Red Hat JBoss Data Virtualization 6.3
Development Guide Volume 6: Metadata Repository Reference Guide

---

This guide is for developers wanting to develop for the Metadata Repository.

Red Hat Customer Content Services

Red Hat JBoss Data Virtualization 6.3 Development Guide Volume 6: Metadata Repository Reference Guide

This guide is for developers wanting to develop for the Metadata Repository.

Red Hat Customer Content Services

Legal Notice

Abstract

This document provides information on concepts and tasks relating to interfacing to Red Hat JBoss Data Virtualization from within client applications.

Table of Contents

Chapter 1. Read Me

1.1. Back Up Your Data

Warning

Red Hat recommends that you back up your system settings and data before undertaking any of the configuration tasks mentioned in this book.

1.2. Variable Name: EAP_HOME

`EAP_HOME` refers to the root directory of the Red Hat JBoss Enterprise Application Platform installation on which JBoss Data Virtualization has been deployed.

1.3. Variable Name: MODE

`MODE` will either be **`standalone`** or **`domain`** depending on whether JBoss Data Virtualization is running in standalone or domain mode. Substitute one of these whenever you see **`MODE`** in a file path in this documentation. (You need to set this variable yourself, based on where the product has been installed in your directory structure.)

1.4. Red Hat Documentation Site

Red Hat's official documentation site is available at https://access.redhat.com/site/documentation/. There you will find the latest version of every book, including this one.

1.5. Target Audience

This reference guide is for application developers that want a better understanding of how ModeShape works, how to take advantage of its advanced features, and how to extend the functionality. This document is also very valuable for community developers because it covers the design and implementation of most of the components that make up ModeShape.

For a higher-level introduction to **ModeShape**, see the *ModeShape Getting Started Guide* document.

Chapter 2. In-Memory Connector

2.1. The In-Memory Connector

The in-memory repository connector is a simple connector that creates a transient, in-memory repository. This repository is used as a very simple in-memory cache or as a standalone transient repository. This connector works well for a readable and writable repository source with small to moderate sized content that need not be permanently saved.

2.2. In-Memory Connector Properties

The **`InMemoryRepositorySource`** class provides a number of JavaBean properties that control its behavior. For more information about these properties, refer to **`org.modeshape.graph.connector.inmemory.InMemoryRepositorySource`** in the Data Services JavaDoc.

2.3. Configuring an In-Memory Connector

One way to configure the in-memory connector is to create **`JcrEngine`** instance with a repository source that uses the **`InMemoryRepositorySource`** class. For example:

```
JcrConfiguration config = ...
config.repositorySource("IMR Store")
      .usingClass(InMemoryRepositorySource.class)
      .setDescription("The repository for our content")
      .setProperty("predefinedWorkspaceNames", new String[] { "staging",
"dev"})
      .setProperty("defaultWorkspaceName", workspaceName);
```

Another way to configure the in-memory connector is to create **`JcrConfiguration`** instance and load an XML configuration file that contains a repository source that uses the **`InMemoryRepositorySource`** class. For example a file named configRepository.xml can be created with these contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:mode="http://www.modeshape.org/1.0"
xmlns:jcr="http://www.jcp.org/jcr/1.0">
    <!--
    Define the sources for the content.  These sources are directly
accessible using the
    ModeShape-specific Graph API.  In fact, this is how the ModeShape JCR
implementation works.  You
    can think of these as being similar to JDBC DataSource objects, except
that they expose
    graph content via the Graph API instead of records via SQL or JDBC.
    -->
    <mode:sources jcr:primaryType="nt:unstructured">
        <!--
        The 'IMR Store' repository is an in-memory source with a single
default workspace (though
        others could be created, too).
        -->
```

```
        <mode:source jcr:name="IMR Store"

mode:classname="org.modeshape.graph.connector.inmemory.InMemoryRepositorySou
rce"
                    mode:description="The repository for our content"
                    mode:defaultWorkspaceName="default">

<mode:predefinedWorkspaceNames>staging</mode:predefinedWorkspaceNames>
       <mode:predefinedWorkspaceNames>dev</mode:predefinedWorkspaceNames>
    </mode:source>

    </mode:sources>

 <!-- MIME type detectors and JCR repositories would be defined below -->
</configuration>
```

The configuration can then be loaded from Java like this:

```
JcrConfiguration config = new
JcrConfiguration().loadFrom("/configRepository.xml");
```

Chapter 3. File System Connector

3.1. The File System Connector

This connector exposes an area of the local file system as a graph of "nt:file" and "nt:folder" nodes. The connector can be configured so that the workspace name is either a path to the directory on the file system that represents the root of that workspace or the name of subdirectory within a root directory (see the **`workspaceRootPath`** property below). Each connector can define whether it allows new workspaces to be created. If the directory for a workspace does not exist, this connector will attempt to create the directory (and any missing parent directories).

By default, this connector is not capable of storing extra properties other than those defined on the **`nt:file`**, **`nt:folder`** and **`nt:resource`** node types. This is because such properties cannot be represented natively on the file system. When the connector is asked to store such properties, the default behavior is to log warnings and then to ignore these extra properties. Obviously this is probably not sufficient for production (unless only the standard properties are to be used). To explicitly turn on this behavior, set the "extraPropertiesBehavior" to "log".

However, the connector can be configured differently. If the "extraPropertiesBehavior" is set to "ignore", then these extra properties will be silently ignored and lost: none will be stored, none will be loaded, and no warnings will be logged. If the "extraPropertiesBehavior" is set to "error", the connector will throw an exception if any extra properties are used.

Perhaps the best setting for general use, however, is to set the "extraPropertiesBehavior" to "store". In this mode, any extra properties are written to files on the file system that are adjacent to the actual file or folder. For example, given a "nt:folder" node that represents the "folder1" directory, all extra properties will be stored in a text file named "folder1.modeshape" in the same parent directory as the "folder1" directory. Similarly, given a "nt:file" node that represents the "file1" file on the file system, all extra properties will be stored in a text file named "file1.modeshape" located next to the "file1" file. Note that the "nt:resource" node for our "nt:file" node also is stored in the same location, so we can't use the "file1.modeshape" file (it is already used for the "nt:file" node), so the connector uses the "file1.content.modeshape" file instead.

Note

The "store" behavior may result in the creation of many "*.modeshape" files, and because of this the "store" behavior is not the default.

3.2. File System Connector Properties

The **`FileSystemSource`** class provides a number of JavaBean properties that control its behavior. For more information about these properties, refer to **`org.modeshape.connector.filesystem.FileSystemSource`** in the Data Services JavaDoc.

3.3. Configuring a File System Connector

One way to configure the file system connector is to create **`JcrConfiguration`** instance with a repository source that uses the **`FileSystemSource`** class. For example:

```
JcrConfiguration config = ...
config.repositorySource("FS Store")
```

```
      .usingClass(FileSystemSource.class)
      .setDescription("The repository for our content")
      .setProperty("workspaceRootPath", "/home/content/someApp")
      .setProperty("defaultWorkspaceName", "prod")
      .setProperty("predefinedWorkspaceNames", new String[] { "staging",
"dev"})
      .setProperty("rootNodeUuid", UUID.fromString("fd129c12-81a8-42ed-aa4b-
820dba49e6f0")
      .setProperty("updatesAllowed", "true")
      .setProperty("creatingWorkspaceAllowed", "false");
```

Another way to configure the file system connector is to create **`JcrConfiguration`** instance and load an XML configuration file that contains a repository source that uses the **`FileSystemSource`** class. For example a file named configRepository.xml can be created with these contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:mode="http://www.modeshape.org/1.0"
xmlns:jcr="http://www.jcp.org/jcr/1.0">
    <!--
    Define the sources for the content.  These sources are directly
accessible using the
    ModeShape-specific Graph API. In fact, this is how the ModeShape JCR
implementation works.  You can
    think of these as being similar to JDBC DataSource objects, except that
they expose graph
    content via the Graph API instead of records via SQL or JDBC.
    -->
    <mode:sources jcr:primaryType="nt:unstructured">
        <!--
        The 'FS Store' repository is a file system source with a three
predefined workspaces
        ("prod", "staging", and "dev").
        -->
        <mode:source jcr:name="FS Store"

mode:classname="org.modeshape.connector.filesystem.FileSystemSource"
         mode:description="The repository for our content"
         mode:workspaceRootPath="/home/content/someApp"
         mode:defaultWorkspaceName="prod"
         mode:creatingWorkspacesAllowed="false"
         mode:rootNodeUuid="fd129c12-81a8-42ed-aa4b-820dba49e6f0"
         mode:updatesAllowed="true" >

<mode:predefinedWorkspaceNames>staging</mode:predefinedWorkspaceNames>

<mode:predefinedWorkspaceNames>dev</mode:predefinedWorkspaceNames>
           <!--
           If desired, specify a cache policy that caches items in memory
for 5 minutes (300 s).
           This fragment can be left out if the connector should not cache
any content.
           -->
           <mode:cachePolicy jcr:name="nodeCachePolicy"

mode:classname="org.modeshape.graph.connector.base.cache.InMemoryNodeCache$P
```

```
athCachePolicy"
             mode:timeToLive="300" />
        </mode:source>
    </mode:sources>

 <!-- MIME type detectors and JCR repositories would be defined below -->
</configuration>
```

The configuration can then be loaded from Java like this:

```
JcrConfiguration config = new
JcrConfiguration().loadFrom("/configRepository.xml");
```

Chapter 4. JPA Connector

4.1. The JPA Connector

This connector stores a graph of any structure or size in a relational database, using a JPA provider on top of a JDBC driver. Currently this connector relies upon some Hibernate-specific capabilities. The schema of the database is dictated by this connector and is optimized for storing a graph structure. (In other words, this connector does not expose as a graph the data in an existing database with an arbitrary schema.)

4.2. JPA Connector Properties

The **`JpaSource`** class provides a number of JavaBean properties that control its behavior. For more information about these properties, refer to **`org.modeshape.connector.store.jpa.JpaSource`** in the JavaDoc.

4.3. Configuring a JPA Connector

One way to configure the JPA connector is to create **`JcrConfiguration`** instance with a repository source that uses the **`JpaSource`** class. For example:

```
JcrConfiguration config = ...
config.repositorySource("JPA Store")
      .usingClass(JpaSource.class)
      .setDescription("The database store for our content")
      .setProperty("dataSourceJndiName", "java:/MyDataSource")
      .setProperty("defaultWorkspaceName", "My Default Workspace")
      .setProperty("autoGenerateSchema", "validate");
```

Of course, setting other more advanced properties would entail calling **`setProperty(...)`** for each. Since almost all of the properties have acceptable default values, however, we don't need to set very many of them.

Another way to configure the JPA connector is to create **`JcrConfiguration`** instance and load an XML configuration file that contains a repository source that uses the **`JpaSource`** class. For example a file named configRepository.xml can be created with these contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:mode="http://www.modeshape.org/1.0"
xmlns:jcr="http://www.jcp.org/jcr/1.0">
    <!--
    Define the sources for the content.  These sources are directly
accessible using the
    ModeShape-specific Graph API.  In fact, this is how the ModeShape JCR
implementation works.  You
    can think of these as being similar to JDBC DataSource objects, except
that they expose
    graph content via the Graph API instead of records via SQL or JDBC.
    -->
    <mode:sources jcr:primaryType="nt:unstructured">
        <!--
        The 'JPA Store' repository is an JPA source with a single default
workspace (though
```

```
        others could be created, too).
        -->
        <mode:source jcr:name="JPA Store"

mode:classname="org.modeshape.connector.store.jpa.JpaSource"
                     mode:description="The database store for our content"
                     mode:dataSourceJndiName="java:/MyDataSource"
                     mode:defaultWorkspaceName="default"
                     mode:autoGenerateSchema="validate"/>
    </mode:sources>

 <!-- MIME type detectors and JCR repositories would be defined below -->
</configuration>
```

The configuration can then be loaded from Java like this:

```
JcrConfiguration config = new
JcrConfiguration().loadFrom("/configRepository.xml");
```

4.4. Simple Model

This database schema model stores node properties as opaque records in the same row as transparent values like the node's namespace, local name, and same-name-sibling index. Large property values are stored separately.

The set of tables used in this model includes:

- Workspaces - the set of workspaces and their names.
- Namespaces - the set of namespace URIs used in paths, property names, and property values.
- Nodes - the nodes in the repository, where each node and its properties are represented by a single record. This approach makes it possible to efficiently work with nodes containing large numbers of children, where adding and removing child nodes is largely independent of the number of children. Since the primary consumer of ModeShape graph information is the JCR layer, and the JCR layer always retrieves the nodes' properties for retrieved nodes, the properties have been moved in-row with the nodes. Properties are still store in an opaque, serialized (and optionally compressed) form.
- Large values - property values larger than a certain size will be broken out into this table, where they are tracked by their SHA-1 has and shared by all properties that have that same value. The values are stored in a binary (and optionally compressed) form.
- Subgraph - a working area for efficiently computing the space of a subgraph; see below
- Options - the parameters for this store's configuration (common to all models)

This database model contains two tables that are used in an efficient mechanism to find all of the nodes in the subgraph below a certain node. This process starts by creating a record for the subgraph query, and then proceeds by executing a join to find all the children of the top-level node, and inserting them into the database (in a working area associated with the subgraph query). Then, another join finds all the children of those children and inserts them into the same working area. This continues until the maximum depth has

been reached, or until there are no more children (whichever comes first). All of the nodes in the subgraph are then represented by records in the working area, and can be used to quickly and efficient work with the subgraph nodes. When finished, the mechanism deletes the records in the working area associated with the subgraph query.

This subgraph query mechanism is extremely efficient, performing one join/insert statement per level of the subgraph, and is completely independent of the number of nodes in the subgraph. For example, consider a subgraph of node A, where A has 10 children, and each child contains 10 children, and each grandchild contains 10 children. This subgraph has a total of 1111 nodes (1 root + 10 children + 10*10 grandchildren + 10*10*10 great-grandchildren). Finding the nodes in this subgraph would normally require 1 query per node (in other words, 1111 queries). But with this subgraph query mechanism, all of the nodes in the subgraph can be found with 1 insert plus 4 additional join/inserts.

This mechanism has the added benefit that the set of nodes in the subgraph are kept in a working area in the database, meaning they don't have to be pulled into memory.

In the Simple model, subgraph queries are used to efficiently process a number of different requests, including **`ReadBranchRequest`** and **`DeleteBranchRequest`**. Processing each of these kinds of requests requires knowledge of the subgraph, and in fact all but the **`ReadBranchRequest`** need to know the complete subgraph.

Warning

Most DBMS systems have built-in sizes for LOB columns (although many allow DB admins to control the size), and thus do not require any special consideration. However, Apache Derby and IBM DB2 require explicit sizes on LOB columns. Currently, the ModeShape database schema has two such columns: the **`MODE_SIMPLE_NODE.DATA`** and **`MODE_LARGE_VALUES.DATA`** columns. The sizes of these columns are sufficiently large (1MB and 1GB, respectively), but attempts to store larger values than these sizes will fail.

Therefore, when using IBM DB2 and Apache Derby, determine the appropriate size of these columns for your environment. For production systems, ModeShape recommends using the DDL generation utility (provided with ModeShape, see above) to generate the DDL for your particular DBMS, and its very easy to adjust that file to specify alternative sizes for the two columns. Alternatively, database administrators can alter the two tables by increasing the size of these columns.

Other databases do not seem to be affected by this issue.

Chapter 5. Disk Connector

5.1. The Disk Connector

This connector stores content in a ModeShape-specific file format on disk. Although this may seem similar in concept to the File System Connector, this connector actually serves a very different purpose. While the File System Connector is designed to expose existing files and folders on the disk and allows ModeShape users to create content that can be read directly by other applications, the Disk Connector is designed for efficiency and stores content in a serialized representation that is not readily accessible to other applications. Conversely, the Disk Connector supports referenceable nodes and can efficiently access nodes by UUID, unlike the File System Connector.

5.2. Disk Connector Properties

The **`DiskSource`** class provides a number of JavaBean properties that control its behavior. For more information about these properties, refer to **`org.modeshape.connector.disk.DiskSource`** in the Red Hat JBoss Data Virtualization JavaDoc.

5.3. Configuring a Disk Connector

One way to configure the file system connector is to create a **`JcrConfiguration`** instance with a repository source that uses the **`DiskSource`** class. For example:

```
JcrConfiguration config = ...
config.repositorySource("Disk Store")
      .usingClass(DiskSource.class)
      .setDescription("The repository for our content")
      .setProperty("repositoryRootPath", "/home/content/someApp")
      .setProperty("defaultWorkspaceName", "prod")
      .setProperty("predefinedWorkspaceNames", new String[] { "staging",
"dev"})
      .setProperty("rootNodeUuid", UUID.fromString("fd129c12-81a8-42ed-aa4b-
820dba49e6f0")
      .setProperty("updatesAllowed", "true")
      .setProperty("creatingWorkspaceAllowed", "false");
```

Another way to configure the file system connector is to create a **`JcrConfiguration`** instance and load an XML configuration file that contains a repository source that uses the **`DiskSource`** class. For example a file named **`configRepository.xml`** can be created with these contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:mode="http://www.modeshape.org/1.0"
xmlns:jcr="http://www.jcp.org/jcr/1.0">
    <!--
    Define the sources for the content.  These sources are directly
accessible using the
    ModeShape-specific Graph API. In fact, this is how the ModeShape JCR
implementation works.  You can
    think of these as being similar to JDBC DataSource objects, except that
they expose graph
    content via the Graph API instead of records via SQL or JDBC.
```

```
    -->
    <mode:sources jcr:primaryType="nt:unstructured">
        <!--
        The 'Disk Store' repository is a disk source with a three predefined
workspaces
        ("prod", "staging", and "dev").
        -->
        <mode:source jcr:name="Disk Store"
         mode:classname="org.modeshape.connector.disk.DiskSource"
         mode:description="The repository for our content"
         mode:repositoryRootPath="/home/content/someApp"
         mode:defaultWorkspaceName="prod"
         mode:creatingWorkspacesAllowed="false"
         mode:rootNodeUuid="fd129c12-81a8-42ed-aa4b-820dba49e6f0"
         mode:updatesAllowed="true" >

<mode:predefinedWorkspaceNames>staging</mode:predefinedWorkspaceNames>

<mode:predefinedWorkspaceNames>dev</mode:predefinedWorkspaceNames>
           <!--
           If desired, specify a cache policy that caches items in memory
for 5 minutes (300 s).
           This fragment can be left out if the connector should not cache
any content.
           -->
           <mode:cachePolicy jcr:name="nodeCachePolicy"

mode:classname="org.modeshape.graph.connector.base.cache.InMemoryNodeCache$M
apCachePolicy"
             mode:timeToLive="300" />
        </mode:source>
    </mode:sources>

 <!-- MIME type detectors and JCR repositories would be defined below -->
</configuration>
```

The configuration can then be loaded from Java like this:

```
JcrConfiguration config = new
JcrConfiguration().loadFrom("/configRepository.xml");
```

Chapter 6. Compact Node Definition Sequencer

6.1. The Compact Node Definition Sequencer

The Compact Node Definition (CND) Sequencer processes JCR CND files to extract nodes and their definitions, inserting them into the repository using JCR built-in types. The node structure generated by this sequencer is equivalent to the node structure used in **`/jcr:system/jcr:nodeTypes`**.

6.2. CND Sequencer Example

This sequencer generates a graph structure that corresponds to what can be found in the /jcr:system/jcr:nodeTypes subtree. As an example, the CND file below:

```
<mode = "http://www.modeshape.org/1.0">

// My CND type
[mode:example] mixin
- mode:name (string) multiple copy
+ mode:child (mode:example) = mode:example version
```

The resulting graph structure (listed in the JCR document view) contains the node type information from the CND file above. Note that comments are not sequenced.

```
<mode:example jcr:primaryType="nt:nodeType"
              jcr:mixinTypes="mode:derived"
              mode:derivedAt="2011-05-13T13:12:03.925Z"
              mode:derivedFrom="/files/docForReferenceGuide.xml"
              jcr:nodeTypeName="mode:example"
              jcr:supertypes="nt:base"
              jcr:isAbstract="false"
              jcr:isMixin="true"
              jcr:isQueryable="true"
              jcr:hasOrderableChildNodes="false">
  <nt:propertyDefinition jcr:name="mode:name"
                         jcr:autoCreated="false"
                         jcr:mandatory="false"
                         jcr:isFullTextSearchable="true"
                         jcr:isQueryOrderable="true"
                         jcr:onParentVersion="copy"
                         jcr:protected="false"
                         jcr:requiredType="STRING"
                         jcr:availableQueryOperators="= > >= < <= <> LIKE"
                         jcr:multiple="true" />
  <nt:childNodeDefinition jcr:name="mode:child"
                          jcr:autoCreated="false"
                          jcr:mandatory="false"
                          jcr:onParentVersion="VERSION"
                          jcr:protected="false"
                          jcr:requiredPrimaryTypes="mode:example"
                          jcr:defaultPrimaryType="mode:example"
                          jcr:sameNameSiblings="false" />
</mode:example>
```

6.3. Configuring a CND Sequencer

1. **Include the relevant libraries**

   Include **`modeshape-sequencer-cnd-`*`VERSION`*`.jar`** in your application.

2. **Choose one of the following for sequencing configuration**

   A. Define sequencing configuration based on standard example provided in ***`SOA-ROOT`*`/eds/modeshape/resources/modeshape-config-standard.xml`**:

```
<mode:sequencer jcr:name="CND File Sequencer"
mode:classname="org.modeshape.sequencer.cnd.CndSequencer">
  <mode:description>
    Sequences CND files loaded under '/files', extracting the
contained node type definitions.
  </mode:description>
  <mode:pathExpression>
    eds-store:default:/files(//(*.cnd[*]))/jcr:content[@jcr:data] =>
eds-store:default:/sequenced/cnd/$1
  </mode:pathExpression>
</mode:sequencer>
```

   B. Configure via **`org.modeshape.jcr.JcrConfiguration`**:

```
JcrConfiguration config = ...

config.sequencer("CND File Sequencer")
      .usingClass("org.modeshape.sequencer.cnd.CndSequencer")
      .loadedFromClasspath()
      .setDescription("Sequences CND files loaded under '/files',
extracting the contained node type definitions.")
      .sequencingFrom("/files(//(*.cnd[*]))/jcr:content[@jcr:data]")
      .andOutputtingTo("/sequenced/cnd/$1");
```

Note

Refer to ***`SOA-ROOT`*`/eds/modeshape/resources/modeshape-config-standard.xml`** for more information.

Chapter 7. XML Document Sequencer

7.1. The XML Document Sequencer

This sequencer stores the structure and data of an XML file in the repository. DTD, entity, comments, and other content are maintained by the sequencer in the output structure.

7.2. XML Document Sequencer Properties

For information about configurable properties relating to the XML Document Sequencer, refer to the **`org.modeshape.sequencer.xml.XmlSequencer`** class in the Red Hat JBoss Data Virtualization JavaDoc.

