may have a significant memory footprint. When making extensive use of prepared statements and/or virtual procedures, the size of the plan cache may be increased proportionally to number of GB intended for use by Teiid.

While the result cache parameters control the cache result entries (max number, eviction, etc.), the result batches themselves are accessed through the *BufferManager*. If the size of the result cache is increased, you may need to tune the BufferManager configuration to ensure there is enough buffer space.

6.4. Socket Transports

Teiid separates the configuration of its socket transports for JDBC, ODBC, and Admin access. Typical installations will not need to adjust the default thread and buffer size settings. At this time, ODBC queries are executed synchronously from the socket thread. Simultaneous long-running queries may exhaust the available threads. Consider increasing the default max threads (15) for ODBC if you expect a higher concurrent load of long-running queries.

6.5. LOBs

LOBs and XML documents are streamed from the Teiid Server to the Teiid JDBC API. Normally, these values are not materialized in the server memory - avoiding potential out-of-memory issues. When using style sheets, or XQuery, whole XML documents must be materialized on the server. Even when using the XMLQuery or XMLTable functions and document projection is applied, memory issues may occur for large documents.

LOBs are broken into pieces when being created and streamed. The maximum size of each piece when fetched by the client can be configured with the `"lobChunkSizeInKB"` property in the `<jboss-install>/server/<profile>/deploy/teiid/teiid-jboss-beans.xml` file. The default value is 100 KB. When dealing with extremely large LOBs, you may consider increasing this value to decrease the amount of round-trips to stream the result. Setting the value too high may cause the server or client to have memory issues.

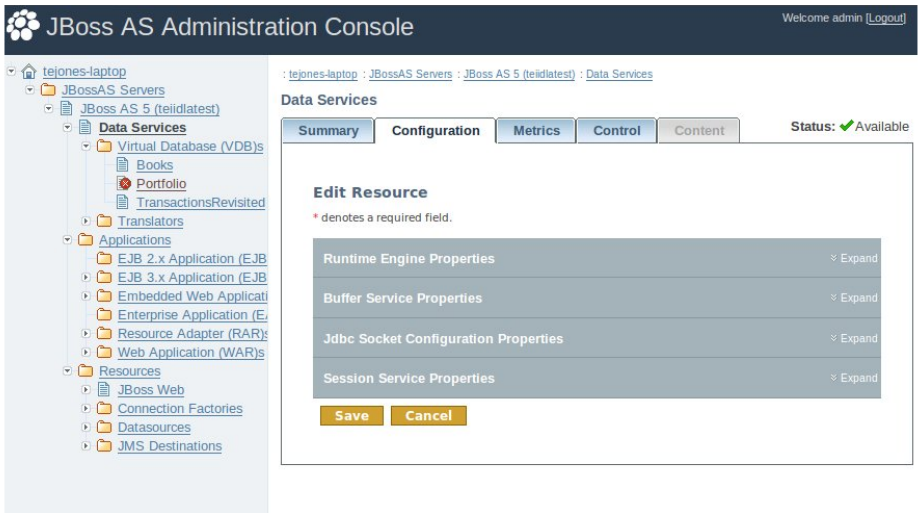
Source LOB values are typically accessed by reference, rather than having the value copied to a temporary location. Thus care must be taken to ensure that source LOBs are returned in a memory-safe manner.

6.6. Other Considerations

When using Teiid in a development environment, you may consider setting the maxSourceRows property in the `<jboss-install>/server/<profile>/deploy/teiid/teiid-jboss-beans.xml` file to reasonably small level value (e.g. 10000) to prevent large amounts of data from being pulled from sources. Leaving the exceptionOnMaxSourceRows set to true will alert the developer through an exception that an attempt was made to retrieve more than the specified number of rows.

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].



7.1. 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.

7.1.1. Configuration

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

7.1.2. Metrics

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

7.1.3. 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

7.1.4. 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

8.1. 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.

8.1.1. Download

AdminShell is distributed along with other Teiid downloads under "teiid-7.3-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

8.2. 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.

8.2.1. 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).

8.2.2. 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")

8.2.3. 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
```

8.3. 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"
```

8.4. 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.

8.5. 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*.

8.6. 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();
```

8.7. Interactive Shell Nuances

The interactive shell uses a special shell interpretter and therefore has different behavior than just writting a script in Groovy. See the *Groovy Shell Documentation* [http://groovy.codehaus.org/Groovy+Shell] for more on its usage. Notable differences:

- Def statements do not define a variable in the context of the Shell, e.g. do not use `def x = 1`, use `x = 1`
- Shell commands (as seen through `help`) using the non-functional shell syntax are only available in the shell.
- Groovy classes using annotations cannot be parsed in the Shell.

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. What does "getAdmin()" call do? Why do I need it?

"getAdmin()" returns this contextual connection object created when you executed "connectAsAdmin()" method. This object implements the interface "org.teiid.adminapi.Admin" and AdminShell commands provided are wrappers around this API. Advanced users can use this API directly if the provided wrapper commands do not meet their needs.

A.6. 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.7. 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-7.3-client.jar and <adminshell dist>/lib/teiid-adminshell-7.3.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 static org.teiid.adminshell.GroovySqlExtensions.*;
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.