7.3. XML Document Sequencer Example

For this XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN" "http://www.oasis-
open.org/docbook/xml/4.4/docbookx.dtd" [
<!ENTITY % RH-ENTITIES SYSTEM "Common_Config/rh-entities.ent">
<!ENTITY versionNumber "0.1">
<!ENTITY copyrightYear "2008">
<!ENTITY copyrightHolder "Red Hat Middleware, LLC.">]>
<?target content ?>
<?target2 other stuff ?>
<Cars xmlns:jcr="http://www.jcp.org/jcr/1.0">
    <!-- This is a comment -->
    <Hybrid>
        <car jcr:name="Toyota Prius"/>
    </Hybrid>
    <Sports>
    </Sports>
</Cars>
```

The sequencer will generate this content (listed in document view) if the sequencer **`outputtingTo`** property generates an output path ending in "myxml":

```
 xml jcr:primaryType=nt:unstructured
 <myxml jcr:primaryType="modexml:document"
        jcr:mixinTypes="mode:derived"
        mode:derivedAt="2011-05-13T13:12:03.925Z"
        mode:derivedFrom="/files/docForReferenceGuide.xml"
        modedtd:name="book"
        modedtd:publicId="-//OASIS//DTD DocBook XML V4.4//EN"
        modedtd:systemId="http://www.oasis-
open.org/docbook/xml/4.4/docbookx.dtd">
  <modedtd:entity jcr:primaryType="modedtd:entity"
                  modedtd:name="%RH-ENTITIES"
                  modedtd:systemId="Common_Config/rh-entities.ent" />
  <modedtd:entity[2] jcr:primaryType="modedtd:entity"
                     modedtd:name="versionNumber"
```

```
                    modedtd:value="0.1" />
  <modedtd:entity[3] jcr:primaryType="modedtd:entity"
                    modedtd:name="copyrightYear"
                    modedtd:value="2008" />
  <modedtd:entity[4] jcr:primaryType="modedtd:entity"
                    modedtd:name="copyrightHolder"
                    modedtd:value="Red Hat Middleware, LLC." />
  <modexml:processingInstruction
jcr:primaryType="modexml:processingInstruction"

modexml:processingInstructionContent="content"
                                  modexml:target="target" />
  <modexml:processingInstruction[2]
jcr:primaryType="modexml:processingInstruction"

modexml:processingInstructionContent="other stuff"
                                    modexml:target="target2" />
  <Cars jcr:primaryType="modexml:element">
    <modexml:comment jcr:primaryType="modexml:comment"
                    modexml:commentContent="This is a comment" />
    <Hybrid jcr:primaryType="modexml:element">
      <car jcr:primaryType="modexml:element" />
    </Hybrid>
    <Sports jcr:primaryType="modexml:element" />
  </Cars>
</myxml>
```

The CND used by this sequencer is provided below. Note that the XML sequencer will parse CDATA into its own node in the sequenced output even though the example above does not explicitly demonstrate this.

```
<modexml='http://www.modeshape.org/xml/1.0'>
<modedtd='http://www.modeshape.org/dtd/1.0'>

[modexml:document] > nt:unstructured, mix:mimeType
  - modexml:cDataContent (string)

[modexml:comment] > nt:unstructured
  - modexml:commentContent (string)

[modexml:element] > nt:unstructured

[modexml:elementContent] > nt:unstructured
  - modexml:elementContent (string)

[modexml:cData] > nt:unstructured
  - modexml:cDataContent (string)

[modexml:processingInstruction] > nt:unstructured
  - modexml:processingInstruction (string)
  - modexml:target (string)

[modedtd:entity] > nt:unstructured
  - modexml:name (string)
  - modexml:value (string)
```

```
- modexml:publicId (string)
- modexml:systemId (string)
```

7.4. Configuring an XML Document Sequencer

1. **Include the relevant libraries**

   Include **`modeshape-sequencer-xml-`*`VERSION`*`.jar`** in your application.

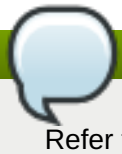
2. **Choose one of the following for sequencing configuration**

   A. Define sequencing configuration based on standard example provided in ***`SOA-ROOT`*`/eds/modeshape/resources/modeshape-config-standard.xml`**:

   ```
   <mode:sequencer jcr:name="XML File Sequencer"
   mode:classname="org.modeshape.sequencer.xml.XmlSequencer">
     <mode:description>
       Sequences XML files loaded under '/files', extracting the
   contents into the equivalent JCR graph structure.
     </mode:description>
     <mode:pathExpression>
       eds-store:default:/files(//)*.xml[*]/jcr:content[@jcr:data] =>
   eds-store:default:/sequenced/xml/$1
     </mode:pathExpression>
   </mode:sequencer>
   ```

   B. Configure via **`org.modeshape.jcr.JcrConfiguration`**:

   ```
   JcrConfiguration config = ...

   config.sequencer("XML File Sequencer")
         .usingClass("org.modeshape.sequencer.xml.XmlSequencer")
         .loadedFromClasspath()
         .setDescription("Sequences XML files loaded under '/files',
   extracting the contents into the equivalent JCR graph structure.")
         .sequencingFrom("/files(//)*.xml[*]/jcr:content[@jcr:data]")
         .andOutputtingTo("/sequenced/xml/$1");
   ```



Note

Refer to ***`SOA-ROOT`*`/eds/modeshape/resources/modeshape-config-standard.xml`** for more information.

Chapter 8. ZIP File Sequencer

8.1. The ZIP File Sequencer

The ZIP file sequencer is included in ModeShape and extracts the files and folders contained in the ZIP archive file, extracting the files and folders into the repository using JCR's **`nt:file`** and **`nt:folder`** built-in node types. The structure of the output thus matches the logical structure of the contents of the ZIP file.

8.2. ZIP File Sequencer Example

This sequencer generates a graph structure that maps to the files and folders in the ZIP file. An example (listed in the JCR document view) from sequencing a ZIP file written into /a/foo and containing one file, /x/y/z.txt is provided below:

```
<foo jcr:primaryType="zip:file"
     jcr:mixinTypes="mode:derived"
     mode:derivedAt="2011-05-13T13:12:03.925Z"
     mode:derivedFrom="/files/docForReferenceGuide.xml" >
  <x jcr:primaryType="nt:folder"
     jcr:created="2011-05-12T20:07Z"
     jcr:createdBy="currentJcrUser">
    <y jcr:primaryType="nt:folder"
       jcr:created="2011-05-12T20:09Z"
       jcr:createdBy="currentJcrUser">
      <z.txt jcr:primaryType="nt:file">
        <jcr:content jcr:primaryType="nt:resource"
                     jcr:data="This is the file content"
                     jcr:lastModified="2011-05-12T20:12Z"
                     jcr:lastModifiedBy="currentJcrUser"
                     jcr:mimeType="text/plain" />
      </z.txt>
    </y>
  </x>
</foo>
```

8.3. ZIP File Sequencer Node Types

The CND for the zip:file node type is listed below.

```
[zip:file] > nt:folder, mix:mimeType
```

8.4. Configuring a ZIP File Sequencer

1. **Include the relevant libraries**

   Include **`modeshape-sequencer-zip-`*`VERSION`*`.jar`** in your application.

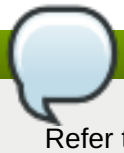
2. **Choose one of the following for sequencing configuration**

A. Define sequencing configuration based on standard example provided in ***SOA-ROOT*/eds/modeshape/resources/modeshape-config-standard.xml**:

```
<mode:sequencer jcr:name="ZIP File Sequencer"
mode:classname="org.modeshape.sequencer.zip.ZipSequencer">
  <mode:description>
    Sequences ZIP files loaded under '/files', extracting the
archive file contents into the equivalent JCR graph structure of
'nt:file' and 'nt:folder' nodes.
  </mode:description>
  <mode:pathExpression>
    eds-store:default:/files(//)(*.zip[*])/jcr:content[@jcr:data] =>
eds-store:default:/sequenced/zip/$1
  </mode:pathExpression>
</mode:sequencer>
```

B. Configure via **org.modeshape.jcr.JcrConfiguration**:

```
JcrConfiguration config = ...

config.sequencer("ZIP File Sequencer")
      .usingClass("org.modeshape.sequencer.zip.ZipSequencer")
      .loadedFromClasspath()
      .setDescription("Sequences ZIP files loaded under '/files',
extracting the archive file contents into the equivalent JCR graph
structure of 'nt:file' and 'nt:folder' nodes.")
      .sequencingFrom("/files(//)
(*.zip[*])/jcr:content[@jcr:data]")
      .andOutputtingTo("/sequenced/zip/$1");
```



Note

Refer to ***SOA-ROOT*/eds/modeshape/resources/modeshape-config-standard.xml** for more information.

Chapter 9. DDL File Sequencer

9.1. The DDL File Sequencer

The DDL file sequencer included in ModeShape is capable of parsing the more important DDL statements from SQL-92, Oracle, Derby, and PostgreSQL, and constructing a graph structure containing a structured representation of these statements. The resulting graph structure is largely the same for all dialects, though some dialects have non-standard additions to their grammar, and thus require dialect-specific additions to the graph structure.

The sequencer is designed to behave as intelligently as possible with as little configuration. Thus, the sequencer automatically determines the dialect used by a given DDL stream. This can be tricky, of course, since most dialects are very similar and the distinguishing features of a dialect may only be apparent in some of the statements.

To get around this, the sequencer uses a "best fit" algorithm: run the DDL stream through the parser for each of the dialects, and determine which parser was able to successfully read the greatest number of statements and tokens.

Note

It is possible to define which DDL dialects (or grammars) should be considered during sequencing using the "grammars" property in the sequencer configuration. Set the values of this property to the names of the grammars (e.g., "oracle", "postgres", "standard", or "derby"), specified in the order they should be used. To use a custom DDL parser not provided by ModeShape, provide the fully-qualified class name of the DdlParser implementation class.

One very interesting capability of this sequencer is that, although only a subset of the (more common) DDL statements are supported, the sequencer is still extremely functional since it does still add all statements into the output graph, just without much detail other than just the statement text and the position in the DDL file. Thus, if a DDL file contains statements the sequencer understands and statements the sequencer does not understand, the graph will still contain all statements, where those statements understood by the sequencer will have full detail. Since the underlying parsers are able to operate upon a single statement, it is possible to go back later (after the parsers have been enhanced to support additional DDL statements) and re-parse only those incomplete statements in the graph.

At this time, the sequencer supports SQL-92 standard DDL as well as dialects from Oracle, Derby, and PostgreSQL. It supports:

- Detailed parsing of CREATE SCHEMA, CREATE TABLE and ALTER TABLE.
- Partial parsing of DROP statements
- General parsing of remaining schema definition statements (i.e. CREATE VIEW, CREATE DOMAIN, etc.

Note that the sequencer does not perform detailed parsing of SQL (i.e. SELECT, INSERT, UPDATE, etc....) statements.

9.2. DDL File Sequencer Properties

For information about configurable properties relating to the DDL File Sequencer, refer to the **`org.modeshape.sequencer.ddl.DdlSequencer`** class in the Red Hat JBoss Data Virtualization JavaDoc.

9.3. DDL File Sequencer Example

Sequencing results in graph nodes basically representing the BNF structure of each DDL statement. Below is an example DDL schema definition statement containing table and view definition statements.

```
CREATE SCHEMA hollywood
   CREATE TABLE films (title varchar(255), release date, producerName
varchar(255))
   CREATE VIEW winners AS SELECT title, release FROM films WHERE
producerName IS NOT NULL;
```

The resulting graph structure contains the raw statement expression, pertinent table, column and key reference information and position of the statement in the text stream (e.g., line number, column number and character index) so the statement can be tied back to the original DDL:

```
<nt:unstructured jcr:name="statements"
                 jcr:mixinTypes = "mode:derived"
                 mode:derivedAt="2011-05-13T13:12:03.925Z"
                 mode:derivedFrom="/files/foo.sql"
                 ddl:parserId="POSTGRES">
  <nt:unstructured jcr:name="hollywood"
jcr:mixinTypes="ddl:createSchemaStatement"
                  ddl:startLineNumber="1"
                   ddl:startColumnNumber="1"
                   ddl:expression="CREATE SCHEMA hollywood"
                   ddl:startCharIndex="0">
    <nt:unstructured jcr:name="films"
jcr:mixinTypes="ddl:createTableStatement"
                   ddl:startLineNumber="2"
                   ddl:startColumnNumber="5"
                   ddl:expression="CREATE TABLE films (title varchar(255),
release date, producerName varchar(255))"
                   ddl:startCharIndex="28"/>
    <nt:unstructured jcr:name="title" jcr:mixinTypes="ddl:columnDefinition"
                   ddl:datatypeName="VARCHAR"
                   ddl:datatypeLength="255"/>
    <nt:unstructured jcr:name="release"
jcr:mixinTypes="ddl:columnDefinition"
                   ddl:datatypeName="DATE"/>
    <nt:unstructured jcr:name="producerName"
jcr:mixinTypes="ddl:columnDefinition"
                   ddl:datatypeName="VARCHAR"
                   ddl:datatypeLength="255"/>
  <nt:unstructured jcr:name="winners"
jcr:mixinTypes="ddl:createViewStatement"
                   ddl:startLineNumber="3"
                   ddl:startColumnNumber="5"
                   ddl:expression="CREATE VIEW winners AS SELECT title,
release FROM films WHERE producerName IS NOT NULL;"
                   ddl:queryExpression="SELECT title, release FROM films
WHERE producerName IS NOT NULL"
                   ddl:startCharIndex="113"/>
</nt:unstructured>
```

Note that all nodes are of type **`nt:unstructured`** while the type of statement is identified using mixins. Also, each of the nodes representing a statement contain: a **`ddl:expression`** property with the exact statement as it appeared in the original DDL stream; a **`ddl:startLineNumber`** and **`ddl:startColumnNumber`** property defining the position in the original DDL stream of the first character in the expression; and a **`ddl:startCharIndex`** property that defines the integral index of the first character in the expression as found in the DDL stream. All of these properties make sure the statement can be traced back to its location in the original DDL.

9.4. Configuring a DDL File Sequencer

1. **Include the relevant libraries**

   Include **`modeshape-sequencer-ddl-`*`VERSION`*`.jar`** in your application.

2. **Choose one of the following for sequencing configuration**

   A. Define sequencing configuration based on standard example provided in ***`SOA-ROOT`*`/eds/modeshape/resources/modeshape-config-standard.xml`**:

   ```
   <mode:sequencer jcr:name="DDL File Sequencer"
   mode:classname="org.modeshape.sequencer.ddl.DdlSequencer">
     <mode:description>
       Sequences DDL files loaded under '/files', extracting the
   structured abstract syntax tree of the DDL commands and expressions.
     </mode:description>
     <mode:pathExpression>
       eds-store:default:/files(//(*.ddl[*]))/jcr:content[@jcr:data] =>
   eds-store:default:/sequenced/ddl/$1
     </mode:pathExpression>
   </mode:sequencer>
   ```

   B. Configure via **`org.modeshape.jcr.JcrConfiguration`**:

   ```
   JcrConfiguration config = ...

   config.sequencer("DDL File Sequencer")
         .usingClass("org.modeshape.sequencer.ddl.DdlSequencer")
         .loadedFromClasspath()
         .setDescription("Sequences DDL files loaded under '/files',
   extracting the structured abstract syntax tree of the DDL commands
   and expressions.")
         .sequencingFrom("/files(//(*.ddl[*]))/jcr:content[@jcr:data]")
         .andOutputtingTo("/sequenced/ddl/$1");
   ```

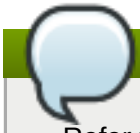
   This will use all of the built-in grammars (e.g., "standard", "oracle", "postgres", and "derby"). To specify a different order or subset of the grammars, use the **`setProperty(...)`** method. The following example uses the standard grammar followed by the PostgreSQL grammar:

   ```
   config.sequencer("DDL File Sequencer")
         .usingClass("org.modeshape.sequencer.ddl.DdlSequencer")
         .loadedFromClasspath()
         .setDescription("Sequences DDL files loaded under '/files',
   extracting the structured abstract syntax tree of the DDL commands
   ```

```
and expressions.")
      .setProperty("grammar","standard","postgres")
      .sequencingFrom("/files(//(*.ddl[*]))/jcr:content[@jcr:data]")
      .andOutputtingTo("/sequenced/ddl/$1");
```

To use a custom implementation of DdlParser , use the fully-qualified name of the implementation class (which must have a no-arg constructor) as the name of the grammar:

```
config.sequencer("DDL File Sequencer")
      .usingClass("org.modeshape.sequencer.ddl.DdlSequencer")
      .loadedFromClasspath()
      .setDescription("Sequences DDL files loaded under '/files',
extracting the structured abstract syntax tree of the DDL commands
and expressions.")

.setProperty("grammar","standard","postgres","org.example.ddl.MyCust
omDdlParser")
      .sequencingFrom("/files(//(*.ddl[*]))/jcr:content[@jcr:data]")
      .andOutputtingTo("/sequenced/ddl/$1");
```



Note

Refer to ***SOA-ROOT*/eds/modeshape/resources/modeshape-config-standard.xml** for more information.

Chapter 10. Text Sequencers

10.1. Text Sequencers

Text sequencers extract data from text streams. There are separate sequencers for character-delimited sequencing and fixed width sequencing, but both treat the incoming text stream as a series of rows (separated by line-terminators, as defined in **BufferedReader**.readLine() with each row consisting of one or more columns. As noted above, each text sequencer provides its own mechanism for splitting the row into columns.

10.2. Abstract Text Sequencer

When using the **AbstractTextSequencer**, the default row factory creates one node in the output location for each row sequenced from the source and adds each column with the row as a child node of the row node. The output graph takes the following form (all nodes have primary type **nt:unstructured**:

```
 <graph root jcr:mixinTypes = mode:derived,
                mode:derivedAt="2011-05-13T13:12:03.925Z",
                mode:derivedFrom="/files/foo.dat">
     + text:row[1]
     |   + text:column[1] (jcr:mixinTypes = text:column, text:data =
<column1 data>)
     |   + ...
     |   + text:column[n] (jcr:mixinTypes = text:column, text:data =
<columnN data>)
     + ...
     + text:row[m]
         + text:column[1] (jcr:mixinTypes = text:column, text:data =
<column1 data>)
         + ...
         + text:column[n] (jcr:mixinTypes = text:column, text:data =
<columnN data>)
```

10.3. Abstract Text Sequencer Properties

For information about configurable properties relating to the Abstract Text Sequencer, refer to the **org.modeshape.sequencer.text.AbstractTextSequencer** class in the Red Hat JBoss Data Virtualization JavaDoc.

10.4. Delimited Text Sequencer

The **DelimitedTextSequencer** splits rows into columns based on a regular expression pattern. Although the default pattern is a comma, any regular expression can be provided allowing for more sophisticated splitting patterns.

10.5. Delimited Text Sequencer Properties

For information about configurable properties relating to the Delimited Text Sequencer, refer to the **org.modeshape.sequencer.text.DelimitedTextSequencer** class in the Red Hat JBoss Data

Virtualization JavaDoc.

10.6. Configuring a Delimited Text Sequencer

To use this sequencer, include the **`modeshape-sequencer-text`** JAR in your application and configure the **`JcrConfiguration`** to use this sequencer using something similar to:

```
JcrConfiguration config = ...

config.sequencer("Delimited Text Sequencer")
      .usingClass("org.modeshape.sequencer.text.DelimitedTextSequencer")
      .loadedFromClasspath()
      .setDescription("Sequences delimited files to extract values")
      .sequencingFrom("//(*.(txt)[*])/jcr:content[@jcr:data]")
      .setProperty("splitPattern", "|")
      .andOutputtingTo("/txt/$1");
```

10.7. Fixed Width Text Sequencer

The **`FixedWidthTextSequencer`** splits rows into columns based on predefined positions. The default setting is to have a single column per row.

10.8. Fixed Width Text Sequencer Properties

For information about configurable properties relating to the Fixed Width Text Sequencer, refer to the **`org.modeshape.sequencer.text.FixedWidthTextSequencer`** class in the Red Hat JBoss Data Virtualization JavaDoc.

10.9. Configuring a Fixed Width Text Sequencer

1. **Include the relevant libraries**

   Include **`modeshape-sequencer-text-`*`VERSION`*`.jar`** in your application.

2. **Choose one of the following for sequencing configuration**

   A. Define sequencing configuration based on standard example provided in ***`SOA-ROOT`*`/eds/modeshape/resources/modeshape-config-standard.xml`**:

```
<mode:sequencer jcr:name="Fixed Width Text File Sequencer"
mode:classname="org.modeshape.sequencer.text.FixedWidthTextSequencer
">
  <mode:description>
    Sequences *.txt fixed-width text files loaded under '/files',
extracting splitting rows into columns based on predefined
positions.
  </mode:description>
  <mode:pathExpression>
    eds-store:default:/files//(*.txt[*])/jcr:content[@jcr:data] =>
```

```
eds-store:default:/sequenced/text/fixedWidth/$1
  </mode:pathExpression>
  <mode:columnStartPositions/>
</mode:sequencer>
```

Note

The **`columnStartPositions`** property defines the 0-based column start positions. Everything before the first start position is treated as the first column. The default value is the empty string (implying that each row should be treated as a single column). There is an implicit column start position of 0 that never needs to be specified.

B. Configure via **`org.modeshape.jcr.JcrConfiguration`**:

```
JcrConfiguration config = ...

config.sequencer("Fixed Width Text Sequencer")

.usingClass("org.modeshape.sequencer.text.FixedWidthTextSequencer")
      .loadedFromClasspath()
      .setDescription("Sequences *.txt fixed-width text files
loaded under '/files', extracting splitting rows into columns based
on predefined positions.")
      .sequencingFrom("/files//(*.txt[*])/jcr:content[@jcr:data]")
      .setProperty("columnStartPositions", "3,6,15")
      .andOutputtingTo("/sequenced/text/fixedWidth/$1");
```

Note

Refer to ***`SOA-ROOT`*`/eds/modeshape/resources/modeshape-config-standard.xml`** for more information.

Chapter 11. Red Hat JBoss Data Virtualization Relational Model Sequencer

11.1. Relational Model Sequencer

Teiid Designer is a visual tool that enables rapid, model-driven definition, integration, management and testing of Red Hat JBoss Data Virtualization without programming using the Red Hat JBoss Data Virtualization runtime engine. It is capable of modeling several different kinds of data structures, but the most common and widely-used are relational models that describe a relational database schema, including the catalogs/schemas, tables, views, columns, primary keys, foreign keys, indexes, procedures, procedure results, procedure results, and logical relationships. Teiid Designer can reverse-engineer a relational model from a JDBC relational database or DDL file. It can also define "virtual" models that are transformations of other models (where the transformations are defined in terms of SQL select, insert, update, and delete statements). These models can then be packaged into a virtual database, which can be deployed to a Red Hat JBoss Data Virtualization runtime engine.

Red Hat JBoss Data Virtualization is a high-performance database virtualization engine that allows JDBC and ODBC client applications access the virtual database as if it were a real database, using relational, XML, XQuery and procedural queries. Red Hat JBoss Data Virtualization dynamically (and in real-time) figures out how to answer the queries and operations issued by clients by efficiently accessing and manipulating the data inside the underlying data sources. The sophisticated engine is able to plan and optimize these operations, even when multiple heterogeneous relational and non-relational data sources must be accessed to obtain the required information.

The Red Hat JBoss Data Virtualization relational model sequencer parses the model files produced by the Teiid Designer, and extracts the structured relational data model described by the XMI file. This means that when these models are uploaded into a ModeShape repository, the sequencer writes to the repository all this relational metadata, where it can be queried and accessed by JCR , RESTful, and even JDBC clients.

11.2. Relational Model Sequencer Properties

For information about configurable properties relating to the Red Hat JBoss Data Virtualization Relational Model Sequencer, refer to the **`org.modeshape.sequencer.classfile.ClassFileSequencer`** class in the Red Hat JBoss Data Virtualization JavaDoc.

11.3. Relational Model Sequencer UUIDs

As mentioned above, the Red Hat JBoss Data Virtualization model sequencer can operate in two modes. The behavior you choose will dramatically change what you can do with the sequenced relational models.

The first mode reuses the "**`xmi:uuid`**" identifiers on each object in the model as the "**`jcr:uuid`**" node identifiers. In this mode, the sequencer represents each model reference as a JCR WEAKREFERENCE, making it very easy to navigate and query relationships. However, there is one major disadvantage of this approach: each time a model is uploaded into the repository, the sequencer will override any output generated by earlier sequencing operations upon that file (or other versions of it). Thus, the sequenced representation of an uploaded model can ever appear only once within the repository, even though different versions of that model might exist in the repository at different locations. This may be desirable in some situations, but for most situations it is not acceptable.

In the second mode of operation (which is the default mode), there is no correlation between the model's "**`xmi:uuid`**" and "**`jcr:uuid`**" node identifiers. Various versions of a given model can be uploaded into the repository at multiple locations, yet each model's relational schema will exist in the repository. The downside of this approach is that references are no longer simply WEAKREFERENCE properties. Instead, each single-

valued reference will be represented as a series of four properties:

- **`{referenceName}Href`** - stores the href literal value from the XMI file; this is always set
- **`{referenceName}XmiUuid`** - stores the XMI UUID to the referenced node; this is set only if the href had an embedded UUID (hrefs to data types and XSD components don't use UUIDs)
- **`{referenceName}Name`** - stores the name of the resolved node, though this may not be set if the object being referenced is in another model
- **`{referenceName}`** - stores the JCR weak reference to the resolve node, though this may not be set if the object being referenced is in another model

where "{referenceName}" is the name of the model reference. Multi-value references are also represented as a series of four properties, but with a slightly different naming pattern:

- **`{singularReferenceName}Href`** - stores the href literal values from the XMI file; this is always set
- **`{singularReferenceName}XmiUuid`** - stores the XMI UUID to the referenced nodes; this is set only if the hrefs have an embedded UUID (hrefs to data types and XSD components don't use UUIDs)
- **`{singularReferenceName}Name`** - stores the name of the resolved nodes, though this may not be set if the object being referenced is in another model
- **`{pluralReferenceName}`** - stores the JCR weak reference to the resolve nodes, though this may not be set if the object being referenced is in another model

Here, "{singularReferenceName}" is the singular form of the model reference name, and "{pluralReferenceName}" is the plural form of the model reference name. For example, for a reference named "columns", the "{singularReferenceName}" value would be "column" and the plural form is "columns". If the reference name is "properties", the singular form is "property" and the plural form is "properties". (ModeShape uses a novel algorithm to determine the singular and plural forms of many English words.)

References to model objects within the same model are easily resolved upon sequencing, and so we set all of the properties (regardless of the mode). However, references to objects in other models cannot be resolved at sequencing time.

Note

The Red Hat JBoss Data Virtualization VDB sequencer behavior is unrelated to this mode, since it always sequences models with new "**`jcr:uuid`**" identifiers that are unrelated to the "**`xmi:uuid`**" values. In this manner, each sequencing of a VDB will produce the relational model representation for each model in the VDB (with all valid references resolved between all models), independent of any generated output from the Red Hat JBoss Data Virtualization model sequencer.

11.4. Relational Model Sequencer Node Types

The model sequencer follows JCR best-practices by defining all nodes to have a primary type of "**`nt:unstructured`**" (or a node type that extends '**`nt:unstructured`**"), meaning it is possible and valid for any node to have any property (with single or multiple values). However, it is still useful to capture the metadata about what that node represents, and so the sequencer use mixins for this. For example, there is a "**`xmi:referenceable`**" mixin with a single "**`xmi:uuid`**" property (patterned after the built-in "**`mix:referenceable`**" mixin). Since all model objects have mmuuids, all nodes produced by this sequencer will have this mixin.

The rest of this section covers the various (and many!) node types defined for and used by this sequencer. Note that these are non-normative definitions of the node types; see the CND files in the "**`modeshape-sequencer-teiid`**" JAR file (or source) for the official definitions.

11.5. Compact Node Definitions for the xmi Namespace

The compact node definitions for the "xmi" namespace are as follows:

```
<nt  = "http://www.jcp.org/jcr/nt/1.0">
<xmi = "http://www.omg.org/XMI">

//------------------------------------------------------------------------
--------
// N O D E T Y P E S
//------------------------------------------------------------------------
--------

[xmi:referenceable] mixin
 - xmi:uuid (string) mandatory

[xmi:model] > nt:unstructured, xmi:referenceable orderable
 - xmi:version (double) = '2.0'
```

11.6. Compact Node Definitions for the mmcore Namespace

The compact node definitions for the mmcore namespace are as follows:

```
<nt  = "http://www.jcp.org/jcr/nt/1.0">
<xmi = "http://www.omg.org/XMI">
<mmcore = "http://www.metamatrix.com/metamodels/Core">
<mode = "http://www.modeshape.org/1.0">

[mmcore:model] > xmi:referenceable, mode:hashed mixin
  - mmcore:modelType (string) = 'UNKNOWN' <
'PHYSICAL','VIRTUAL','TYPE','VDB_ARCHIVE',

'UNKNOWN','FUNCTION','CONFIGURATION','METAMODEL',

'EXTENSION','LOGICAL','MATERIALIZATION'
  - mmcore:primaryMetamodelUri (string)
  - mmcore:description (string)
  - mmcore:nameInSource (string)
  - mmcore:maxSetSize (long) = '100'
  - mmcore:visible (boolean) = 'true'
  - mmcore:supportsDistinct (boolean) = 'true'
  - mmcore:supportsJoin (boolean) = 'true'
  - mmcore:supportsOrderBy (boolean) = 'true'
  - mmcore:supportsOuterJoin (boolean) = 'true'
  - mmcore:supportsWhereAll (boolean) = 'true'
  - mmcore:supportsDistinct (boolean) = 'true'
  - mmcore:producerName (string)
  - mmcore:producerVersion (string)
  - mmcore:originalFile (string)
```

```
  - mmcore:sha1 (string)

[mmcore:import] > nt:unstructured, xmi:referenceable orderable
  - mmcore:modelType (string) = 'UNKNOWN' <
'PHYSICAL','VIRTUAL','TYPE','VDB_ARCHIVE',

'UNKNOWN','FUNCTION','CONFIGURATION','METAMODEL',

'EXTENSION','LOGICAL','MATERIALIZATION'
  - mmcore:primaryMetamodelUri (string)
  - mmcore:path (string)
  - mmcore:name (string)
  - mmcore:modelLocation (string)

[mmcore:annotated] mixin
  - mmcore:description (string)
  - mmcore:keywords (string) multiple

[mmcore:tags] mixin
  - * (undefined) multiple
  - * (undefined)
```

11.7. Compact Node Definitions for the relational Namespace

The compact node definitions for the "relational" namespace are as follows:

```
<nt  = "http://www.jcp.org/jcr/nt/1.0">
<relational='http://www.metamatrix.com/metamodels/Relational'>
<xmi = "http://www.omg.org/XMI">

//------------------------------------------------------------------------
--------
// N O D E T Y P E S
//------------------------------------------------------------------------
--------

[relational:relationalEntity] > xmi:referenceable abstract mixin
 - relational:nameInSource (string)

[relational:relationship] > nt:unstructured, relational:relationalEntity
abstract

// ----------------------------------------
// Columns and Column Sets
// ----------------------------------------

[relational:column] > nt:unstructured, relational:relationalEntity
 - relational:nativeType (string)
 - relational:type (weakreference)
 - relational:typeHref (string)
 - relational:typeXmiUuid (string)
 - relational:typeName (string)
 - relational:length (long)
 - relational:fixedLength (boolean)
 - relational:precision (long)
```

```
 - relational:scale (long)
 - relational:nullable (string) = 'NULLABLE' < 'NO_NULLS', 'NULLABLE',
'NULLABLE_UNKNOWN'
 - relational:autoIncremented (boolean) = 'false'
 - relational:defaultValue (string)
 - relational:minimumValue (string)
 - relational:maximumValue (string)
 - relational:format (string)
 - relational:characterSetName (string)
 - relational:collationName (string)
 - relational:selectable (boolean) = 'true'
 - relational:updateable (boolean) = 'true'
 - relational:caseSensitive (boolean) = 'true'
 - relational:searchability (string) = 'SEARCHABLE' < 'SEARCHABLE',
                                      'ALL_EXCEPT_LIKE', 'LIKE_ONLY',
'UNSEARCHABLE'
 - relational:currency (boolean) = 'false'
 - relational:radix (long) = '10'
 - relational:signed (boolean) = 'true'
 - relational:distinctValueCount (long) = '-1'
 - relational:nullValueCount (long) = '-1'
 - relational:uniqueKeys (weakreference) multiple
 - relational:uniqueKeyHrefs (string) multiple
 - relational:uniqueKeyXmiUuids (string) multiple
 - relational:uniqueKeyNames (string) multiple
 - relational:indexes (weakreference) multiple
 - relational:indexHrefs (string) multiple
 - relational:indexXmiUuids (string) multiple
 - relational:indexNames (string) multiple
 - relational:foreignKeys (weakreference) multiple
 - relational:foreignKeyHrefs (string) multiple
 - relational:foreignKeyXmiUuids (string) multiple
 - relational:foreignKeyNames (string) multiple
 - relational:accessPatterns (weakreference) multiple
 - relational:accessPatternHrefs (string) multiple
 - relational:accessPatternXmiUuids (string) multiple
 - relational:accessPatternNames (string) multiple

[relational:columnSet] > nt:unstructured, relational:relationalEntity
abstract orderable
 + * (relational:column) = relational:column copy

// -----------------------------------------
// Constraints
// -----------------------------------------

[relational:uniqueKey] > nt:unstructured, relational:relationalEntity
abstract
 - relational:columns (weakreference) multiple
 - relational:columnXmiUuids (string) multiple
 - relational:columnNames (string) multiple
 - relational:foreignKeys (weakreference) multiple
 - relational:foreignKeyHrefs (string) multiple
 - relational:foreignKeyXmiUuids (string) multiple
 - relational:foreignKeyNames (string) multiple
```

```
[relational:uniqueConstraint] > relational:uniqueKey

[relational:primaryKey] > relational:uniqueKey

[relational:foreignKey] > relational:relationship
 - relational:foreignKeyMultiplicity (string) = 'ZERO_TO_MANY' < 'ONE',
'MANY',
                                                'ZERO_TO_ONE',
'ZERO_TO_MANY', 'UNSPECIFIED'
 - relational:primaryKeyMultiplicity (string) = 'ONE' < 'ONE', 'MANY',
'ZERO_TO_ONE',
                                                'ZERO_TO_MANY',
'UNSPECIFIED'
 - relational:columns (weakreference) multiple
 - relational:columnXmiUuids (string) multiple
 - relational:columnNames (string) multiple
 - relational:uniqueKeys (weakreference) multiple
 - relational:uniqueKeyHrefs (string) multiple
 - relational:uniqueKeyXmiUuids (string) multiple
 - relational:uniqueKeyNames (string) multiple

[relational:index] > nt:unstructured, relational:relationalEntity
 - relational:filterCondition (string)
 - relational:nullable (boolean) = 'true'
 - relational:autoUpdate (boolean)
 - relational:unique (boolean)
 - relational:columns (weakreference) multiple
 - relational:columnXmiUuids (string) multiple
 - relational:columnNames (string) multiple

[relational:accessPattern] > nt:unstructured, relational:relationalEntity
orderable
 - relational:columns (UNDEFINED) multiple

// -----------------------------------------
// Tables and Views
// -----------------------------------------

[relational:table] > relational:columnSet abstract orderable
 - relational:system (boolean) = 'false'
 - relational:cardinality (long)
 - relational:supportsUpdate (boolean) = 'true'
 - relational:materialized (boolean) = 'false'
 - relational:logicalRelationships (weakreference) multiple
 - relational:logicalRelationshipHrefs (string) multiple
 - relational:logicalRelationshipXmiUuids (string) multiple
 - relational:logicalRelationshipNames (string) multiple
 + * (relational:primaryKey) = relational:primaryKey copy
 + * (relational:foreignKey) = relational:foreignKey copy
 + * (relational:accessPattern) = relational:accessPattern copy sns

[relational:baseTable] > relational:table orderable

[relational:view] > relational:table orderable
```

```
// ----------------------------------------
// Procedures
// ----------------------------------------

[relational:procedureParameter] > nt:unstructured,
relational:relationalEntity
 - relational:direction (string) < 'IN', 'OUT', 'INOUT', 'RETURN', 'UNKNOWN'
 - relational:defaultValue (string)
 - relational:nativeType (string)
 - relational:type (weakreference)
 - relational:typeXmiUuid (string)
 - relational:typeName (string)
 - relational:length (long)
 - relational:precision (long)
 - relational:scale (long)
 - relational:nullable (string) = 'NULLABLE' < 'NO_NULLS', 'NULLABLE',
'NULLABLE_UNKNOWN'
 - relational:radix (long) = '10'

[relational:procedureResult] > relational:columnSet orderable

[relational:procedure] > nt:unstructured, relational:relationalEntity
orderable
 - relational:function (boolean)
 - relational:updateCount (string) < 'AUTO', 'ZERO', 'ONE', 'MULTIPLE'
 + * (relational:procedureParameter) = relational:procedureParameter copy sns
 + * (relational:procedureResult) = relational:procedureResult copy


// ----------------------------------------
// Logical Relationships
// ----------------------------------------

[relational:logicalRelationshipEnd] > nt:unstructured,
relational:relationalEntity
 - relational:multiplicity (string) < 'ONE', 'MANY', 'ZERO_TO_ONE',
'ZERO_TO_MANY', 'UNSPECIFIED'
 - relational:table (weakreference)
 - relational:tableHref (string)
 - relational:tableXmiUuid (string)
 - relational:tableName (string)

[relational:logicalRelationship] > relational:relationship orderable
 + * (relational:logicalRelationshipEnd) = relational:logicalRelationshipEnd
copy sns

// ----------------------------------------
// Catalogs and Schemas
// ----------------------------------------

[relational:schema] > nt:unstructured, relational:relationalEntity orderable
 + * (relational:table) = relational:baseTable copy
 + * (relational:procedure) = relational:procedure copy sns
 + * (relational:index) = relational:index copy
 + * (relational:logicalRelationship) = relational:logicalRelationship copy
```

```
[relational:catalog] > nt:unstructured, relational:relationalEntity
orderable
 + * (relational:schema) = relational:schema copy
 + * (relational:table) = relational:baseTable copy
 + * (relational:procedure) = relational:procedure copy sns
 + * (relational:index) = relational:index copy
 + * (relational:logicalRelationship) = relational:logicalRelationship copy
```

11.8. Compact Node Definitions for the jdbcs Namespace

The compact node definitions for the "jdbcs" namespace are as follows:

```
<nt  = "http://www.jcp.org/jcr/nt/1.0">
<xmi = "http://www.omg.org/XMI">
<jdbcs = "http://www.metamatrix.com/metamodels/JDBC">

//------------------------------------------------------------------------
--------
// N O D E T Y P E S
//------------------------------------------------------------------------
--------

[jdbcs:source] > nt:unstructured, xmi:referenceable
 - jdbcs:name (string)
 - jdbcs:driverName (string)
 - jdbcs:driverClass (string)
 - jdbcs:username (string)
 - jdbcs:url (string)

[jdbcs:imported] > nt:unstructured, xmi:referenceable
 - jdbcs:createCatalogsInModel (boolean) = 'true'
 - jdbcs:createSchemasInModel (boolean) = 'true'
 - jdbcs:convertCaseInModel (string) < 'NONE', 'TO_UPPERCASE',
'TO_LOWERCASE'
 - jdbcs:generateSourceNamesInModel (string) = 'UNQUALIFIED' < 'NONE',
'UNQUALIFIED', 'FULLY_QUALIFIED'
 - jdbcs:includedCatalogPaths (string) multiple
 - jdbcs:includedSchemaPaths (string) multiple
 - jdbcs:excludedObjectPaths (string) multiple
 - jdbcs:includeForeignKeys (boolean) = 'true'
 - jdbcs:includeIndexes (boolean) = 'true'
 - jdbcs:includeProcedures (boolean) = 'false'
 - jdbcs:includeApproximateIndexes (boolean) = 'true'
 - jdbcs:includeUniqueIndexes (boolean) = 'false'
 - jdbcs:includedTableTypes (string) multiple
```

11.9. Compact Node Definitions for the transform Namespace

The compact node definitions for the "transform" namespace are as follows:

```
<transform='http://www.metamatrix.com/metamodels/Transformation'>

//------------------------------------------------------------------------
```

```
--------
// N O D E T Y P E S
//---------------------------------------------------------------------
--------

[transform:transformed] mixin
 - transform:transformedFrom (weakreference)
 - transform:transformedFromHrefs (string)
 - transform:transformedFromXmiUuids (string)
 - transform:transformedFromNames (string)

[transform:withSql] mixin
 - transform:selectSql (string)
 - transform:insertSql (string)
 - transform:updateSql (string)
 - transform:deleteSql (string)
 - transform:insertAllowed (boolean) = 'true'
 - transform:updateAllowed (boolean) = 'true'
 - transform:deleteAllowed (boolean) = 'true'
 - transform:outputLocked (boolean) = 'false'
 - transform:insertSqlDefault (boolean) = 'true'
 - transform:updateSqlDefault (boolean) = 'true'
 - transform:deleteSqlDefault (boolean) = 'true'
```

11.10. Default Values

Teiid Designer does not persist default values in the XMI files. The sequencer knows these default values, and includes them in the sequenced output so that they can be accessed and queried.

11.11. Annotations

Rather than creating a separate "Annotation" object like what exist in the XMI models, the annotation's description and keywords are simply recorded as a "**`mmcore:description`**" and "**`mmcore:keywords`**" properties on the node created for the target of the annotation. This is really nice, because if a description is placed on a relational column object in a model, then that description appears as a property directly on the corresponding "**`relational:column`**" node. Note that when any annotation properties are placed on a node, the "**`mmcore:annotated`**" mixin is added to that node.

11.12. Tags

Tags are also stored on "Annotation" objects, and each tag consist of a key-value pair. The sequencer does two things depending upon what the key looks like. When the key is a simple string without a ':', then a property is created on the annotation's target object using this string as the property name and the tag's value as the property's value. More recently Teiid Designer has started to use tags with keys of the form "**`namespace:name`**", where "**`namespace`**" is really informal and can theoretically be any string value. While this format is the same as JCR property names, treating them as namespaced JCR property names would require there be a namespace URI registered with the prefix matching the "namespace" value.

The sequencer tries to parse the tag key as a property name, and if it works then the tag is added as a property just as mentioned earlier. However, if the namespace does not exist, then the sequencer splits the key into the two parts, where the first is used to identify a child node and the second is used as a property name.

For example, a tag on the "ID" column object under the "MyTable" base table:

```
foo="bar"
```

will be stored as a property "**foo**" with value "**bar**" on the "**MyTable/ID**" node. However, the

```
connection:driver-class="oracle.jdbc.OracleDriver"
```

tag on the same object would be stored as the "**driver-class**" property (with value "**oracle.jdbc.OracleDriver**") on the "**MyTable/ID/connection**" object.

11.13. Transformation

The transformation information, like with annotations, is projected onto the nodes representing the model objects that are the "output" of the transformation, where the objects that are "inputs" to the transformation are recorded as a (potentially multi-valued) property on the "output" object, and the "**transform:transformed**" mixin is added to the output node. In other words, virtual base tables, columns, procedures, etc., are marked as "transform:transformed" and have an "input" property pointing to the node(s) that are the inputs for the transformation. The SQL statements, supports flags, and defaults flags are also added as properties on the output virtual base table and procedures, and the "**transform:withSql**" mixin that defines these properties is added to that output node.

11.14. Relational Model Sequencer Example

Here is a representation of the nodes output by the sequencing of an example virtual relational model:

```
PartsVirtual jcr:primaryType="xmi:model"
   - jcr:mixinTypes=["mmcore:model","mix:referenceable","xmi:referenceable",
"mode:derived"]
   - mode:derivedAt="2011-05-13T13:12:03.925Z"
   - mode:derivedFrom="/files/foo.xmi"
   - jcr:uuid="d1a1b82f-055b-4db2-a3e7-a9668f3a70b6"
   - mmcore:maxSetSize="100"
   - mmcore:modelType="VIRTUAL"
   - mmcore:originalFile="/model/parts/PartsVirtual.xmi"
   -
mmcore:primaryMetamodelUri="http://www.metamatrix.com/metamodels/Relational"
   - mmcore:producerName="Teiid Designer"
   - mmcore:producerVersion="6.0"
   - mode:sha1="84a77940f9140a358861d12d4bbb4160afadc08c"
   - mmcore:supportsDistinct="true"
   - mmcore:supportsJoin="true"
   - mmcore:supportsOrderBy="true"
   - mmcore:supportsOuterJoin="true"
   - mmcore:supportsWhereAll="true"
   - xmi:uuid="fb52cb80-128a-1eec-8518-c32201e76066"
   - xmi:version="2.0"
   - mmcore:visible="true"
   PartSupplier_SourceB jcr:primaryType="mmcore:import"
     - jcr:mixinTypes=["mix:referenceable","xmi:referenceable"]
     - jcr:uuid="c3a98bf2-7dbf-4c46-8baa-bf32e389cddd"
     - mmcore:modelType="PHYSICAL"
     -
```

```
mmcore:primaryMetamodelUri="http://www.metamatrix.com/metamodels/Relational"
     - xmi:uuid="mmuuid:980de782-b1e5-1f55-853c-ed5dfdd1bb78"
   PartsSupplier_SourceA jcr:primaryType="mmcore:import"
     - jcr:mixinTypes=["mix:referenceable","xmi:referenceable"]
     - jcr:uuid="55385418-01c9-4d5c-9f79-91b8e10c6946"
     - mmcore:modelType="PHYSICAL"
     -
mmcore:primaryMetamodelUri="http://www.metamatrix.com/metamodels/Relational"
     - xmi:uuid="mmuuid:980de784-b1e5-1f55-853c-ed5dfdd1bb78"
   XMLSchema jcr:primaryType="mmcore:import"
     - jcr:mixinTypes=["mix:referenceable","xmi:referenceable"]
     - jcr:uuid="8b5c2268-0770-405b-a4d8-12a868cc27a4"
     - mmcore:modelType="PHYSICAL"
     - mmcore:primaryMetamodelUri="http://www.eclipse.org/xsd/2002/XSD"
     - xmi:uuid="mmuuid:a6591280-bf1d-1f2c-9911-b53abd16b14e"
   SupplierInfo jcr:primaryType="relational:baseTable"
     - jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
     - jcr:uuid="37bf368e-0618-4f2f-b4c2-2ab4c0729502"
     - transform:deleteAllowed="true"
     - transform:deleteSqlDefault="true"
     - transform:inputHrefs="PartsSupplier_SourceA.xmi#mmuuid/bc400080-1284-
1eec-8518-c32201e76066"
     - transform:inputXmiUuids="bc400080-1284-1eec-8518-c32201e76066"
     - transform:insertAllowed="true"
     - transform:insertSqlDefault="true"
     - relational:materialized="false"
     - transform:selectSql="SELECT
PartSupplier_Oracle.SUPPLIER_PARTS.SUPPLIER_ID,
PartSupplier_Oracle.SUPPLIER_PARTS.PART_ID,
PartSupplier_Oracle.SUPPLIER_PARTS.QUANTITY,
PartSupplier_Oracle.SUPPLIER_PARTS.SHIPPER_ID,
PartsSupplier_SQLServer.SUPPLIER.SUPPLIER_NAME,
PartsSupplier_SQLServer.SUPPLIER.SUPPLIER_STATUS,
PartsSupplier_SQLServer.SUPPLIER.SUPPLIER_CITY,
PartsSupplier_SQLServer.SUPPLIER.SUPPLIER_STATE FROM
PartSupplier_Oracle.SUPPLIER_PARTS, PartsSupplier_SQLServer.SUPPLIER WHERE
PartSupplier_Oracle.SUPPLIER_PARTS.SUPPLIER_ID =
PartsSupplier_SQLServer.SUPPLIER.SUPPLIER_ID"
     - relational:supportsUpdate="true"
     - relational:system="false"
     - transform:updateAllowed="true"
     - transform:updateSqlDefault="true"
     - xmi:uuid="2473dbc0-128c-1eec-8518-c32201e76066"
     SUPPLIER_ID jcr:primaryType="relational:column"
       - jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
      - jcr:uuid="5f62a519-7948-4c9d-95df-131b489cec8e"
       - relational:autoIncremented="false"
       - relational:caseSensitive="true"
       - relational:currency="false"
       - relational:distinctValueCount="-1"
       - transform:inputHrefs="PartSupplier_SourceB.xmi#mmuuid/55e12d01-
1275-1eec-8518-c32201e76066"
       - transform:inputXmiUuids="55e12d01-1275-1eec-8518-c32201e76066"
       - relational:length="10"
```

```
        - relational:nativeType="VARCHAR2"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="143ff680-1291-1eec-8518-c32201e76066"
     PART_ID jcr:primaryType="relational:column"
        - jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
       - jcr:uuid="bcce191f-acfd-48b9-8be8-ea04c0d37283"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:fixedLength="true"
        - transform:inputHrefs="PartSupplier_SourceB.xmi#mmuuid/54ed0902-
1275-1eec-8518-c32201e76066"
        - transform:inputXmiUuids="54ed0902-1275-1eec-8518-c32201e76066"
        - relational:length="4"
        - relational:nativeType="CHAR"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="1d9b97c0-1291-1eec-8518-c32201e76066"
     QUANTITY jcr:primaryType="relational:column"
        - jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
       - jcr:uuid="126d6138-ce5e-40e3-92d9-48a239453dbb"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:fixedLength="true"
        - transform:inputHrefs="PartSupplier_SourceB.xmi#mmuuid/55e12d02-
1275-1eec-8518-c32201e76066"
        - transform:inputXmiUuids="55e12d02-1275-1eec-8518-c32201e76066"
        - relational:nativeType="NUMBER"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:precision="3"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
```

```
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#short"
        - relational:typeName="short"
        - relational:typeXmiUuid="5bbcf140-b9ae-1e21-b812-969c8fc8b016"
        - relational:updateable="true"
        - xmi:uuid="250ef100-1291-1eec-8518-c32201e76066"
      SHIPPER_ID jcr:primaryType="relational:column"
        - jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
       - jcr:uuid="d9856363-6950-40ea-9c9a-44c4af43ec38"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:fixedLength="true"
        - transform:inputHrefs="PartSupplier_SourceB.xmi#mmuuid/54ed0903-
1275-1eec-8518-c32201e76066"
        - transform:inputXmiUuids="54ed0903-1275-1eec-8518-c32201e76066"
        - relational:nativeType="NUMBER"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:precision="2"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#short"
        - relational:typeName="short"
        - relational:typeXmiUuid="5bbcf140-b9ae-1e21-b812-969c8fc8b016"
        - relational:updateable="true"
        - xmi:uuid="2b8e2640-1291-1eec-8518-c32201e76066"
      SUPPLIER_NAME jcr:primaryType="relational:column"
        - jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
       - jcr:uuid="d0b9d5cc-f95a-4e97-a3f9-59571f58e206"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="PartsSupplier_SourceA.xmi#mmuuid/bc400084-
1284-1eec-8518-c32201e76066"
        - transform:inputXmiUuids="bc400084-1284-1eec-8518-c32201e76066"
        - relational:length="30"
        - relational:nativeType="varchar"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="34da8540-1291-1eec-8518-c32201e76066"
      SUPPLIER_STATUS jcr:primaryType="relational:column"
```

```
       - jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
      - jcr:uuid="06253965-9f6f-4d6e-8219-2eb70a2745ed"
       - relational:autoIncremented="false"
       - relational:caseSensitive="true"
       - relational:currency="false"
       - relational:distinctValueCount="-1"
       - relational:fixedLength="true"
       - transform:inputHrefs="PartsSupplier_SourceA.xmi#mmuuid/bc400083-
1284-1eec-8518-c32201e76066"
       - transform:inputXmiUuids="bc400083-1284-1eec-8518-c32201e76066"
       - relational:nativeType="numeric"
       - relational:nullValueCount="-1"
       - relational:nullable="NULLABLE"
       - relational:precision="2"
       - relational:radix="10"
       - relational:searchability="SEARCHABLE"
       - relational:selectable="true"
       - relational:signed="true"
       - relational:typeHref="http://www.w3.org/2001/XMLSchema#short"
       - relational:typeName="short"
       - relational:typeXmiUuid="5bbcf140-b9ae-1e21-b812-969c8fc8b016"
       - relational:updateable="true"
       - xmi:uuid="3c4dde80-1291-1eec-8518-c32201e76066"
     SUPPLIER_CITY jcr:primaryType="relational:column"
       - jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
      - jcr:uuid="a9cfd1fd-1a99-4b7d-83dc-3dbeb86c7f0a"
       - relational:autoIncremented="false"
       - relational:caseSensitive="true"
       - relational:currency="false"
       - relational:distinctValueCount="-1"
       - transform:inputHrefs="PartsSupplier_SourceA.xmi#mmuuid/bc400081-
1284-1eec-8518-c32201e76066"
       - transform:inputXmiUuids="bc400081-1284-1eec-8518-c32201e76066"
       - relational:length="30"
       - relational:nativeType="varchar"
       - relational:nullValueCount="-1"
       - relational:nullable="NULLABLE"
       - relational:radix="10"
       - relational:searchability="SEARCHABLE"
       - relational:selectable="true"
       - relational:signed="true"
       - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
       - relational:typeName="string"
       - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
       - relational:updateable="true"
       - xmi:uuid="43c137c0-1291-1eec-8518-c32201e76066"
     SUPPLIER_STATE jcr:primaryType="relational:column"
       - jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
      - jcr:uuid="8e040c5d-acf8-407f-a090-4bc1feac45cc"
       - relational:autoIncremented="false"
       - relational:caseSensitive="true"
       - relational:currency="false"
       - relational:distinctValueCount="-1"
```

```
        - transform:inputHrefs="PartsSupplier_SourceA.xmi#mmuuid/bc400082-
1284-1eec-8518-c32201e76066"
        - transform:inputXmiUuids="bc400082-1284-1eec-8518-c32201e76066"
        - relational:length="2"
        - relational:nativeType="varchar"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="4a4faf40-1291-1eec-8518-c32201e76066"
```

11.15. Configuring a Red Hat JBoss Data Virtualization Relational Model Sequencer

1. **Include the relevant libraries**

   Include **`modeshape-sequencer-teiid-`*`VERSION`*`.jar`** in your application.

2. **Choose one of the following for sequencing configuration**

   A. Define sequencing configuration based on standard example provided in ***`SOA-ROOT`*`/eds/modeshape/resources/modeshape-config-standard.xml`**:

   ```
   <mode:sequencer jcr:name="Teiid Model Sequencer"
   mode:classname="org.modeshape.sequencer.teiid.ModelSequencer">
     <mode:description>
       Sequences Teiid relational models (e.g., *.xmi) loaded under
   '/files', extracting the structure defined in the models.
     </mode:description>
     <mode:pathExpression>
       eds-store:default:/files(//)(*.xmi[*])/jcr:content[@jcr:data] =>
   eds-store:default:/sequenced/teiid/models$1
     </mode:pathExpression>
   </mode:sequencer>
   ```

   B. Configure via **`org.modeshape.jcr.JcrConfiguration`**:

   ```
   JcrConfiguration config = ...

   config.sequencer("Teiid Model Sequencer")
         .usingClass("org.modeshape.sequencer.teiid.ModelSequencer")
         .loadedFromClasspath()
         .setDescription("Sequences Teiid relational models (e.g.,
   *.xmi) loaded under '/files', extracting the structure defined in
   the models.")
         .sequencingFrom("/files(//)
   (*.xmi[*])/jcr:content[@jcr:data]")
         .andOutputtingTo("/sequenced/teiid/models$1");
   ```

Note

Refer to ***SOA-ROOT*/eds/modeshape/resources/modeshape-config-standard.xml** for more information.

Chapter 12. Red Hat JBoss Data Virtualization VDB Sequencer

12.1. VDB Sequencer

Teiid Designer is a visual tool that enables rapid, model-driven definition, integration, management and testing of Red Hat JBoss Data Virtualization without programming using the Red Hat JBoss Data Virtualization runtime engine. It is capable of modeling several different kinds of data structures, but the most common and widely-used are relational models that describe a relational database schema, including the catalogs/schemas, tables, views, columns, primary keys, foreign keys, indexes, procedures, procedure results, procedure results, and logical relationships. Teiid Designer can reverse-engineer a relational model from a JDBC relational database or DDL file. It can also define "virtual" models that are transformations of other models (where the transformations are defined in terms of SQL select, insert, update, and delete statements). These models can then be packaged into a virtual database, which can be deployed to a Red Hat JBoss Data Virtualization runtime engine.

Red Hat JBoss Data Virtualization is a high-performance database virtualization engine that allows JDBC and ODBC client applications access the virtual database as if it were a real database, using relational, XML, XQuery and procedural queries. Red Hat JBoss Data Virtualization dynamically (and in real-time) figures out how to answer the queries and operations issued by clients by efficiently accessing and manipulating the data inside the underlying data sources. The sophisticated engine is able to plan and optimize these operations, even when multiple heterogeneous relational and non-relational data sources must be accessed to obtain the required information.

The Red Hat JBoss Data Virtualization VDB sequencer parses the VDB archive files produced by the Teiid Designer, and extracts the structured relational data model described by each of the contained XMI files. This means that when VDB files are uploaded into a ModeShape repository, the sequencer writes to the repository all this virtual database and relational metadata contained in the VDB, where it can be queried and accessed by JCR , RESTful, and even JDBC clients.

The **`VdbSequencer`** has no properties for changing behavior.

12.2. VDB Sequencer UUIDs and References

A Red Hat JBoss Data Virtualization virtual database file is entirely self-contained: it contains all of the models required for the VDB. No model can contain references to objects outside of these models, so the entire VDB archive is consistent and complete. When the sequencer extracts the relational information from these models, it automatically resolves all references. Also, the resulting content is independent of any the content from all other previous sequencing operations, including that of the Red Hat JBoss Data Virtualization Model Sequencer.

12.3. VDB Sequencer Node Types

The VDB sequencer follows JCR best-practices by defining all nodes to have a primary type of "**`nt:unstructured`**" (or a node type that extends '**`nt:unstructured`**"), meaning it is possible and valid for any node to have any property (with single or multiple values). However, it is still useful to capture the metadata about what that node represents, and so the sequencer use mixins for this.

The VDB sequencer reuses all of the node types from the Red Hat JBoss Data Virtualization Model Sequencer, plus several new node types that are used for the VDB-specific metadata, as described below. Note that these are non-normative definitions of the node types; see the CND files in the "**`modeshape-sequencer-teiid`**" JAR file (or source) for the official definitions.

12.4. Content Node Definitions for the vdb Namespace

The compact node definitions for the "vdb" namespace are as follows:

```
<nt  = "http://www.jcp.org/jcr/nt/1.0">
<xmi = "http://www.omg.org/XMI">
<vdb = "http://www.metamatrix.com/metamodels/VirtualDatabase">
<mmcore = "http://www.metamatrix.com/metamodels/Core">

//------------------------------------------------------------------------
--------
// N O D E T Y P E S
//------------------------------------------------------------------------
--------

[vdb:virtualDatabase] > nt:unstructured
 - vdb:description (string)
 - vdb:version (long) = '1'
 - vdb:preview (boolean) = 'false'
 - vdb:originalFile (string)
 - mmcore:sha1 (string)

[vdb:model] > xmi:model, mmcore:model
 - vdb:visible (boolean) = 'true'
 - vdb:checksum (long)
 - vdb:builtIn (boolean) = 'false'
 - vdb:pathInVdb (string)
 - vdb:sourceTranslator (string)
 - vdb:sourceJndiName (string)
 - vdb:sourceName (string)
 + vdb:markers (vdb:markers) = vdb:markers copy

[vdb:markers] > nt:unstructured
 + vdb:marker (vdb:marker) = vdb:marker copy sns

[vdb:marker] > nt:unstructured
 - vdb:severity (string) = 'WARNING' < 'WARNING','ERROR','INFO'
 - vdb:path (string)
 - vdb:message (string)
```

12.5. Red Hat JBoss Data Virtualization VDB Sequencer Example

Here is a representation of the nodes output by the sequencing of an example "**qe.2.vdb**" virtual database:

```
qe jcr:primaryType="vdb:virtualDatabase"
  - jcr:mixinTypes=["mix:referenceable", "mode:derived"]
  - jcr:uuid="1d110326-f8e9-4f5e-becd-2f3e4d63296e"
  - mode:derivedAt="2011-05-13T13:12:03.925Z"
  - mode:derivedFrom="/files/foo.vdb"
  - vdb:description="This VDB is for testing Recursive XML documents and Text
Sources"
  - vdb:originalFile="/vdb/qe.vdb"
  - vdb:preview="false"
  - mode:sha1="4cec9166f20a8d3772a1cfddb493329e35c3adb7"
  - vdb:version="2"
```

```
  text jcr:primaryType="vdb:model" jcr:mixinTypes=
["mmcore:model","mix:referenceable","xmi:referenceable"] jcr:uuid="5cffd0ee-
2edd-44af-8a8d-46459d849afe"
    - vdb:builtIn="true"
    - vdb:checksum="958072371"
    - mmcore:maxSetSize="100"
    - mmcore:modelType="PHYSICAL"
    - mmcore:originalFile="/vdb/qe.vdb"
    - vdb:pathInVdb="QuickText/text.xmi"
    -
mmcore:primaryMetamodelUri="http://www.metamatrix.com/metamodels/Relational"
    - mmcore:producerName="Teiid Designer"
    - mmcore:producerVersion="7.0.0.v20100807-1026-H168-M1"
    - mode:sha1="893accdcb0745f8061626b4ab60079daeb3eb74f"
    - vdb:sourceJndiName="empdata-file"
    - vdb:sourceName="text"
    - vdb:sourceTranslator="file"
    - mmcore:supportsDistinct="true"
    - mmcore:supportsJoin="true"
    - mmcore:supportsOrderBy="true"
    - mmcore:supportsOuterJoin="true"
    - mmcore:supportsWhereAll="true"
    - xmi:uuid="ba1f1ca6-b9a7-44f8-9d89-8d9ba9f801ba"
    - xmi:version="2.0"
    - mmcore:visible="true"
    - vdb:visible="true"
    vdb:markers jcr:primaryType="vdb:markers"
      vdb:marker jcr:primaryType="vdb:marker"
        - vdb:message="Missing or invalid Length on column with a
string/character datatype  (See validation Preferences)"
        - vdb:path="getTextFiles/NewProcedureResult/filePath"
        - vdb:severity="WARNING"
    XMLSchema jcr:primaryType="mmcore:import" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="1787cc24-d545-437c-a7ef-
e18569eec9c3"
      - mmcore:modelType="TYPE"
      - mmcore:primaryMetamodelUri="http://www.eclipse.org/xsd/2002/XSD"
      - xmi:uuid="mmuuid:5a23faba-871a-490e-9799-efdffea80b6b"
    SimpleDatatypes-instance jcr:primaryType="mmcore:import" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="4e11258b-06e2-4d39-8a10-
7e6b8e02dc37"
      - mmcore:modelType="TYPE"
      - mmcore:primaryMetamodelUri="http://www.eclipse.org/xsd/2002/XSD"
      - xmi:uuid="mmuuid:b09c455c-1c5a-4de4-8373-e823482ce517"
    getTextFiles jcr:primaryType="relational:procedure" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="137968e0-e375-43c3-b25a-
03953ff975ff"
      - xmi:uuid="bf60b5cb-fd8c-474a-9f4c-68eb42ca40f2"
      pathAndPattern jcr:primaryType="relational:procedureParameter"
jcr:mixinTypes=["mix:referenceable","xmi:referenceable"] jcr:uuid="2a2de9a3-
561d-4d14-82cf-8155b965d2bb"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
```

```
        - xmi:uuid="f44bb026-8bdf-413b-b705-65dcd40bf437"
      NewProcedureResult jcr:primaryType="relational:procedureResult"
jcr:mixinTypes=["mix:referenceable","xmi:referenceable"] jcr:uuid="b5d7be35-
9a73-477e-ab20-5a4b9248da9f"
        - xmi:uuid="eb2f5c65-bede-4dd2-8c85-441c240ebca1"
        file jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="2ebe184b-25ce-4cb4-89f4-
0e6289112c68"
          - relational:autoIncremented="false"
          - relational:caseSensitive="true"
          - relational:currency="false"
          - relational:distinctValueCount="-1"
          - relational:nullValueCount="-1"
          - relational:nullable="NULLABLE"
          - relational:radix="10"
          - relational:searchability="SEARCHABLE"
          - relational:selectable="true"
          - relational:signed="true"
          -
relational:typeHref="http://www.metamatrix.com/metamodels/SimpleDatatypes-
instance#clob"
          - relational:typeName="clob"
          - relational:typeXmiUuid="559646c0-4941-1ece-b22b-f49159d22ad3"
          - relational:updateable="true"
          - xmi:uuid="092a2a85-7ec6-40da-9437-afd0812eccbb"
        filePath jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="ec43aa0d-20df-49ff-8b4f-
ed95961aa9a5"
          - relational:autoIncremented="false"
          - relational:caseSensitive="true"
          - relational:currency="false"
          - relational:distinctValueCount="-1"
          - relational:nullValueCount="-1"
          - relational:nullable="NULLABLE"
          - relational:radix="10"
          - relational:searchability="SEARCHABLE"
          - relational:selectable="true"
          - relational:signed="true"
          - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
          - relational:typeName="string"
          - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
          - relational:updateable="true"
          - xmi:uuid="232d5fd7-e5a6-49a7-bd5f-e5d6b7e753a3"
  Employees jcr:primaryType="vdb:model" jcr:mixinTypes=
["mmcore:model","mix:referenceable","xmi:referenceable"] jcr:uuid="88ca643b-
ebab-43a1-902c-462f3ea17fd8"
    - vdb:builtIn="true"
    - vdb:checksum="1269937912"
    - mmcore:maxSetSize="100"
    - mmcore:modelType="VIRTUAL"
    - mmcore:originalFile="/vdb/qe.vdb"
    - vdb:pathInVdb="QuickEmployees/Employees.xmi"
    -
mmcore:primaryMetamodelUri="http://www.metamatrix.com/metamodels/Relational"
    - mmcore:producerName="Teiid Designer"
    - mmcore:producerVersion="7.0.0.v20100807-1026-H168-M1"
```

```
    - mode:sha1="a63c108098232739aad1d6ab4cf0d3cc1911aa12"
    - mmcore:supportsDistinct="true"
    - mmcore:supportsJoin="true"
    - mmcore:supportsOrderBy="true"
    - mmcore:supportsOuterJoin="true"
    - mmcore:supportsWhereAll="true"
    - xmi:uuid="9c034c0d-10c7-4fa5-beae-ff602bfcf88e"
    - xmi:version="2.0"
    - mmcore:visible="true"
    - vdb:visible="true"
    vdb:markers jcr:primaryType="vdb:markers"
      vdb:marker jcr:primaryType="vdb:marker"
        - vdb:message="Possible cross-join: Group/s '[f, emp]' are not
joined either directly or transitively to other groups through a join
criteria. Check all queries in the transformation."
        - vdb:path="EmpTable"
        - vdb:severity="WARNING"
    text jcr:primaryType="mmcore:import" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="ee4b288a-47b0-4c81-98e5-
ddf01f8a4cda"
      - mmcore:modelType="PHYSICAL"
      -
mmcore:primaryMetamodelUri="http://www.metamatrix.com/metamodels/Relational"
      - xmi:uuid="mmuuid:46ba6b40-bb81-43ba-996e-6f3ebaffea3b"
    SimpleDatatypes-instance jcr:primaryType="mmcore:import" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="98b1dbae-5ad6-4439-adb0-
64d6e5d0a42f"
      - mmcore:modelType="TYPE"
      - mmcore:primaryMetamodelUri="http://www.eclipse.org/xsd/2002/XSD"
      - xmi:uuid="mmuuid:36a2080b-7243-445c-a153-79a19d42f558"
    XMLSchema jcr:primaryType="mmcore:import" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="71eba18c-195e-47ec-b925-
415f981bcd45"
      - mmcore:modelType="TYPE"
      - mmcore:primaryMetamodelUri="http://www.eclipse.org/xsd/2002/XSD"
      - xmi:uuid="mmuuid:ea4a1ff7-fa32-4348-b5a2-192c554b70a4"
    EmpTable jcr:primaryType="relational:baseTable" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="6209d827-62eb-4909-8e66-edbf615a42db"
      - transform:deleteAllowed="true"
      - transform:deleteSqlDefault="true"
      - transform:inputHrefs="../QuickText/text.xmi#mmuuid/bf60b5cb-fd8c-
474a-9f4c-68eb42ca40f2"
      - transform:inputNames="getTextFiles"
      - transform:inputXmiUuids="bf60b5cb-fd8c-474a-9f4c-68eb42ca40f2"
      - transform:inputs="137968e0-e375-43c3-b25a-03953ff975ff"
      - transform:insertAllowed="true"
      - transform:insertSqlDefault="true"
      - relational:materialized="false"
      - transform:selectSql="SELECT * FROM (EXEC
text.getTextFiles('EmpData.txt')) AS f, TEXTTABLE(F.file COLUMNS lastName
string, firstName string, middleName string, empId biginteger, department
string, annualSalary double, title string, homePhone string, mgrId
biginteger, street string, city string, state string, ZipCode string HEADER
3) AS emp"
      - relational:supportsUpdate="true"
```

```
      - relational:system="false"
      - transform:updateAllowed="true"
      - transform:updateSqlDefault="true"
      - xmi:uuid="6179a495-7b7e-4e12-9da3-998e4f709de4"
      file jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="de7902d1-9782-4137-a002-85681e45c0c6"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="../QuickText/text.xmi#mmuuid/092a2a85-7ec6-
40da-9437-afd0812eccbb"
        - transform:inputNames="file"
        - transform:inputXmiUuids="092a2a85-7ec6-40da-9437-afd0812eccbb"
        - transform:inputs="2ebe184b-25ce-4cb4-89f4-0e6289112c68"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        -
relational:typeHref="http://www.metamatrix.com/metamodels/SimpleDatatypes-
instance#clob"
        - relational:typeName="clob"
        - relational:typeXmiUuid="559646c0-4941-1ece-b22b-f49159d22ad3"
        - relational:updateable="true"
        - xmi:uuid="5ca79549-8edc-4972-9d05-cb3066d41676"
      filePath jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="4b19a0f6-d65b-4b5b-845f-f30027947f6c"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="../QuickText/text.xmi#mmuuid/232d5fd7-e5a6-
49a7-bd5f-e5d6b7e753a3"
        - transform:inputNames="filePath"
        - transform:inputXmiUuids="232d5fd7-e5a6-49a7-bd5f-e5d6b7e753a3"
        - transform:inputs="ec43aa0d-20df-49ff-8b4f-ed95961aa9a5"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="fea43d8f-94e4-41f3-9743-3286f8c28590"
      lastName jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="6672adb4-1ded-4289-989d-
3b707fc7384b"
```

```
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="f0b80cce-dd11-44b7-ab2d-4e382befd701"
      firstName jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="13986096-1e2f-483b-b603-
3e7098bc0897"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="aae0eea7-fb09-4b46-9a41-8815bf5331db"
      middleName jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="53ba886e-e6b7-4771-8a70-
bf0e6c9cad63"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="74333281-f3f8-4907-8ac1-4c819dfc76a8"
      empId jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="80fcf4ac-d51d-4d24-b84f-
```

```
3db7dbbcfa2b"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        -
relational:typeHref="http://www.metamatrix.com/metamodels/SimpleDatatypes-
instance#biginteger"
        - relational:typeName="biginteger"
        - relational:typeXmiUuid="822b9a40-a066-1e26-9b08-d6079ebe1f0d"
        - relational:updateable="true"
        - xmi:uuid="5e42fcfc-fe7a-476d-8b55-8a5ce0cd7050"
      department jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="8c805831-d1f9-4070-9e4b-
a95234e6a7d7"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="36ea6df0-ddc0-4311-be2e-f4a6cbe2b580"
      annualSalary jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="d9ee87bd-3567-4446-80f8-
17bead52dd4b"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#double"
        - relational:typeName="double"
        - relational:typeXmiUuid="1f18b140-c4a3-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="79c7b080-c9de-42c9-b252-a449d44e5d34"
      title jcr:primaryType="relational:column" jcr:mixinTypes=
```

```
["mix:referenceable","xmi:referenceable"] jcr:uuid="794a1e06-6160-4255-8f7a-
23f30e5e9af5"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="001ac238-21c6-45f3-8959-3fa0c7bea6c6"
      homePhone jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="24edebac-093d-4aaf-8063-
2b4e48c9f08d"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="432c3937-e7ad-40de-9cb4-deb9d52511b2"
      mgrId jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="c61173a4-27c8-41c2-bebf-
05e24fa82f94"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        -
relational:typeHref="http://www.metamatrix.com/metamodels/SimpleDatatypes-
instance#biginteger"
        - relational:typeName="biginteger"
        - relational:typeXmiUuid="822b9a40-a066-1e26-9b08-d6079ebe1f0d"
        - relational:updateable="true"
```

```
        - xmi:uuid="9c7b26dc-bbf6-4b83-9f03-438ad6a0b3f0"
      street jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="ae505cbf-13bd-4c87-b020-
526dece5c8b9"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="1181dfe5-0d2b-4331-b10b-5d6409dd6cbe"
      city jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="7f32a1c1-622f-40ff-8005-
ab42bb02a857"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="60792162-1659-416b-a6da-b78119429247"
      state jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="41067115-320e-44e3-a70a-
8a71e85fa8d8"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
```

```
        - relational:updateable="true"
        - xmi:uuid="67ed3d16-7fd6-43bb-b16a-61579a49db91"
      ZipCode jcr:primaryType="relational:column" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="93c47676-fec3-46fa-aa19-
777be6136de2"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="2c2267b6-bddf-4d42-aef8-7d24e7527b65"
  EmpV jcr:primaryType="vdb:model" jcr:mixinTypes=
["mmcore:model","mix:referenceable","xmi:referenceable"] jcr:uuid="c9722b47-
03ad-4cdd-81d1-d75e639517a1"
    - vdb:builtIn="true"
    - vdb:checksum="2273245105"
    - mmcore:maxSetSize="100"
    - mmcore:modelType="VIRTUAL"
    - mmcore:originalFile="/vdb/qe.vdb"
    - vdb:pathInVdb="QuickEmployees/EmpV.xmi"
    -
mmcore:primaryMetamodelUri="http://www.metamatrix.com/metamodels/Relational"
    - mmcore:producerName="Teiid Designer"
    - mmcore:producerVersion="7.0.0.v20100807-1026-H168-M1"
    - mode:sha1="502cc1e3dbec4c5cd880662473e8dc2a668d5e78"
    - mmcore:supportsDistinct="true"
    - mmcore:supportsJoin="true"
    - mmcore:supportsOrderBy="true"
    - mmcore:supportsOuterJoin="true"
    - mmcore:supportsWhereAll="true"
    - xmi:uuid="e17f3917-d880-4bad-9a19-7d0f8f3d2135"
    - xmi:version="2.0"
    - mmcore:visible="true"
    - vdb:visible="true"
    vdb:markers jcr:primaryType="vdb:markers"
      vdb:marker jcr:primaryType="vdb:marker"
        - vdb:message="Missing or invalid Precision on column with a numeric
datatype (See validation Preferences)"
        - vdb:path="EmpTable/empId"
        - vdb:severity="WARNING"
    XMLSchema jcr:primaryType="mmcore:import" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="54a5401e-3bab-4918-81cb-
4a278d0263c4"
      - mmcore:modelType="TYPE"
      - mmcore:primaryMetamodelUri="http://www.eclipse.org/xsd/2002/XSD"
      - xmi:uuid="mmuuid:deb854d2-af4d-4158-9846-4ac17f207291"
```

```
    SimpleDatatypes-instance jcr:primaryType="mmcore:import" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="699f153e-7301-4ef4-bffa-
7522475f8c0a"
      - mmcore:modelType="TYPE"
      - mmcore:primaryMetamodelUri="http://www.eclipse.org/xsd/2002/XSD"
      - xmi:uuid="mmuuid:6471e823-eeee-46e8-8d7d-fb00b336cfe7"
    Employees jcr:primaryType="mmcore:import" jcr:mixinTypes=
["mix:referenceable","xmi:referenceable"] jcr:uuid="5f672add-38cd-469f-a180-
ac75306298b5"
      - mmcore:modelType="VIRTUAL"
      -
mmcore:primaryMetamodelUri="http://www.metamatrix.com/metamodels/Relational"
      - xmi:uuid="mmuuid:5806eb6e-fc70-4ad7-b7ff-13f14ec00ca2"
    EmpTable jcr:primaryType="relational:baseTable" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="0568b4b9-44c9-4798-9bee-662094015d67"
      - transform:deleteAllowed="true"
      - transform:deleteSqlDefault="true"
      - transform:inputHrefs="Employees.xmi#mmuuid/6179a495-7b7e-4e12-9da3-
998e4f709de4"
      - transform:inputNames="EmpTable"
      - transform:inputXmiUuids="6179a495-7b7e-4e12-9da3-998e4f709de4"
      - transform:inputs="6209d827-62eb-4909-8e66-edbf615a42db"
      - transform:insertAllowed="true"
      - transform:insertSqlDefault="true"
      - relational:materialized="false"
      - transform:selectSql="SELECT "Employees.EmpTable.lastName",
"Employees.EmpTable.firstName", "Employees.EmpTable.middleName",
"Employees.EmpTable.empId", "Employees.EmpTable.department",
"Employees.EmpTable.annualSalary", "Employees.EmpTable.title",
"Employees.EmpTable.homePhone", "Employees.EmpTable.mgrId",
"Employees.EmpTable.street", "Employees.EmpTable.city",
"Employees.EmpTable.state", "Employees.EmpTable.ZipCode" FROM
"Employees.EmpTable""
      - relational:supportsUpdate="true"
      - relational:system="false"
      - transform:updateAllowed="true"
      - transform:updateSqlDefault="true"
      - xmi:uuid="92cbc96b-f080-42d6-85dc-95cd07edd682"
      lastName jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="cf26def7-de5a-4a1a-8276-f48d988439e4"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/f0b80cce-dd11-44b7-
ab2d-4e382befd701"
        - transform:inputNames="lastName"
        - transform:inputXmiUuids="f0b80cce-dd11-44b7-ab2d-4e382befd701"
        - transform:inputs="6672adb4-1ded-4289-989d-3b707fc7384b"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
```

```
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="a4c30553-7f10-445b-971b-c54cee534639"
      firstName jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="5b932651-6de2-4e89-917f-0d515a5270b0"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/aae0eea7-fb09-4b46-
9a41-8815bf5331db"
        - transform:inputNames="firstName"
        - transform:inputXmiUuids="aae0eea7-fb09-4b46-9a41-8815bf5331db"
        - transform:inputs="13986096-1e2f-483b-b603-3e7098bc0897"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="a48c7515-d271-45ed-8920-22cf8c9d01bb"
      middleName jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="ac7511b4-4b3d-4d42-886e-26ea5b669b5b"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/74333281-f3f8-4907-
8ac1-4c819dfc76a8"
        - transform:inputNames="middleName"
        - transform:inputXmiUuids="74333281-f3f8-4907-8ac1-4c819dfc76a8"
        - transform:inputs="53ba886e-e6b7-4771-8a70-bf0e6c9cad63"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="1d792d5e-ae70-4855-b59f-3eb7dceeb5a3"
      empId jcr:primaryType="relational:column" jcr:mixinTypes=
```

```
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="d1a61bb1-f94d-40c9-bee7-f91d12b26d80"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/5e42fcfc-fe7a-476d-
8b55-8a5ce0cd7050"
        - transform:inputNames="empId"
        - transform:inputXmiUuids="5e42fcfc-fe7a-476d-8b55-8a5ce0cd7050"
        - transform:inputs="80fcf4ac-d51d-4d24-b84f-3db7dbbcfa2b"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        -
relational:typeHref="http://www.metamatrix.com/metamodels/SimpleDatatypes-
instance#biginteger"
        - relational:typeName="biginteger"
        - relational:typeXmiUuid="822b9a40-a066-1e26-9b08-d6079ebe1f0d"
        - relational:updateable="true"
        - xmi:uuid="d9cc45f7-c9de-44f9-b22e-3674b1a7d33c"
      department jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="46c5910c-0cb0-481c-bbad-577e52ac9c96"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/36ea6df0-ddc0-4311-
be2e-f4a6cbe2b580"
        - transform:inputNames="department"
        - transform:inputXmiUuids="36ea6df0-ddc0-4311-be2e-f4a6cbe2b580"
        - transform:inputs="8c805831-d1f9-4070-9e4b-a95234e6a7d7"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="19932ef1-4794-496d-a98b-027971cb5599"
      annualSalary jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="61cb7c5a-c7f8-420d-b84a-0a4cabceed81"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/79c7b080-c9de-42c9-
```

```
b252-a449d44e5d34"
        - transform:inputNames="annualSalary"
        - transform:inputXmiUuids="79c7b080-c9de-42c9-b252-a449d44e5d34"
        - transform:inputs="d9ee87bd-3567-4446-80f8-17bead52dd4b"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#double"
        - relational:typeName="double"
        - relational:typeXmiUuid="1f18b140-c4a3-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="925999e2-15a5-4728-a76e-e0c9ae235d80"
      title jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="e2db3e5e-d2c9-462c-9581-24c700792f0c"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/001ac238-21c6-45f3-
8959-3fa0c7bea6c6"
        - transform:inputNames="title"
        - transform:inputXmiUuids="001ac238-21c6-45f3-8959-3fa0c7bea6c6"
        - transform:inputs="794a1e06-6160-4255-8f7a-23f30e5e9af5"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="36cf25e4-5164-4c9f-81a9-572d0fc11e8b"
      homePhone jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="6ef5a18f-db08-44e6-beb0-d7e6842f3ca5"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/432c3937-e7ad-40de-
9cb4-deb9d52511b2"
        - transform:inputNames="homePhone"
        - transform:inputXmiUuids="432c3937-e7ad-40de-9cb4-deb9d52511b2"
        - transform:inputs="24edebac-093d-4aaf-8063-2b4e48c9f08d"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
```

```
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="8a782b60-0296-4e10-85b1-e0ec03b34d00"
      mgrId jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="11a41f81-5a32-434a-8cd9-04bbbdc4b00c"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/9c7b26dc-bbf6-4b83-
9f03-438ad6a0b3f0"
        - transform:inputNames="mgrId"
        - transform:inputXmiUuids="9c7b26dc-bbf6-4b83-9f03-438ad6a0b3f0"
        - transform:inputs="c61173a4-27c8-41c2-bebf-05e24fa82f94"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        -
relational:typeHref="http://www.metamatrix.com/metamodels/SimpleDatatypes-
instance#biginteger"
        - relational:typeName="biginteger"
        - relational:typeXmiUuid="822b9a40-a066-1e26-9b08-d6079ebe1f0d"
        - relational:updateable="true"
        - xmi:uuid="4f0439b3-8899-44f9-99a6-30971c4a563f"
      street jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="8b7dea23-1a6b-4a76-9fbf-5b4770b8cac9"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/1181dfe5-0d2b-4331-
b10b-5d6409dd6cbe"
        - transform:inputNames="street"
        - transform:inputXmiUuids="1181dfe5-0d2b-4331-b10b-5d6409dd6cbe"
        - transform:inputs="ae505cbf-13bd-4c87-b020-526dece5c8b9"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="c68dfbeb-bc26-4932-ae0b-d354ed000a4e"
```

```
      city jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="a1c94897-33ae-4d9e-9dd5-7c9a09e7ebf2"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/60792162-1659-416b-
a6da-b78119429247"
        - transform:inputNames="city"
        - transform:inputXmiUuids="60792162-1659-416b-a6da-b78119429247"
        - transform:inputs="7f32a1c1-622f-40ff-8005-ab42bb02a857"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="b27874ae-4c7b-4545-84f1-5d95c6a70b3a"
      state jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="d5fded76-1ddc-4a87-af76-a566bb5919dc"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/67ed3d16-7fd6-43bb-
b16a-61579a49db91"
        - transform:inputNames="state"
        - transform:inputXmiUuids="67ed3d16-7fd6-43bb-b16a-61579a49db91"
        - transform:inputs="41067115-320e-44e3-a70a-8a71e85fa8d8"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="248a034b-7331-46ee-a4a4-5db5176ce1bc"
      ZipCode jcr:primaryType="relational:column" jcr:mixinTypes=
["transform:transformed","mix:referenceable","xmi:referenceable"]
jcr:uuid="a73b2a7c-9b19-4697-ae83-f7be5cca7778"
        - relational:autoIncremented="false"
        - relational:caseSensitive="true"
        - relational:currency="false"
        - relational:distinctValueCount="-1"
        - transform:inputHrefs="Employees.xmi#mmuuid/2c2267b6-bddf-4d42-
```

```
aef8-7d24e7527b65"
        - transform:inputNames="ZipCode"
        - transform:inputXmiUuids="2c2267b6-bddf-4d42-aef8-7d24e7527b65"
        - transform:inputs="93c47676-fec3-46fa-aa19-777be6136de2"
        - relational:length="10"
        - relational:nullValueCount="-1"
        - relational:nullable="NULLABLE"
        - relational:radix="10"
        - relational:searchability="SEARCHABLE"
        - relational:selectable="true"
        - relational:signed="true"
        - relational:typeHref="http://www.w3.org/2001/XMLSchema#string"
        - relational:typeName="string"
        - relational:typeXmiUuid="bf6c34c0-c442-1e24-9b01-c8207cd53eb7"
        - relational:updateable="true"
        - xmi:uuid="836656b4-30b1-4c57-a64b-f810763a4a0c"
```

12.6. Configuring a Red Hat JBoss Data Virtualization VDB Sequencer

1. **Include the relevant libraries**

   Include **`modeshape-sequencer-teiid-`*`VERSION`*`.jar`** in your application.

2. **Choose one of the following for sequencing configuration**

   A. Define sequencing configuration based on standard example provided in ***`SOA-ROOT`*`/eds/modeshape/resources/modeshape-config-standard.xml`**:

   ```
   <mode:sequencer jcr:name="Teiid VDB Sequencer"
   mode:classname="org.modeshape.sequencer.teiid.VdbSequencer">
     <mode:description>
       Sequences Teiid Virtual Databases (e.g., *.vdb) loaded under
   '/files', extracting the VDB metadata and the structure defined in
   the VDB's relational models.
     </mode:description>
     <mode:pathExpression>
       eds-store:default:/files(//)(*.vdb[*])/jcr:content[@jcr:data] =>
   eds-store:default:/sequenced/teiid/vdbs$1
     </mode:pathExpression>
   </mode:sequencer>
   ```

   B. Configure via **`org.modeshape.jcr.JcrConfiguration`**:

   ```
   JcrConfiguration config = ...

   config.sequencer("Teiid VDB Sequencer")
         .usingClass("org.modeshape.sequencer.teiid.VdbSequencer")
         .loadedFromClasspath()
         .setDescription("Sequences Teiid Virtual Databases (e.g.,
   *.vdb) loaded under '/files', extracting the VDB metadata and the
   structure defined in the VDB's relational models.")
         .sequencingFrom("/files(//)
   (*.vdb[*])/jcr:content[@jcr:data]")
         .andOutputtingTo("/sequenced/teiid/vdbs$1");
   ```

Note

Refer to ***SOA-ROOT*/eds/modeshape/resources/modeshape-config-standard.xml** for more information.

Chapter 13. Red Hat JBoss Data Virtualization Text Extractor

13.1. Text Extractor

The **`TeiidVdbTextExtractor`** class is an implementation of TextExtractor that extracts from Red Hat JBoss Data Virtualization virtual database (i.e., ".vdb") files the virtual database's logical name, description, and version, plus the logical name, description, source name, source translator name, and JNDI name for each of the virtual database's models.

13.2. Configuring Your Text Extractor

This sequencer is not enabled by default, but it is very easy to add this text extractor to the ModeShape configuration. To do so in a configuration file, add the following fragment under the "<mode:textExtractors>" element (which should be immediately under the "<configuration>" root element):

```
<mode:textExtractor jcr:name="VDB Text Extractors">
  <mode:description>Extract text from Red Hat JBoss Data Virtualization VDB
files</mode:description>

<mode:classname>org.modeshape.extractor.teiid.TeiidVdbTextExtractor</mode:cl
assname>
</mode:textExtractor>
```

Then include the "**`modeshape-sequencer-teiid-`*`VERSION`*`.jar`**" file in your application.

Chapter 14. Custom Text Extractors

14.1. Custom Extractors

Creating a custom text extractor involves the following steps:

- Implement the TextExtractor interface with your own implementation, and create unit tests to verify the functionality and expected behavior; and
- Deploy the JAR file with your implementation (as well as any dependencies), and make them available to ModeShape in your application via ModeShape's configuration.

Chapter 15. Web Console

15.1. Web Console

The Web Console is a tool that allows system administrators to monitor and configure services deployed within a running Enterprise Application Platform instance.

15.2. The Web Console and ModeShape

When Red Hat JBoss Data Virtualization is installed on top of the Red Hat JBoss Enterprise Application Platform (EAP), a plug-in is added to the Web Console. This allows system administrators to monitor and configure Data Virtualization services deployed within a running EAP instance via a web interface.

The following dashboards, specific to ModeShape, are available in the tree hierarchy presented on the left side of the Admin Console:

ModeShape

Repositories

Sequencing Service

Sequencers

Connectors

15.3. Web Console: ModeShape Dashboard

From the **`ModeShape`** dashboard, system administrators have access to the following items:

Summary

Control

From the **`Control`** tab administrators can start/restart and stop the ModeShape engine.

15.4. ModeShape Dashboard: Control

Within the Web Console, system administrators can issue the following commands from the **`Control`** tab of the ModeShape Dashboard:

Table 15.1. ModeShape Control Commands Available via the Web Console

Command	Description
Start/restart	Start the ModeShape engine (or restart the ModeShape engine if it is already running).
Shutdown	Stop the ModeShape engine.

15.5. Web Console: Repositories Dashboard

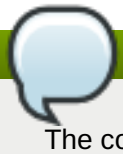
From the **`Repositories`** dashboard, the **`Summary`** tab displays all repositories and indicates whether they are "UP" or "DOWN".

For a selected repository, system administrators have access to the following items:

Summary

Configuration

Metrics



Note

The configuration cannot be modified via Web Console.

15.6. Repositories Dashboard: Metrics

Having selected a repository from the **`Summary`** tab of the **`Repositories`** dashboard, system administrators can then monitor the following metrics from the **`Metrics`** tab for the selected repository:

Table 15.2. Repository Metrics Available via Web Console

Name	Description
Total Active Sessions	The number of JCR sessions that are currently active.

15.7. Web Console: Sequencing Service Dashboard

From the **`Sequencing Service`** dashboard, system administrators have access to the following items:

Summary

Metrics

15.8. Sequencing Service Dashboard: Metrics

Within the Web Console, system administrators can monitor the following metrics from the **`Metrics`** tab of the **`Sequencing Service`** dashboard:

Table 15.3. Sequencing Throughput Metrics Available via the Web Console

Name	Description
Number Of Nodes Sequenced	The number of nodes sequenced.
Number of Nodes Skipped	The number of nodes that were skipped because no sequencers applied.

15.9. Web Console: Sequencers Dashboard

From the **`Sequencers`** dashboard, the **`Summary`** tab displays all sequencers deployed to the ModeShape instance and indicates whether they are "UP" or "DOWN".

For a selected sequencer, system administrators have access to the following items:

Summary

Configuration

Note

The configuration cannot be modified via the Web Console.

15.10. Web Console: Connectors Dashboard

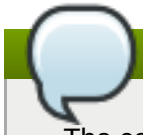
From the **`Connectors`** dashboard, the **`Summary`** tab displays all connectors and indicates whether they are "UP" or "DOWN".

For a selected connector, system administrators have access to the following items:

Configuration

Metrics

Control



Note

The configuration cannot be modified via the Web Console.

15.11. Connectors Dashboard: Metrics

Having selected a connector from the **`Summary`** tab of the **`Connectors`** dashboard, system administrators can then monitor the following metrics from the **`Metrics`** tab for the selected connector:

Table 15.4. Connector Utilization Metrics Available via Admin Console

Name	Description
Total Connections in Use	The number of connections in use.

15.12. Connectors Dashboard: Control

Having selected a connector from the **`Summary`** tab of the **`Connectors`** dashboard, system administrators then have access to the following controls from the **`Control`** tab for the selected connector:

Table 15.5. Connector Control Commands Available via the Web Console

Name	Description
Ping	Ping the connector to test availability.

Chapter 16. Modeshape Core Concepts

16.1. Modeshape is Deprecated

Warning

Modeshape is deprecated. New users should not adopt the features discussed in this section.

16.2. Core Modules

The following modules make up the core components of the system:

- modeshape-jcr contains ModeShape's implementation of the JCR 2.0 API . If you're using ModeShape as a JCR repository, this is the top-level dependency that you'll want to use. The module defines all required dependencies, except for the repository connector(s) and any sequencer implementations needed by your configuration. As we'll see later on, using ModeShape as a JCR repository is as easy as defining a configuration, obtaining the JCR Repository object for your repository using the RepositoryFactory , and then using the standard JCR API. This module also uses the JCR unit tests from the reference implementation to verify the behavior of the ModeShape implementation. modeshape-jcr also provides two essential connectors: the In-Memory Connector and the Federation Connector.
- modeshape-jcr-api is a small module containing ModeShape's public API, which extends the standard JCR API to expose ModeShape-specific functionality. For example, this module defines a Repositories interface, implemented by the ModeShape JcrEngine, that defines a way to look up javax.jcr.Repository instances by name. It also defines several new interfaces that extend the JCR 2.0 API's Query Object Model with additional behavior, including more criteria options (such as BETWEEN, the mode:Depth and jcr:path pseudo-columns, and the REFERENCE function), formal LIMIT and OFFSET clauses, and a set query operator for unions, intersects, and difference queries. Client applications can depend only upon this module without having to depend on the modeshape-jcr interfaces or its dependencies.

16.3. Other Essential Modules

Several other modules are also essential, but for the most part are hidden to client applications as they provide components used within the JCR implementation:

- modeshape-repository provides the core ModeShape graph engine and services for managing repository connections, sequencers, MIME type detectors, and observation. If you're using ModeShape repositories via our graph API rather than JCR , then this is where you'd start.
- modeshape-cnd provides a self-contained utility for parsing CND (Compact Node Definition) files and transforming the node definitions into a graph notation compatible with ModeShape's JCR implementation.
- modeshape-graph defines the Application Programming Interface (API) for ModeShape's low-level graph model, including a fluent-style API for working with graph content. This module also defines the APIs necessary to implement custom connectors, sequencers, and MIME type detectors.
- modeshape-common is a small low-level library of common utilities and frameworks, including logging, progress monitoring, internationalization/localization, text translators, component management, and class loader factories.

16.4. Miscellaneous Optional Modules

16.4. Miscellaneous Optional Modules

Most of the ModeShape modules, however, are optional extensions. The following modules are provided.

- modeshape-clustering contains ModeShape's clustering components and are needed only when two or more ModeShape engines are to be clustered together (so listeners in one session get notifications made from within any of the engines). ModeShape clustering uses the powerful, flexible and mature JGroups reliable multicast communication library. Simply enable clustering in ModeShape's configuration, include this library, and start your cluster. Engines can be dynamically added and removed from the cluster.
- modeshape-connector-filesystem is a ModeShape repository connector that accesses the files and folders on (a part of) the local file system, providing that content in the form of **`nt:file`** and **`nt:folder`** nodes. This connector does support updating the file system when changes are made to the **`nt:file`** and **`nt:folder`** nodes. However, this connector does not support storing other kinds of nodes.
- modeshape-connector-store-jpa is a ModeShape repository connector that stores content in a JDBC database, using the Java Persistence API (JPA) and the very highly-regarded and widely-used Hibernate implementation. This connector is capable of storing any kind of content, and dictates the schema in which it stores the content. Therefore, this connector cannot be used to access the data in existing created by/for other applications.
- modeshape-connector-disk is a ModeShape repository connector that stores content in a ModeShape specific file format on disk. Content is stored in a serialized representation for efficiency. It supports referenceable nodes and can efficiently access nodes by UUID, unlike the File System Connector.
- modeshape-sequencer-cnd is a ModeShape sequencer that extracts JCR node definitions from JCR Compact Node Definition (CND) files.
- modeshape-sequencer-ddl is a ModeShape sequencer that extracts the structure and content from DDL files. This is still under development and includes support for the basic DDL statements in in the Oracle, PostgreSQL, Derby, and standard DDL dialects.
- modeshape-sequencer-zip is a ModeShape sequencer that extracts the files (with content) and directories from ZIP archives.
- modeshape-sequencer-xml is a ModeShape sequencer that extracts the structure and content from XML files.
- modeshape-sequencer-xsd is a ModeShape sequencer that extracts the structure and content from XML Schema Definition (XSD) files.
- modeshape-sequencer-wsdl is a ModeShape sequencer that extracts the structure and content from Web Service Definition Language (WSDL) 1.1 files.
- modeshape-sequencer-sramp is a library with reusable node types patterned after the core model of S-RAMP, and used by other ModeShape sequencers.
- modeshape-sequencer-teiid contains two sequencers. ModelSequencer extracts the structured data model contained with a Data Services relational XMI model, including the catalogs, schemas, tables, views, columns, primary keys, foreign keys, indexes, procedures, procedure parameters, procedure results, logical relationships, and the JDBC source from which the model was imported. Data Services VDB files contain several models, so the VdbSequencer extracts the virtual database metadata and the structured data model from each of the models contained within the VDB.
- modeshape-sequencer-text is a ModeShape sequencer that extracts data from text streams. There are separate sequencers for character-delimited sequencing and fixed width sequencing, but both treat the incoming text stream as a series of rows separated by line-terminators with each row consisting of one or more columns.

- modeshape-search-lucene is an implementation of the SearchEngine interface that uses the Lucene library. This module is one of the few extensions that is used directly by the **`modeshape-jcr`** module.

16.5. Modules for Use with Web Applications

The following modules make up the various web application projects. You may be able to use these artifacts "out of the box", but more likely the configuration defined in the WAR files will not be exactly what you want for your environment. In this case, you can replicate one of our "-war" modules and customize the configuration settings to easily assemble a custom WAR.

- modeshape-web-jcr-webdav provides a WebDAV server for Java Content Repositories. This project provides integration with ModeShape's JCR implementation (of course) but also contains a service provider interface (SPI) that can be used to integrate other JCR implementations with these WebDAV services in the future. For ease of packaging, these classes are provided as a JAR that can be placed in the WEB-INF/lib of a deployed WebDAV server WAR.
- modeshape-web-jcr-webdav-war wraps the WebDAV services from the modeshape-web-jcr-webdav JAR into a WAR and provides in-container integration tests. This project can be consulted as a template for how to deploy the WebDAV services in a custom implementation.
- modeshape-web-jcr-rest provides a set of JSR-311 (JAX-RS) objects that form the basis of a RESTful server for Java Content Repositories. This project provides integration with ModeShape's JCR implementation (of course) but also contains a service provider interface (SPI) that can be used to integrate other JCR implementations with these RESTful services in the future. For ease of packaging, these classes are provided as a JAR that can be placed in the WEB-INF/lib of a deployed RESTful server WAR.
- modeshape-web-jcr-rest-war wraps the RESTful services from the modeshape-web-jcr-rest JAR into a WAR and provides in-container integration tests. This project can be consulted as a template for how to deploy the RESTful services in a custom implementation.
- modeshape-web-jcr-rest-client is a library that uses POJOs to access the REST web service. This module eliminates the need for applications to know how to create HTTP request URLs and payloads, and how to parse the JSON responses. It can be used to publish (upload) and unpublish (delete) files from ModeShape repositories. This module is included within the modeshape-client library.
- modeshape-web-jcr provides a reusable library for web applications using JCR , and is used by the **`modeshape-web-jcr-rest`** and **`modeshape-web-jcr-webdav`** modules.

16.6. Modules for Deploying Modeshape in JBoss

ModeShape recently added several modules that make it very easy to deploy ModeShape in JBoss EAP as a full-fledged, central, shared service that can be monitored and administered using the embedded console and used directly by web applications deployed to the application server.

- modeshape-jbossas-service provides several components that are deployed through the microcontainer in JBoss EAP, registered in JNDI, and exposed through the Profile Service for monitoring and management. This service leverages the JAAS support within the application server. The corresponsing JAR can be found in ***`SOA_ROOT`*`/server/`*`PROFILE`*`/deploy/modeshape-services.jar`**.
- modeshape-jbossas-console defines the plugin for RHQ that enables administration, monitoring, alerting, operational control and configuration. All of the major components within a ModeShape engine are exposed as RHQ resources, and the plugin provides a number of metrics and administrative operations as well as exposing most configuration properties. (We plan to add more metrics and operations over the next few releases, as we gain more experience using the ModeShape RHQ plugin.) The corresponsing JAR can be found in ***`SOA_ROOT`*`/server/`*`PROFILE`*`/deploy/admin-console.war/plugins/`**.

- modeshape-jbossas-web-rest-war defines a variant of the more general **modeshape-web-rest-war** that is tailored for deployment on JBoss EAP, since it reuses the same ModeShape service deployed into the application server. Corresponsing libraries can be found in ***SOA_ROOT*/server/*PROFILE*/deploy/modeshape-rest.war/**.

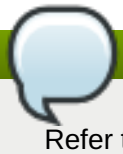
- modeshape-jbossas-web-webdav-war defines a variant of the more general **modeshape-web-webdav-war** that is tailored for deployment on JBoss EAP, since it reuses the same ModeShape service deployed into the application server. Corresponsing libraries can be found in ***SOA_ROOT*/server/*PROFILE*/deploy/modeshape-webdav.war/**.

16.7. Utility Modules

There are several utility modules:

- modeshape-jdbc-local provides a JDBC driver implementation that allows JDBC clients to query the contents of a local JCR repository using JCR-SQL2. The driver supports JDBC metadata, making it possible to dynamically discover the tables and columns available for querying (which are determined from the node types). It can be configured as a data source in JBoss EAP, and can leverage the ModeShape service, allowing JDBC-based access by clients deployed to that JBoss EAP instance to query the repository content. This library is lightweight and fast, since it directly accesses the repository using the JCR API.

- modeshape-jdbc provides a JDBC driver implementation that allows JDBC clients to query the contents of a local or remote JCR repository using JCR-SQL2. The driver supports JDBC metadata, making it possible to dynamically discover the tables and columns available for querying (which are determined from the node types). It can be configured as a data source in JBoss EAP, and can leverage the ModeShape service, allowing JDBC-based access to the same repository content available via the JCR API , RESTful service, or WebDAV. This module is included within the ***SOA_ROOT*/eds/modeshape/client/modeshape-client.jar** file.



Note

Refer to ***SOA_ROOT*/jboss-as/server/*PROFILE*/deploy/modeshape-jdbc-ds.xml** for an example data source file for access to a local repository.

16.8. Dependency Injection

The various components of ModeShape are designed as plain old Java objects, or POJOs (Plain Old Java Objects). And rather than making assumptions about their environment, each component instead requires that any external dependencies necessary for it to operate must be supplied to it. This pattern is known as Dependency Injection, and it allows the components to be simpler and allows for a great deal of flexibility and customization in how the components are configured.

16.9. Execution Context

The approach that ModeShape takes is simple: a simple POJO that represents everything about the environment in which components operate. Called **ExecutionContext**, it contains references to most of the essential facilities, including: security (authentication and authorization); namespace registry; name factories; factories for properties and property values; logging; and access to class loaders (given a classpath). Most of the ModeShape components require an **ExecutionContext** and thus have access to all these facilities.

The fact that so many of the ModeShape components take **`ExecutionContext`** instances gives us some interesting possibilities. For example, one execution context instance can be used as the highest-level (or application-level) context for all of the services (e.g., **`RepositoryService`**, **`SequencingService`**, etc.). Then, an execution context could be created for each user that will be performing operations, and that user's context can be passed around to not only provide security information about the user but also to allow the activities being performed to be recorded for user feedback, monitoring and/or auditing purposes.

16.10. Execution Context Class

The **`ExecutionContext`** is a concrete class that is instantiated with the no-argument constructor:

```
public class ExecutionContext implements ClassLoaderFactory {

    /**
     * Create an instance of an execution context, with default
implementations for all components.
     */
    public ExecutionContext() { ... }

    /**
     * Get the factories that should be used to create values for {@link
Property properties}.
     * @return the property value factory; never null
     */
    public ValueFactories getValueFactories() {...}

    /**
     * Get the namespace registry for this context.
     * @return the namespace registry; never null
     */
    public NamespaceRegistry getNamespaceRegistry() {...}

    /**
     * Get the factory for creating {@link Property} objects.
     * @return the property factory; never null
     */
    public PropertyFactory getPropertyFactory() {...}

    /**
     * Get the security context for this environment.
     * @return the security context; never null
     */
    public SecurityContext getSecurityContext() {...}

    /**
     * Return a logger associated with this context. This logger records
only those activities within the
     * context and provide a way to capture the context-specific activities.
All log messages are also
     * sent to the system logger, so classes that log via this mechanism
should not also
     * {@link Logger#getLogger(Class) obtain a system logger}.
     * @param clazz the class that is doing the logging
     * @return the logger, named after clazz; never null
     */
```

```
    public Logger getLogger( Class<?> clazz ) {...}

    /**
    * Return a logger associated with this context. This logger records only
those activities within the
    * context and provide a way to capture the context-specific activities.
All log messages are also
    * sent to the system logger, so classes that log via this mechanism
should not also
    * {@link Logger#getLogger(Class) obtain a system logger}.
     * @param name the name for the logger
     * @return the logger, named after clazz; never null
     */
    public Logger getLogger( String name ) {...}

  ...
}
```

16.11. Create an Execution Context

As mentioned above, the starting point is to create a default execution context, which will have all the default components:

```
ExecutionContext context = new ExecutionContext();
```

Once you have this top-level context, you can start creating subcontexts with different components, and different security contexts. (Of course, you can create a subcontext from any **`ExecutionContext`** instance.) To create a subcontext, use one of the ***`with(...)`*** methods on the parent context. We'll show examples later on in this chapter.

16.12. Security

ModeShape uses a simple abstraction layer to isolate it from the security infrastructure used within an application. A SecurityContext represents the context of an authenticated user, and is defined as an interface:

```
public interface SecurityContext {

    /**
     * Get the name of the authenticated user.
     * @return the authenticated user's name
     */
    String getUserName();

    /**
     * Determine whether the authenticated user has the given role.
     * @param roleName the name of the role to check
     * @return true if the user has the role and is logged in; false
otherwise
     */
    boolean hasRole( String roleName );

    /**
     * Logs the user out of the authentication mechanism.
```

```
     * For some authentication mechanisms, this will be implemented as a no-
op.
     */
    void logout();
}
```

Every **`ExecutionContext`** has a SecurityContext instance, though the top-level (default) execution context does not represent an authenticated user. But you can create a subcontext for a user authenticated via JAAS :

```
ExecutionContext context = ...
String username = ...
char[] password = ...
String jaasRealm = ...
SecurityContext securityContext = new JaasSecurityContext(jaasRealm,
username, password);
ExecutionContext userContext = context.with(securityContext);
```

In the case of JAAS , you might not have the password but would rather prompt the user. In that case, create a subcontext with a different security context:

```
ExecutionContext context = ...
String jaasRealm = ...
CallbackHandlercallbackHandler = ...
ExecutionContext userContext = context.with(new
JaasSecurityContext(jaasRealm, callbackHandler);
```

Of course if your application has a non- JAAS authentication and authorization system, you can provide your own implementation of SecurityContext :

```
ExecutionContext context = ...
SecurityContext mySecurityContext = ...
ExecutionContext myAppContext = context.with(mySecurityContext);
```

These **`ExecutionContext`**s then represent the authenticated user in any component that uses the context.

16.13. JAAS Security

One of the SecurityContext implementations provided by ModeShape is the **`JaasSecurityContext`**, which delegates any authentication or authorization requests to a Java Authentication and Authorization Service (JAAS) provider. This is the standard approach for authenticating and authorizing in Java.

EDS uses JAAS for all authentication and authorization in ModeShape, using the 'modeshape' policy defined in the 'jboss-as/server/<config>/conf/login-config.xml' file. This policy references the usernames defined in **`soa-users.properties`** and the roles defined in **`soa-roles.properties`**. (The **`ModeShapeEDSRepoDbRealm`** defines the credentials used for the data source in which ModeShape stores its content.)

Modify the 'soa-users.properties' and 'soa-roles.properties' files as required.

16.14. Configuring Users

Each line of the **soa-users.properties** file defines a new user and the roles to which that user belongs, and is of the following form.

```
<username>=<role>[,<role>,...]
```

- <username> is the name of the user
- <role> is the name of a role defined in the **soa-roles.properties** file.

For example, adding "**jsmith=admin,reviewer**" would define the user 'jsmith' with assigned roles, 'admin' and 'reviewer'.

16.15. Configuring Roles

Each line of the **soa-roles.properties** file defines a new role of the following form.

```
<role>=<privilege>[.<repositoryName>[.<workspaceName>]]
```

- <privilege> is one of "admin", "readonly", "readwrite", or (for WebDAV and RESTful access) "connect"
- <repositoryName> is the name of the repository source to which the role is granted; if absent, the role will be granted for all repository sources
- <workspaceName> is the name of the repository workspace to which the role is granted; if absent, the role will be granted for all workspaces in the repository

For example, one of the existing lines is **admin=admin,connect,readonly,readwrite**, which defines the "admin" role as having all privileges. However, a line such as **editCustomer=readonly,readwrite.customer** would define a role allowing read access to all repositories and workspaces, but write access only to the repositories using the "customer" source.

16.16. Web Application Security

If ModeShape is being used within a web application, then it is probably desirable to reuse the security infrastructure of the application server. This can be accomplished by implementing the SecurityContext interface with an implementation that delegates to the HttpServletRequest. Then, for each request, create a SecurityContextCredentials instance around your SecurityContext, and use these credentials to obtain a JCR Session.

Here is an example of the SecurityContext implementation that uses the servlet request:

```
@Immutable
public class ServletSecurityContext implements SecurityContext {

    private final String userName;
    private final HttpServletRequest request;

    /**
     * Create a {@link ServletSecurityContext} with the supplied
     * {@link HttpServletRequest servlet information}.
     *
     * @param request the servlet request; may not be null
     */
    public ServletSecurityContext( HttpServletRequest request ) {
```

```
        this.request = request;
        this.userName = request.getUserPrincipal() != null ?
request.getUserPrincipal().getName() : null;
    }

    /**
     * Get the name of the authenticated user.
     * @return the authenticated user's name
     */
    public String getUserName() {
        return userName;
    }

    /**
     * Determine whether the authenticated user has the given role.
     * @param roleName the name of the role to check
     * @return true if the user has the role and is logged in; false
otherwise
     */
    boolean hasRole( String roleName ) {
        request.isUserInRole(roleName);
    }

    /**
     * Logs the user out of the authentication mechanism.
     * For some authentication mechanisms, this will be implemented as a no-
op.
     */
    public void logout() {
    }
}
```

Then use this to create a Session:

```
HttpServletRequest request = ...
Repository repository = engine.getRepository("my repository");
SecurityContext securityContext = new
ServletSecurityContext(httpServletRequest);
ExecutionContext servletContext = context.with(securityContext);
```

We'll see later how this can be used to obtain a JCR Session for the authenticated user.

16.17. Namespace Registry

As we saw earlier, every ExecutionContext has a registry of namespaces. Namespaces are used throughout the graph API (as we'll see soon), and the prefix associated with each namespace makes for more readable string representations. The namespace registry tracks all of these namespaces and prefixes, and allows registrations to be added, modified, or removed. The interface for the NamespaceRegistry shows how these operations are done:

```
public interface NamespaceRegistry {

    /**
     * Return the namespace URI that is currently mapped to the empty
prefix.
```

```
     * @return the namespace URI that represents the default namespace,
     * or null if there is no default namespace
     */
    String getDefaultNamespaceUri();

    /**
     * Get the namespace URI for the supplied prefix.
     * @param prefix the namespace prefix
     * @return the namespace URI for the supplied prefix, or null if there
is no
     * namespace currently registered to use that prefix
     * @throws IllegalArgumentException if the prefix is null
     */
    String getNamespaceForPrefix( String prefix );

    /**
     * Return the prefix used for the supplied namespace URI.
     * @param namespaceUri the namespace URI
     * @param generateIfMissing true if the namespace URI has not already
been registered and the
     *        method should auto-register the namespace with a generated
prefix, or false if the
     *        method should never auto-register the namespace
     * @return the prefix currently being used for the namespace, or "null"
if the namespace has
     *         not been registered and "generateIfMissing" is "false"
     * @throws IllegalArgumentException if the namespace URI is null
     * @see #isRegisteredNamespaceUri(String)
     */
    String getPrefixForNamespaceUri( String namespaceUri, boolean
generateIfMissing );

    /**
     * Return whether there is a registered prefix for the supplied
namespace URI.
     * @param namespaceUri the namespace URI
     * @return true if the supplied namespace has been registered with a
prefix, or false otherwise
     * @throws IllegalArgumentException if the namespace URI is null
     */
    boolean isRegisteredNamespaceUri( String namespaceUri );

    /**
     * Register a new namespace using the supplied prefix, returning the
namespace URI previously
     * registered under that prefix.
     * @param prefix the prefix for the namespace, or null if a namesapce
prefix should be generated
     *        automatically
     * @param namespaceUri the namespace URI
     * @return the namespace URI that was previously registered with the
supplied prefix, or null if the
     *         prefix was not previously bound to a namespace URI
     * @throws IllegalArgumentException if the namespace URI is null
     */
    String register( String prefix, String namespaceUri );
```

```
    /**
     * Unregister the namespace with the supplied URI.
     * @param namespaceUri the namespace URI
     * @return true if the namespace was removed, or false if the namespace
was not registered
     * @throws IllegalArgumentException if the namespace URI is null
     * @throws NamespaceException if there is a problem unregistering the
namespace
     */
    boolean unregister( String namespaceUri );

    /**
     * Obtain the set of namespaces that are registered.
     * @return the set of namespace URIs; never null
     */
    Set<String> getRegisteredNamespaceUris();

    /**
     * Obtain a snapshot of all of the {@link Namespace namespaces}
registered at the time this method
     * is called. The resulting set is immutable, and will not reflect
changes made to the registry.
     * @return an immutable set of Namespace objects reflecting a snapshot
of the registry; never null
     */
    Set<Namespace> getNamespaces();
}
```

This interfaces exposes Namespace objects that are immutable:

```
@Immutable
interface Namespace extends Comparable<Namespace> {
    /**
     * Get the prefix for the namespace
     * @return the prefix; never null but possibly the empty string
     */
    String getPrefix();

    /**
     * Get the URI for the namespace
     * @return the namespace URI; never null but possibly the empty string
     */
    String getNamespaceUri();
}
```

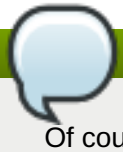
ModeShape actually uses several implementations of NamespaceRegistry , but you can even implement your own and create **`ExecutionContext`**s that use it:

```
NamespaceRegistry myRegistry = ...
ExecutionContext contextWithMyRegistry = context.with(myRegistry);
```

16.18. Classloaders

ModeShape is designed around extensions: sequencers, connectors, MIME type detectors, and class loader factories. The core part of ModeShape is relatively small and has few dependencies, while many of the "interesting" components are extensions that plug into and are used by different parts of the core or by layers above (such as the JCR implementation). The core does not really care what the extensions do or what external libraries they require, as long as the extension fulfills its end of the extension contract.

This means that you only need the core modules of ModeShape on the application classpath, while the extensions do not have to be on the application classpath. And because the core modules of ModeShape have few dependencies, the risk of ModeShape libraries conflicting with the application's are lower. Extensions, on the other hand, will likely have a lot of unique dependencies. By separating the core of ModeShape from the class loaders used to load the extensions, your application is isolated from the extensions and their dependencies.



Note

Of course, you can put all the JARs on the application classpath, too. This is what the examples in the *ModeShape Getting Started Guide* document do.

But in this case, how does ModeShape load all the extension classes? You may have noticed earlier that **`ExecutionContext`** implements the ClassLoaderFactory interface with a single method:

```
public interface ClassLoaderFactory {
    /**
     * Get a class loader given the supplied classpath.  The meaning of the
classpath
     * is implementation-dependent.
     * @param classpath the classpath to use
     * @return the class loader; may not be null
     */
    ClassLoader getClassLoader( String... classpath );
}
```

This means that any component that has a reference to an **`ExecutionContext`** has the ability to create a class loader with a supplied class path. As we'll see later, the connectors and sequencers are all defined with a class and optional class path. This is where that class path comes in.

The actual meaning of the class path, however, is a function of the implementation. ModeShape uses a StandardClassLoaderFactory that just loads the classes using the Thread's current context class loader (or, if there is none, delegates to the class loader that loaded the StandardClassLoaderFactory class). Of course, it is possible to implement other ClassLoaderFactory with other implementations. Then, just create a subcontext with your implementation:

```
ClassLoaderFactory myClassLoaderFactory = ...
ExecutionContext contextWithMyClassLoaderFactories =
context.with(myClassLoaderFactory);
```

16.19. Text Extractors

ModeShape can store all kinds of content, and ModeShape makes it easy to perform full-text searches on that content. To support searching, ModeShape extracts the text from the various properties on each node. The way it does this for most property types (e.g., STRING, LONG, DATE, PATH, NAME, etc.) is to read and use the literal values. But BINARY properties are another story: there's no way to indexes the binary content

directly. Instead, ModeShape has a small pluggable framework for extracting useful text from the binary content, based upon the MIME type of the content itself.

The process works like this: when a BINARY property needs to be indexed for search, ModeShape determines the MIME type of the content, determines if there is a text extractor capable of handling that MIME type, and if so it passes the content to the text extractor and gets back a string of text, and it indexes that text.

The Data Services VDB text extractor operates only upon Data Services virtual database (i.e., ".vdb") files and extracts the virtual database's logical name, description, and version, plus the logical name, description, source name, source translator name, and JNDI name for each of the virtual database's models.

Text extraction can be an intensive process, so it is not enabled by default. But enabling the text extractors in ModeShape's configuration is actually pretty easy. When using a configuration file, add a "<mode:textExtractors>" fragment under the "<configuration>" root element. Within the "<mode:textExtractors>" element place one or more "<mode:textExtractor>" fragments specifying at least the extractor's name and fully-qualified Java class.

For example, here is the fragment that defines the Data Services text extractor.

```
<mode:textExtractors>
    <mode:textExtractor jcr:name="VDB Text Extractors">
      <mode:description>Extract text from Data Services VDB
files</mode:description>

<mode:classname>org.modeshape.extractor.teiid.TeiidVdbTextExtractor</mode:cl
assname>
    </mode:textExtractor>
```

It's also possible to define your own text extractors by implementing the TextExtractor interface:

```
@ThreadSafe
public interface TextExtractor {

    /**
     * Determine if this extractor is capable of processing content with the
supplied MIME type.
     *
     * @param mimeType the MIME type; never null
     * @return true if this extractor can process content with the supplied
MIME type, or false otherwise.
     */
    boolean supportsMimeType( String mimeType );

    /**
     * Sequence the data found in the supplied stream, placing the output
information into the supplied map.
     *
     * ModeShape's SequencingService determines the sequencers that should
be executed by monitoring the changes to one or more
     * workspaces that it is monitoring. Changes in those workspaces are
aggregated and used to determine which sequencers should
     * be called. If the sequencer implements this interface, then this
method is called with the property that is to be sequenced
     * along with the interface used to register the output. The framework
takes care of all the rest.
     *
```

```
     *
     * @param stream the stream with the data to be sequenced; never null
     * @param output the output from the sequencing operation; never null
     * @param context the context for the sequencing operation; never null
     * @throws IOException if there is a problem reading the stream
     */
    void extractFrom( InputStream stream,
                      TextExtractorOutput output,
                      TextExtractorContext context ) throws IOException;

}
```

As mentioned above, the "supportsMimeType" method will be called first, and only if your implementation returns true for a given MIME type will the "extractFrom" method be called. The supplied **`TextExtractorContext`** object provides information about the text being processed, while the TextExtractorOutput is a simple interface that your extractor uses to record one or more strings containing the extracted text.

If you need text extraction in sequencers or connectors, you can always get a TextExtractor instance from the **`ExecutionContext`**. That TextExtractor implementation is actually a composite of all of the text extractors defined in the configuration.

Of course, you can always use a different TextExtractor by creating a subcontext and supplying your implementation:

```
TextExtractor myExtractor = ...
ExecutionContext contextWithMyExtractor = context.with(myExtractor);
```

16.20. Property Factory and Value Factory

Two other components are made available by the **`ExecutionContext`**. The PropertyFactory is an interface that can be used to create Property instances, which are used throughout the graph API . The ValueFactories interface provides access to a number of different factories for different kinds of property values. These will be discussed in much more detail in the next chapter. But like the other components that are in an **`ExecutionContext`**, you can create subcontexts with different implementations:

```
PropertyFactory myPropertyFactory = ...
ExecutionContext contextWithMyPropertyFactory =
context.with(myPropertyFactory);
```

and

```
ValueFactories myValueFactories = ...
ExecutionContext contextWithMyValueFactories =
context.with(myValueFactories);
```

Of course, implementing your own factories is a pretty advanced topic, and it will likely be something you do not need to do in your application.

16.21. Graph Model

One of the central concepts within ModeShape is its graph model. Information is structured into a hierarchy of nodes with properties, where nodes in the hierarchy are identified by their path (and/or identifier properties).

Properties are identified by a name that incorporates a namespace and local name, and contain one or more property values consisting of normal Java strings, names, paths, URIs, booleans, longs, doubles, decimals, binary content, dates, UUIDs, references to other nodes, or any other serializable object.

This graph model is used throughout ModeShape: it forms the basis for the connector framework, it is used by the sequencing framework for the generated output, and it is what the JCR uses internally to access and operate on the repository content.

16.22. Names

ModeShape uses names to identify different types of objects. As we'll soon see, each property of a node is given by a name, and each segment in a path is comprised of a name.

ModeShape names consist of a local part that is qualified with a namespace. The local part can consist of any character, and the namespace is identified by a URI. Namespaces were introduced in the **`ExecutionContext`** and are managed by the **`ExecutionContext`**'s namespace registry. Namespaces help reduce the risk of clashes in names that have an equivalent same local part.

All names are immutable, which means that once a Name object is created, it will never change. This characteristic makes it much easier to write thread-safe code - the objects never change and therefore require no locks or synchronization to guarantee atomic reads. This is a technique that is more and more often found in newer languages and frameworks that simplify concurrent operations.

You should use a factory to create Name instances.

16.23. Name Interface

The Name interface is defined as follows:

```
@Immutable
public interface Name extends Comparable<Name>, Serializable, Readable {

    /**
     * Get the local name part of this qualified name.
     * @return the local name; never null
     */
    String getLocalName();

    /**
     * Get the URI for the namespace used in this qualified name.
     * @return the URI; never null but possibly empty
     */
    String getNamespaceUri();
}
```

16.24. Name Factories

The use of a factory may seem like a disadvantage and unnecessary complexity, but there actually are several benefits. First, it hides the concrete implementations, which is very appealing if an optimized implementation can be chosen for particular situations. It also simplifies the usage, since Name only has a few methods. Third, it allows the factory to cache or pool instances where appropriate to help conserve memory. Finally, the very same factory actually serves as a conversion mechanism from other forms. We'll actually see more of this later in this chapter, when we talk about other kinds of graph properties.

The factory for creating Name objects is called NameFactory and is available within the **`ExecutionContext`**, via the **`getValueFactories()`** method.

We'll see how names are used later on, but one more point to make: Name is both serializable and comparable, and all implementations should support **`equals(...)`** and **`hashCode()`** so that Name can be used as a key in a hash-based map. Name also extends the Readable interface, which we'll learn more about later in this chapter.

16.25. Paths

Another important concept in ModeShape's graph model is the *path*, which provides a way of locating a node within a hierarchy. ModeShape's Path object is an immutable ordered sequence of Path.Segment objects.

Like Name , the only way to create a Path or a Path.Segment is to use the PathFactory , which is available within the **`ExecutionContext`** via the **`getValueFactories()`** method.

16.26. Path Interface

A portion of the Path interface is shown here:

```
@Immutable
public interface Path extends Comparable<Path>, Iterable<Path.Segment>,
Serializable, Readable {

    /**
     * Return the number of segments in this path.
     * @return the number of path segments
     */
    public int size();

    /**
     * Return whether this path represents the root path.
     * @return true if this path is the root path, or false otherwise
     */
    public boolean isRoot();

    /**
     * {@inheritDoc}
     */
    public Iterator<Path.Segment> iterator();

    /**
     * Obtain a copy of the segments in this path. None of the segments are
encoded.
     * @return the array of segments as a copy
     */
    public Path.Segment[] getSegmentsArray();

    /**
     * Get an unmodifiable list of the path segments.
     * @return the unmodifiable list of path segments; never null
     */
    public List<Path.Segment> getSegmentsList();
    /**
```

```
     * Get the last segment in this path.
     * @return the last segment, or null if the path is empty
     */
    public Path.Segment getLastSegment();

    /**
     * Get the segment at the supplied index.
     * @param index the index
     * @return the segment
     * @throws IndexOutOfBoundsException if the index is out of bounds
     */
    public Path.Segment getSegment( int index );

    /**
     * Return an iterator that walks the paths from the root path down to
this path. This method
     * always returns at least one path (the root returns an iterator
containing itself).
     * @return the path iterator; never null
     */
    public Iterator<Path> pathsFromRoot();

    /**
     * Return a new path consisting of the segments starting at beginIndex
index (inclusive).
     * This is equivalent to calling path.subpath(beginIndex,path.size()-1).
     * @param beginIndex the beginning index, inclusive.
     * @return the specified subpath
     * @exception IndexOutOfBoundsException if the beginIndex is negative or
larger
     *            than the length of this Path object
     */
    public Path subpath( int beginIndex );

    /**
     * Return a new path consisting of the segments between the beginIndex
index (inclusive)
     * and the endIndex index (exclusive).
     * @param beginIndex the beginning index, inclusive.
     * @param endIndex the ending index, exclusive.
     * @return the specified subpath
     * @exception IndexOutOfBoundsException if the beginIndex is negative,
or
     *            endIndex is larger than the length of this Path
     *            object, or beginIndex is larger than endIndex.
     */
    public Path subpath( int beginIndex, int endIndex );

    ...
}
```

There are actually quite a few methods (not shown above) for obtaining related paths: the path of the parent, the path of an ancestor, resolving a path relative to this path, normalizing a path (by removing "." and ".." segments), finding the lowest common ancestor shared with another path, etc. There are also a number of methods that compare the path with others, including determining whether a path is above, equal to, or below this path.

16.27. Path Segment Interface

Each Path.Segment is an immutable pair of a Name and same-name-sibling (SNS) index. When two sibling nodes have the same name, then the first sibling will have SNS index of "1" and the second will be given a SNS index of "2". (This mirrors the same-name-sibling index behavior of JCR paths.)

```
@Immutable
public static interface Path.Segment extends Cloneable,
Comparable<Path.Segment>, Serializable, Readable
{

    /**
     * Get the name component of this segment.
     * @return the segment's name
     */
    public Name getName();

    /**
     * Get the index for this segment, which will be 1 by default.
     * @return the index
     */
    public int getIndex();

    /**
     * Return whether this segment has an index that is not "1"
     * @return true if this segment has an index, or false otherwise.
     */
    public boolean hasIndex();

    /**
     * Return whether this segment is a self-reference (or ".").
     * @return true if the segment is a self-reference, or false otherwise.
     */
    public boolean isSelfReference();

    /**
     * Return whether this segment is a reference to a parent (or "..")
     * @return true if the segment is a parent-reference, or false
otherwise.
     */
    public boolean isParentReference();
}
```

16.28. Properties

The ModeShape graph model allows nodes to hold multiple properties, where each property is identified by a unique Name and may have one or more values. Like many of the other classes used in the graph model, Property is an immutable object that, once constructed, can never be changed and therefore provides a consistent snapshot of the state of a property as it existed at the time it was read.

ModeShape properties can hold a wide range of value objects, including normal Java strings, names, paths, URIs, booleans, longs, doubles, decimals, binary content, dates, UUIDs, references to other nodes, or any other serializable object. All but three of these are the standard Java classes: dates are represented by an immutable DateTime class; binary content is represented by an immutable Binary interface patterned after

the interface of the same name in JSR-283; and Reference is an immutable interface patterned after the corresponding interface is JSR-170 and JSR-283.

16.29. Property Interface

The Property interface defines methods for obtaining the name and property values:

```
@Immutable
public interface Property extends Iterable<Object>, Comparable<Property>,
Readable {

    /**
     * Get the name of the property.
     *
     * @return the property name; never null
     */
    Name getName();

    /**
     * Get the number of actual values in this property.
     * @return the number of actual values in this property; always non-
negative
     */
    int size();

    /**
     * Determine whether the property currently has multiple values.
     * @return true if the property has multiple values, or false otherwise.
     */
    boolean isMultiple();

    /**
     * Determine whether the property currently has a single value.
     * @return true if the property has a single value, or false otherwise.
     */
    boolean isSingle();

    /**
     * Determine whether this property has no actual values. This method may
return true
     * regardless of whether the property has a single value or multiple
values.
     * This method is a convenience method that is equivalent to size() ==
0.
     * @return true if this property has no values, or false otherwise
     */
    boolean isEmpty();

    /**
     * Obtain the property's first value in its natural form. This is
equivalent to calling
     * isEmpty() ? null : iterator().next()
     * @return the first value, or null if the property is {@link #isEmpty()
empty}
     */
```

```
    Object getFirstValue();

    /**
     * Obtain the property's values in their natural form. This is
equivalent to calling iterator().
     * A valid iterator is returned if the property has single valued or
multi-valued.
     * The resulting iterator is immutable, and all property values are
immutable.
     * @return an iterator over the values; never null
     */
    Iterator<?> getValues();

    /**
     * Obtain the property's values as an array of objects in their natural
form.
     * A valid iterator is returned if the property has single valued or
multi-valued, or a
     * null value is returned if the property is {@link #isEmpty() empty}.
     * The resulting array is a copy, guaranteeing immutability for the
property.
     * @return the array of values
     */
    Object[] getValuesAsArray();
}
```

16.30. Property Factory

Creating Property instances is done by using the PropertyFactory object owned by the **`ExecutionContext`**. This factory defines methods for creating properties with a Name and various representation of values, including variable-length arguments, arrays, Iterator, and Iterable .

16.31. Property Values

When it comes to using the property values, ModeShape takes a non-traditional approach. Many other graph models (including JCR) mark each property with a data type and then require all property values adhere to this data type. When the property values are obtained, they are guaranteed to be of the correct type. However, many times the property's data type may not match the data type expected by the caller, and so a conversion may be required and thus has to be coded.

The ModeShape graph model uses a different tact. Because callers almost always have to convert the values to the types they can handle, ModeShape skips the steps of associating the Property with a data type and ensuring the values match. Instead, ModeShape provides a very easy mechanism to convert the property values to the type desired by the caller. In fact, the conversion mechanism is exactly the same as the factories that create the values in the first place.

16.32. Value Factories

ModeShape properties can hold a variety of value object types: strings, names, paths, URIs, booleans, longs, doubles, decimals, binary content, dates, UUIDs, references to other nodes, or any other serializable object. To assist in the creation of these values and conversion into other types, ModeShape defines a ValueFactory interface. This interface is parameterized with the type of value that is being created, but defines methods for creating those values from all of the other known value types.

16.33. Value Factory Interface

```
public interface ValueFactory<T> {

    /**
     * Get the PropertyType of values created by this factory.
     * @return the value type; never null
     */
    PropertyType getPropertyType();

  /*
   * Methods to create a value by converting from another value type.
   * If the supplied value is the same type as returned by this factory,
   * these methods return the supplied value.
   * All of these methods throw a ValueFormatException if the supplied value
   * could not be converted to this type.
   */
    T create( String value ) throws ValueFormatException;
    T create( String value, TextDecoder decoder ) throws
ValueFormatException;
    T create( int value ) throws ValueFormatException;
    T create( long value ) throws ValueFormatException;
    T create( boolean value ) throws ValueFormatException;
    T create( float value ) throws ValueFormatException;
    T create( double value ) throws ValueFormatException;
    T create( BigDecimal value ) throws ValueFormatException;
    T create( Calendar value ) throws ValueFormatException;
    T create( Date value ) throws ValueFormatException;
    T create( DateTime value ) throws ValueFormatException;
    T create( Name value ) throws ValueFormatException;
    T create( Path value ) throws ValueFormatException;
    T create( Reference value ) throws ValueFormatException;
    T create( URI value ) throws ValueFormatException;
    T create( UUID value ) throws ValueFormatException;
    T create( byte[] value ) throws ValueFormatException;
    T create( Binary value ) throws ValueFormatException, IoException;
    T create( InputStream stream, long approximateLength ) throws
ValueFormatException, IoException;
    T create( Reader reader, long approximateLength ) throws
ValueFormatException, IoException;
    T create( Object value ) throws ValueFormatException, IoException;

    /*
     * Methods to create an array of values by converting from another array
of values.
     * If the supplied values are the same type as returned by this factory,
     * these methods return the supplied array.
     * All of these methods throw a ValueFormatException if the supplied
values
     * could not be converted to this type.
   */
    T[] create( String[] values ) throws ValueFormatException;
    T[] create( String[] values, TextDecoder decoder ) throws
ValueFormatException;
    T[] create( int[] values ) throws ValueFormatException;
```

```
    T[] create( long[] values ) throws ValueFormatException;
    T[] create( boolean[] values ) throws ValueFormatException;
    T[] create( float[] values ) throws ValueFormatException;
    T[] create( double[] values ) throws ValueFormatException;
    T[] create( BigDecimal[] values ) throws ValueFormatException;
    T[] create( Calendar[] values ) throws ValueFormatException;
    T[] create( Date[] values ) throws ValueFormatException;
    T[] create( DateTime[] values ) throws ValueFormatException;
    T[] create( Name[] values ) throws ValueFormatException;
    T[] create( Path[] values ) throws ValueFormatException;
    T[] create( Reference[] values ) throws ValueFormatException;
    T[] create( URI[] values ) throws ValueFormatException;
    T[] create( UUID[] values ) throws ValueFormatException;
    T[] create( byte[][] values ) throws ValueFormatException;
    T[] create( Binary[] values ) throws ValueFormatException, IoException;
    T[] create( Object[] values ) throws ValueFormatException, IoException;

    /**
     * Create an iterator over the values (of an unknown type). The factory
converts any
     * values as required.  This is useful when wanting to iterate over the
values of a property,
     * where the resulting iterator exposes the desired type.
     * @param values the values
     * @return the iterator of type T over the values, or null if the
supplied parameter is null
     * @throws ValueFormatException if the conversion from an iterator of
objects could not be performed
     * @throws IoException If an unexpected problem occurs during the
conversion.
     */
    Iterator<T> create( Iterator<?> values ) throws ValueFormatException,
IoException;
    Iterable<T> create(Iterable<?> valueIterable ) throws
ValueFormatException, IoException;
}
```

This makes it very easy to convert one or more values (of any type, including mixtures) into corresponding value(s) that are of the desired type. For example, converting the first value of a property (regardless of type) to a String is simple:

```
ValueFactory<String> stringFactory = ...
Property property = ...
String value = stringFactory.create( property.getFirstValue() );
```

Likewise, iterating over the values in a property and converting them is just as easy:

```
ValueFactory<String> stringFactory = ...
Property property = ...
for ( String value : stringFactory.create(property) ) {
    // do something with the values
}
```

What we've glossed over so far, however, is how to obtain the correct ValueFactory for the desired type. If you remember back in the previous chapter, **`ExecutionContext`** has a **`getValueFactories()`** method that return a ValueFactories interface:

```
```

This interface exposes a ValueFactory for each of the types, and even has methods to obtain a ValueFactory given the PropertyType enumeration. So, the previous examples could be expanded a bit:

```
ValueFactory<String> stringFactory =
context.getValueFactories().getStringFactory();
Property property = ...
String value = stringFactory.create( property.getFirstValue() );
```

and

```
ValueFactory<String> stringFactory =
context.getValueFactories().getStringFactory();
Property property = ...
for ( String value : stringFactory.create(property) ) {
    // do something with the values
}
```

16.34. Subinterfaces of a Value Factory

Several of the ValueFactories methods return subinterfaces of ValueFactory; for example, NameFactory , DateTimeFactory , PathFactory , and BinaryFactory . These add type-specific methods that are more commonly needed in certain cases.

ModeShape provides efficient implementations of all of these interfaces: the ValueFactory interfaces and subinterfaces; the Path, Path.Segment, Name, Binary, DateTime, and Reference interfaces; and the ValueFactories interface returned by the **`ExecutionContext`**. In fact, some of these interfaces have multiple implementations that are optimized for specific but frequently-occurring conditions.

16.35. Name Value Factory Interface

```
public interface NameFactory extends ValueFactory<Name> {

    Name create( String namespaceUri, String localName );
    Name create( String namespaceUri, String localName, TextDecoder decoder
);

    NamespaceRegistry getNamespaceRegistry();
}
```

16.36. DateTimeFactory Interface

This interface adds methods for creating DateTime values for the current time as well as for specific instants in time:

```
public interface DateTimeFactory extends ValueFactory<DateTime> {
```

```
    /**
     * Create a date-time instance for the current time in the local time
zone.
     */
    DateTime create();

    /**
     * Create a date-time instance for the current time in UTC.
     */
    DateTime createUtc();

    DateTime create( DateTime original, long offsetInMillis );
    DateTime create( int year, int monthOfYear, int dayOfMonth,
                     int hourOfDay, int minuteOfHour, int secondOfMinute,
int millisecondsOfSecond );
    DateTime create( int year, int monthOfYear, int dayOfMonth,
                     int hourOfDay, int minuteOfHour, int secondOfMinute,
int millisecondsOfSecond,
                     int timeZoneOffsetHours );
    DateTime create( int year, int monthOfYear, int dayOfMonth,
                     int hourOfDay, int minuteOfHour, int secondOfMinute,
int millisecondsOfSecond,
                     int timeZoneOffsetHours, String timeZoneId );
}
```

16.37. PathFactory Interface

The PathFactory interface defines methods for creating relative and absolute Path objects using combinations of other Path objects and Names and Path.Segments, and introduces methods for creating Path.Segment objects:

```
public interface PathFactory extends ValueFactory<Path> {

    Path createRootPath();
    Path createAbsolutePath( Name... segmentNames );
    Path createAbsolutePath( Path.Segment... segments );
    Path createAbsolutePath( Iterable<Path.Segment> segments );

    Path createRelativePath();
    Path createRelativePath( Name... segmentNames );
    Path createRelativePath( Path.Segment... segments );
    Path createRelativePath( Iterable<Path.Segment> segments );

    Path create( Path parentPath, Path childPath );
    Path create( Path parentPath, Name segmentName, int index );
    Path create( Path parentPath, String segmentName, int index );
    Path create( Path parentPath, Name... segmentNames );
    Path create( Path parentPath, Path.Segment... segments );
    Path create( Path parentPath, Iterable<Path.Segment> segments );
    Path create( Path parentPath, String subpath );

    Path.Segment createSegment( String segmentName );
    Path.Segment createSegment( String segmentName, TextDecoder decoder );
```

```
    Path.Segment createSegment( String segmentName, int index );
    Path.Segment createSegment( Name segmentName );
    Path.Segment createSegment( Name segmentName, int index );
}
```

16.38. BinaryFactory Interface

The BinaryFactory defines methods for creating Binary objects from a variety of binary formats, as well as a method that looks for a cached Binary instance given the supplied secure hash:

```
public interface BinaryFactory extends ValueFactory<Binary> {

    /**
     * Create a value from the binary content given by the supplied input,
the approximate length,
     * and the SHA-1 secure hash of the content. If the secure hash is null,
then a secure hash is
     * computed from the content. If the secure hash is not null, it is
assumed to be the hash for
     * the content and may not be checked.
     */
    Binary create( InputStream stream, long approximateLength, byte[]
secureHash )
                           throws ValueFormatException, IoException;
    Binary create( Reader reader, long approximateLength, byte[] secureHash
)
                           throws ValueFormatException, IoException;

    /**
     * Create a binary value from the given file.
     */
    Binary create( File file ) throws ValueFormatException, IoException;

    /**
     * Find an existing binary value given the supplied secure hash. If no
such binary value exists,
     * null is returned. This method can be used when the caller knows the
secure hash (e.g., from
     * a previously-held Binary object), and would like to reuse an existing
binary value
     * (if possible) rather than recreate the binary value by processing the
stream contents. This is
     * especially true when the size of the binary is quite large.
     *
     * @param secureHash the secure hash of the binary content, which was
probably obtained from a
     *        previously-held Binary object; a null or empty value is
allowed, but will always
     *        result in returning null
     * @return the existing Binary value that has the same secure hash, or
null if there is no
     *        such value available at this time
     */
    Binary find( byte[] secureHash );
}
```

16.39. Readable Interface

As shown above, the Name, Path.Segment, Path, and Property interfaces all extend the Readable interface, which defines a number of **`getString(...)`** methods that can produce a (readable) string representation of of that object. Recall that all of these objects contain names with namespace URIs and local names (consisting of any characters), and so obtaining a readable string representation will require converting the URIs to prefixes, escaping certain characters in the local names, and formatting the prefix and escaped local name appropriately. The different **`getString(...)`** methods of the Readable interface accept various combinations of NamespaceRegistry and TextEncoder parameters:

```
@Immutable
public interface Readable {

    /**
     * Get the string form of the object. A default encoder is used to
encode characters.
     * @return the encoded string
     */
    public String getString();

    /**
     * Get the encoded string form of the object, using the supplied encoder
to encode characters.
     * @param encoder the encoder to use, or null if the default encoder
should be used
     * @return the encoded string
     */
    public String getString( TextEncoder encoder );

    /**
     * Get the string form of the object, using the supplied namespace
registry to convert any
     * namespace URIs to prefixes. A default encoder is used to encode
characters.
     * @param namespaceRegistry the namespace registry that should be used
to obtain the prefix
     *        for any namespace URIs
     * @return the encoded string
     * @throws IllegalArgumentException if the namespace registry is null
     */
    public String getString( NamespaceRegistry namespaceRegistry );

    /**
     * Get the encoded string form of the object, using the supplied
namespace registry to convert
     * the any namespace URIs to prefixes.
     * @param namespaceRegistry the namespace registry that should be used
to obtain the prefix for
     *        the namespace URIs
     * @param encoder the encoder to use, or null if the default encoder
should be used
     * @return the encoded string
     * @throws IllegalArgumentException if the namespace registry is null
     */
```

```
    public String getString( NamespaceRegistry namespaceRegistry,
                             TextEncoder encoder );

    /**
     * Get the encoded string form of the object, using the supplied
namespace registry to convert
     * the names' namespace URIs to prefixes and the supplied encoder to
encode characters, and using
     * the second delimiter to encode (or convert) the delimiter used
between the namespace prefix
     * and the local part of any names.
     * @param namespaceRegistry the namespace registry that should be used
to obtain the prefix
     *        for the namespace URIs in the names
     * @param encoder the encoder to use for encoding the local part and
namespace prefix of any names,
     *        or null if the default encoder should be used
     * @param delimiterEncoder the encoder to use for encoding the delimiter
between the local part
     *        and namespace prefix of any names, or null if the standard
delimiter should be used
     * @return the encoded string
     */
    public String getString( NamespaceRegistry namespaceRegistry,
                             TextEncoder encoder, TextEncoder
delimiterEncoder );
}
```

16.40. Text Encoder Interface

We've seen the NamespaceRegistry in the execution context, but we've haven't yet talked about the TextEncoder interface. A TextEncoder merely does what you'd expect: it encodes the characters in a string using some implementation-specific algorithm. ModeShape provides a number of TextEncoder implementations, including:

- The Jsr283Encoder escapes characters that are not allowed in JCR names, per the JSR-283 specification. Specifically, these are the '*', '/', ':', '[', ']', and '|' characters, which are escaped by replacing them with the Unicode characters U+F02A, U+F02F, U+F03A, U+F05B, U+F05D, and U+F07C, respectively.
- The NoOpEncoder does no conversion.
- The UrlEncoder converts text to be used within the different parts of a URL, as defined by Section 2.3 of RFC 2396. Note that this class does not encode a complete URL (since **`java.net.URLEncoder`** and **`java.net.URLDecoder`** should be used for such purposes).
- The XmlNameEncoder converts any UTF-16 unicode character that is not a valid XML name character according to the World Wide Web Consortium (W3C) Extensible Markup Language (XML) 1.0 (Fourth Edition) Recommendation, escaping such characters as _**`xHHHH`**_, where **`HHHH`** stands for the four-digit hexadecimal UTF-16 unicode value for the character in the most significant bit first order. For example, the name "Customer_ID" is encoded as "Customer_x0020_ID".
- The XmlValueEncoder escapes characters that are not allowed in XML values. Specifically, these are the '&', '<', '>', "", and '', which are all escaped to "&", '<', '>', '"', and '''.

- The FileNameEncoder escapes characters that are not allowed in file names on Linux, OS X, or Windows XP. Unsafe characters are escaped as described in the UrlEncoder.
- The SecureHashTextEncoder performs a secure hash of the input text and returns that hash as the encoded text. This encoder can be configured to use different secure hash algorithms and to return a fixed number of characters from the hash.

All of these classes also implement the TextDecoder interface, which defines a method that decodes an encoded string using the opposite transformation.

Of course, you can provide alternative implementations, and supply them to the appropriate **`getString(...)`** methods as required.

16.41. Locations

In addition to Path objects, nodes can be identified by one or more identification properties. These are Property instances with names that have a special meaning (usually to the connector framework). ModeShape also defines a Location class that encapsulates:

- the Path to the node;
- one or more identification properties that are likely source-specific and that are represented with Property objects; or
- a combination of both.

So, when a client knows the path and/or the identification properties, they can create a Location object and then use that to identify the node. Location is a class that can be instantiated through factory methods on the class:

```
public abstract class Location implements Iterable<Property>,
Comparable<Location> {

    public static Location create( Path path ) { ... }
    public static Location create( UUID uuid ) { ... }
    public static Location create( Path path, UUID uuid ) { ... }
    public static Location create( Path path, Property idProperty ) { ... }
    public static Location create( Path path, Property firstIdProperty,
                                   Property... remainingIdProperties ) {
... }
    public static Location create( Path path, Iterable<Property idProperties
) { ... }
    public static Location create( Property idProperty ) { ... }
    public static Location create( Property firstIdProperty,
                                   Property... remainingIdProperties ) {
... }
    public static Location create( Iterable<Property> idProperties ) { ... }
    public static Location create( List<Property> idProperties ) { ... }
    ...
}
```

Like many of the other classes and interfaces, Location is immutable and cannot be changed once created. However, there are methods on the class to create a copy of the Location object with a different Path, a different UUID, or different identification properties:

```
public abstract class Location implements Iterable<Property>,
```

```
Comparable<Location> {
    ...
    public Location with( Property newIdProperty );
    public Path newPath );
    public UUID uuid );
    ...
}
```

When creating requests, clients usually have an incomplete location (e.g., a path but no identification properties). When processing the requests, connectors provide an actual location that contains the path and all identification properties. If actual Location objects are then reused in subsequent requests by the client, the connectors will have the benefit of having both the path and identification properties and may be able to more efficiently locate the identified node.

16.42. Graph API

ModeShape's Graph API was designed as a lightweight public API for working with graph information. The Graph class is the primary class in API , and each instance represents a single, independent view of a single graph. Graph instances don't maintain state, so every request (or batch of requests) operates against the underlying graph and then returns immutable snapshots of the requested state at the time the request was made.

The Graph class represents an internal domain specific language (DSL), designed to be easy to use in an application. The Graph API makes extensive use of interfaces and method chaining, so that methods return a simplified interface with relevant methods only. This is made simpler if your IDE has code completion. Under the hood, a Graph is just building Request objects, submitting them to the connector, and then exposing the results.

16.43. Using Workspaces

ModeShape graphs have the notion of workspaces that provide different views of the content. Some graphs may have one workspace, while others may have multiple workspaces. Some graphs will allow a client to create new workspaces or destroy existing workspaces, while other graphs will not allow adding or removing workspaces. Some graphs may have workspaces that may show the same (or very similar) content, while other graphs may have workspaces that contain completely independent content.

The Graph object is always bound to a workspace, which initially is the default workspace. To find out what the name of the default workspace is, ask for the current workspace after creating the Graph:

```
Workspace current = graph.getCurrentWorkspace();
```

To obtain the list of workspaces available in a graph, ask for them:

```
Set<String> workspaceNames = graph.getWorkspaces();
```

Once you know the name of a particular workspace, you can specify that the graph should use it:

```
graph.useWorkspace("myWorkspace");
```

From this point forward, all requests will apply to the workspace named "myWorkspace". At any time, you can use a different workspace, which will affect all subsequent requests made using the graph. To go back to the default workspace, supply a null name:

```
graph.useWorkspace(null);
```

Of course, creating a new workspace is just as easy:

```
graph.createWorkspace().named("newWorkspace");
```

This will attempt to create a workspace named "newWorkspace", which will fail if that workspace already exists. You may want to create a new workspace with a name that should be altered if the name you supply is already used. The following code shows how you can do this:

```
graph.createWorkspace().namedSomethingLike("newWorkspace");
```

If there is no existing workspace named "newWorkspace", a new one will be created with this name. However, if "newWorkspace" already exists, this call will create a workspace with a name that is some alteration of the supplied name.

You can also clone workspaces, too:

```
graph.createWorkspace().clonedFrom("original").named("something");
```

or

```
graph.createWorkspace().clonedFrom("original").namedSomethingLike("something
");
```

As you can see, it is very easy to specify which workspace you want to use or to create new workspaces. You can also find out which workspace the graph is currently using:

```
String current = graph.getCurrentWorkspaceName();
```

or, if you want, you can get more information about the workspace:

```
Workspace current = graph.getCurrentWorkspace();
String name = current.getName();
Location rootLocation = current.getRoot();
```

16.44. Working with Nodes

Now let's switch to working with nodes. This first example returns a map of properties (keyed by property name) for a node at a specific Path:

```
Path path = ...
Map<Name,Property> propertiesByName = graph.getPropertiesByName().on(path);
```

This next example shows how the graph can be used to obtain and loop over the properties of a node:

```
Path path = ...
for ( Property property : graph.getProperties().on(path) ) {
   ...
}
```

Likewise, the next example shows how the graph can be used to obtain and loop over the children of a node:

```
Path path = ...
for ( Location child : graph.getChildren().of(path) ) {
    Path childPath = child.getPath();
   ...
}
```

Notice that the examples pass a Path instance to the **`on(...)`** and **`of(...)`** methods. Many of the Graph API methods take a variety of parameter types, including String, Paths, Locations, UUID, or Property parameters. This should make it easy to use in many different situations.

Of course, changing content is more interesting and offers more interesting possibilities. Here are a few examples:

```
Path path = ...
Location location = ...
Property idProp1 = ...
Property idProp2 = ...
UUID uuid = ...
graph.move(path).into(idProp1, idProp2);
graph.copy(path).into(location);
graph.delete(uuid);
graph.delete(idProp1,idProp2);
```

The methods shown above work immediately, as soon as each request is built. However, there is another way to use the Graph object, and that is in a batch mode. Simply create a Graph.Batch object using the **`batch()`** method, create the requests on that batch object, and then execute all of the commands on the batch by calling its **`execute()`** method. That **`execute()`** method returns a Results interface that can be used to read the node information retrieved by the batched requests.

Method chaining works really well with the batch mode, since multiple commands can be assembled together very easily:

```
Path path = ...
String path2 = ...
Location location = ...
Property idProp1 = ...
Property idProp2 = ...
UUID uuid = ...
graph.batch().move(path).into(idProp1, idProp2)
       .and().copy(path2).into(location)
       .and().delete(uuid)
       .execute();
Results results = graph.batch().read(path2)
                         .and().readChildren().of(idProp1,idProp2)
                         .and().readSugraphOfDepth(3).at(uuid2)
                         .execute();
for ( Location child : results.getNode(path2) ) {
    ...
}
```

Of course, this section provided just a hint of the Graph API . The Graph interface is actually quite complete and offers a full-featured approach for reading and updating a graph. For more information, see the Graph JavaDocs.

16.45. Requests

ModeShape Graph objects operate upon the underlying graph content, but we haven't really talked about how that works. Recall that the Graph objects don't maintain any stateful representation of the content, but instead submit requests to the underlying graph and return representations of the requested portions of the content. This section focuses on what those requests look like, since they'll actually become very important when working with the connector framework in the next chapter.

A graph Request is an encapsulation of a command that is to be executed by the underlying graph owner (typically a connector). Request objects can take many different forms, as there are different classes for each kind of request. Each request contains the information needed to complete the processing, and it also is the place where the results (or error) are recorded.

The Graph object creates the Request objects using Location objects to identify the node (or nodes) that are the subject of the request. The Graph can either submit the request immediately, or it can batch multiple requests together into "units of work". The submitted requests are then processed by the underlying system (e.g., connector) and returned back to the Graph object, which then extracts and returns the results.

16.46. Read Requests

Table 16.1. Types of Read Requests

Name	Description
ReadNodeRequest	A request to read a node's properties and children from the named workspace in the source. The node may be specified by path and/or by identification properties. The connector returns all properties and the locations for all children, or sets a PathNotFoundException error on the request if the node did not exist in the workspace. If the node is found, the connector sets on the request the actual location of the node (including the path and identification properties). The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.
VerifyNodeExistsRequest	A request to verify the existence of a node at the specified location in the named workspace of the source. The connector returns all the actual location for the node if it exists, or sets a PathNotFoundException error on the request if the node does not exist in the workspace. The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.
ReadAllPropertiesRequest	A request to read all of the properties of a node from the named workspace in the source. The node may be specified by path and/or by identification properties. The connector returns all properties that were found on the node, or sets a PathNotFoundException error on the request if the node did not exist in the workspace. If the node is found, the connector sets on the request the actual location of the node (including the path and identification properties). The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.

Name	Description
ReadPropertyRequest	A request to read a single property of a node from the named workspace in the source. The node may be specified by path and/or by identification properties, and the property is specified by name. The connector returns the property if found on the node, or sets a PathNotFoundException error on the request if the node or property did not exist in the workspace. If the node is found, the connector sets on the request the actual location of the node (including the path and identification properties). The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.
ReadAllChildrenRequest	A request to read all of the children of a node from the named workspace in the source. The node may be specified by path and/or by identification properties. The connector returns an ordered list of locations for each child found on the node, an empty list if the node had no children, or sets a PathNotFoundException error on the request if the node did not exist in the workspace. If the node is found, the connector sets on the request the actual location of the parent node (including the path and identification properties). The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.
ReadBlockOfChildrenRequest	A request to read a block of children of a node, starting with the nth child from the named workspace in the source. This is designed to allow paging through the children, which is much more efficient for large numbers of children. The node may be specified by path and/or by identification properties, and the block is defined by a starting index and a count (i.e., the block size). The connector returns an ordered list of locations for each of the node's children found in the block, or an empty list if there are no children in that range. The connector also sets on the request the actual location of the parent node (including the path and identification properties) or sets a PathNotFoundException error on the request if the parent node did not exist in the workspace. The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.
ReadNextBlockOfChildrenRequest	A request to read a block of children of a node, starting with the children that immediately follow a previously-returned child from the named workspace in the source. This is designed to allow paging through the children, which is much more efficient for large numbers of children. The node may be specified by path and/or by identification properties, and the block is defined by the location of the node immediately preceding the block and a count (i.e., the block size). The connector returns an ordered list of locations for each of the node's children found in the block, or an empty list if there are no children in that range. The connector also sets on the request the actual location of the parent node (including the path and identification properties) or sets a PathNotFoundException error on the request if the parent node did not exist in the workspace. The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.

Name	Description
ReadBranchRequest	A request to read a portion of a subgraph that has as its root a particular node, up to a maximum depth. This request is an efficient mechanism when a branch (or part of a branch) is to be navigated and processed, and replaces some non-trivial code to read the branch iteratively using multiple **`ReadNodeRequest`**s. The connector reads the branch to the specified maximum depth, returning the properties and children for all nodes found in the branch. The connector also sets on the request the actual location of the branch's root node (including the path and identification properties). The connector sets a PathNotFoundException error on the request if the node at the top of the branch does not exist in the workspace. The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.
CompositeRequest	A request comprising multiple requests (none of which will be a composite). The connector processes all of the requests in the composite request, but should set on the composite request any error (usually the first error) that occurs during processing of the contained requests.

16.47. Change Requests

ChangeRequest is a subclass of Request that provides a base class for all the requests that request a change be made to the content. As we'll see later, these ChangeRequest objects also get reused by the graph observation system.

Table 16.2. Types of Change Requests

Name	Description
CreateNodeRequest	A request to create a node at the specified location and setting on the new node the properties included in the request. The connector creates the node at the desired location, adjusting any same-name-sibling indexes as required. (If an SNS index is provided in the new node's location, existing children with the same name after that SNS index will have their SNS indexes adjusted. However, if the requested location does not include a SNS index, the new node is added after all existing children, and its SNS index is set accordingly.) The connector also sets on the request the actual location of the new node (including the path and identification properties).. The connector sets a PathNotFoundException error on the request if the parent node does not exist in the workspace. The connector sets an InvalidWorkspaceException error on the request if the named workspace does not exist.
RemovePropertiesRequest	A request to remove a set of properties on an existing node. The request contains the location of the node as well as the names of the properties to be removed. The connector performs these changes and sets on the request the actual location (including the path and identification properties) of the node. The connector sets a PathNotFoundException error on the request if the node does not exist in the workspace. The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.

Name	Description
UpdatePropertiesRequest	A request to set or update properties on an existing node. The request contains the location of the node as well as the properties to be set and those to be deleted. The connector performs these changes and sets on the request the actual location (including the path and identification properties) of the node. The connector sets a PathNotFoundException error on the request if the node does not exist in the workspace. The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.
RenameNodeRequest	A request to change the name of a node. The connector changes the node's name, adjusts all SNS indexes accordingly, and returns the actual locations (including the path and identification properties) of both the original location and the new location. The connector sets a PathNotFoundException error on the request if the node does not exist in the workspace. The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.
CopyBranchRequest	A request to copy a portion of a subgraph that has as its root a particular node, up to a maximum depth. The request includes the name of the workspace where the original node is located as well as the name of the workspace where the copy is to be placed (these may be the same, but may be different). The connector copies the branch from the original location, up to the specified maximum depth, and places a copy of the node as a child of the new location. The connector also sets on the request the actual location (including the path and identification properties) of the original location as well as the location of the new copy. The connector sets a PathNotFoundException error on the request if the node at the top of the branch does not exist in the workspace. The connector sets a InvalidWorkspaceException error on the request if one of the named workspaces does not exist.
MoveBranchRequest	A request to move a subgraph that has a particular node as its root. The connector moves the branch from the original location and places it as child of the specified new location. The connector also sets on the request the actual location (including the path and identification properties) of the original and new locations. The connector will adjust SNS indexes accordingly. The connector sets a PathNotFoundException error on the request if the node that is to be moved or the new location do not exist in the workspace. The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.
DeleteBranchRequest	A request to delete an entire branch specified by a single node's location. The connector deletes the specified node and all nodes below it, and sets the actual location, including the path and identification properties, of the node that was deleted. The connector sets a PathNotFoundException error on the request if the node being deleted does not exist in the workspace. The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.

16.48. Workspace Read Requests

Table 16.3. Types of Workspace Read Requests

Name	Description
GetWorkspacesRequest	A request to obtain the names of the existing workspaces that are accessible to the caller.
VerifyWorkspaceRequest	A request to verify that a workspace with a particular name exists. The connector returns the actual location for the root node if the workspace exists, as well as the actual name of the workspace (e.g., the default workspace name if a null name is supplied).

16.49. Workspace Change Requests

Requests that deal with changing workspaces extend ChangeRequest.

Table 16.4. Types of Workspace Change Requests

Name	Description
CreateWorkspaceRequest	A request to create a workspace with a particular name. The connector returns the actual location for the root node if the workspace exists, as well as the actual name of the workspace (e.g., the default workspace name if a null name is supplied). The connector sets a InvalidWorkspaceException error on the request if the named workspace already exists.
DestroyWorkspaceRequest	A request to destroy a workspace with a particular name. The connector sets a InvalidWorkspaceException error on the request if the named workspace does not exist.
CloneWorkspaceRequest	A request to clone one named workspace as another new named workspace. The connector sets a InvalidWorkspaceException error on the request if the original workspace does not exist, or if the new workspace already exists.

16.50. Search Requests

Several requests are designed to push searches and queries down to the connector, if connectors support such operations:

Table 16.5. Types of Search Requests

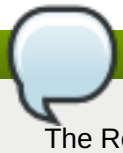
Name	Description
SearchRequest	A request to query a named workspace using a supplied query. The connector returns tuples containing the columns and resulting values, plus statistics about the execution of the query.
FullTextSearchRequest	A request to search a named workspace using a supplied full-text search string and optional offset and limit values. The connector returns tuples containing the columns and resulting values, plus statistics about the execution of the query.

Name	Description
FunctionRequest	A request that executes a supplied function at a particular location within a named workspace. The inputs to the function can be set on the request (as a series of name-value pairs) and when executed, the function will set the outputs as name-value pairs on the request. This request is extremely useful for (complex) operations that must first read information from the workspace and then perform other actions.

16.51. Request Processors

ModeShape connectors are typically the components that receive these Request objects. Before we look further into the connector framework, there is one more component related to Requests that should be discussed.

The RequestProcessor class is an abstract class that defines a **`process(...)`** method for each concrete Request subclass. In other words, there is a **`process(CompositeRequest)`** method, a **`process(ReadNodeRequest)`** method, and so on. This makes it easy to implement behavior that responds to the different kinds of Request classes: subclass the RequestProcessor, override all of the abstract methods, and optionally overriding any of the other methods that have a default implementation.



Note

The RequestProcessor abstract class contains default implementations for quite a few of the **`process(...)`** methods, and these will be sufficient but probably not efficient or optimum. If you can provide a more efficient implementation given your source, feel free to do so. However, if performance is not a big issue, all of the concrete methods will provide the correct behavior. Keep things simple to start out - you can always provide better implementations later.

16.52. Observation Framework

The ModeShape graph model also incorporates an observation framework that allows components to register and be notified when changes occur within the content owned by a graph.

Many event frameworks define the listeners and sources as interfaces. While this is often useful, it requires that the implementations properly address the thread-safe semantics of managing and calling the listeners. The ModeShape observation framework uses abstract or concrete classes to minimize the effort required for implementing ChangeObserver or Observable. These abstract classes provide implementations for a number of utility methods (such as the **`unregister()`** method on ChangeObserver) that also save effort and code.

However, one of the more important reasons for providing classes is that ChangeObserver uses weak references to track the Observable instances, and the ChangeObservers class uses weak references for the listeners. This means that an observer does not prevent Observable instances from being garbage collected, nor do observers prevent Observable instances from being garbage collected. These abstract class provide all this functionality for free.

Each connector is responsible for propagating ChangeRequest objects to the connector's Observer. The sequencing system uses Observers to monitor for changes in the graph content to determine which, if any, sequencers should be run. The JCR implementation also uses the observation framework to propagate those changes to JCR clients.

16.53. Observable Interface

16.53. Observable Interface

Any component that can have changes and be observed can implement the Observable interface. This interface allows Observers to register (or be registered) to receive notifications of the changes. However, a concrete and thread-safe implementation of this interface, called ChangeObservers, is available and should be used where possible, since it automatically manages the registered ChangeObserver instances and properly implements the register and unregister mechanisms.

16.54. Observers

Components that are to receive notifications of changes are called observers. To create an observer, extend the ChangeObserver abstract class and provide an implementation of the **`notify(Changes)`** method. Then, register the observer with an Observable using its **`register(ChangeObserver)`** method. The observer's **`notify(Changes)`** method will then be called with the changes that have been made to the Observable.

When an observer is no longer needed, it should be unregistered from all Observable instances with which it was registered. The ChangeObserver class automatically tracks which Observable instances it is registered with, and calling the observer's **`unregister()`** will unregister the observer from all of these Observables. Alternatively, an observer can be unregistered from a single Observable using the Observable's **`unregister(ChangeObserver)`** method.

16.55. Change Class

The Changes class represents the set of individual changes that have been made during a single, atomic operation. Each Changes instance has information about the source of the changes, the timestamp at which the changes occurred, and the individual changes that were made. These individual changes take the form of ChangeRequest objects, which we'll see more of in the next chapter. Each request is frozen, meaning it is immutable and will not change. Also none of the change requests will be marked as canceled.

Using the actual ChangeRequest objects as the "events" has a number of advantages:

- The existing ChangeRequest subclasses already contain the information to accurately and completely describe the operation. Reusing these classes means we don't need a duplicate class structure or come up with a generic event class.
- The requests have all the state required for an event, plus they often will have more. For example, the DeleteBranchRequest has the actual location of the branch that was deleted (and in this way is not much different than a more generic event), but the CreateNodeRequest has the actual location of the created node along with the properties of that node. Additionally, the RemovePropertyRequest has the actual location of the node along with the name of the property that was removed. In many cases, these requests have all the information a more general event class might have but then hopefully enough information for many observers to use directly without having to read the graph to decide what actually changed.
- The requests that make up a Changes instance can actually be replayed. Consider the case of a cache that is backed by a JcrRepositorySource, which might use an observer to keep the cache in sync. As the cache is notified of Changes, the cache can replay the changes against its source.

16.56. Connectors

With ModeShape, your applications use the JCR 2.0 API to work with the repository, but the ModeShape repository transparently fetches information from different kinds of repositories and storage systems, as opposed to a single, purpose-built store.

At the heart of ModeShape and its JCR implementation is a simple graph-based connector system. Essentially, ModeShape's JCR implementation uses a single connector to access all content.

16.57. Connector Types

A single repository connector could be any one of the following:

- In-Memory Connector – to access a transient, in-memory repository
- JDBC Connector – to access a JDBC database used as a store for repository content
- File System Connector – to access a file system to present its files and directory structure as (updatable) repository content
- Disk Connector – to access data stored in a serialized format on disk

However, it may also be the following special type of connector:

- Federation Connector – to facilitate access of multiple other systems, provding a unified, updatable view of multiple sources (which is coordinated via multiple other connectors)

The federated connector provides many options, since we can use that connector on top of several connectors to other individual sources. This simple connector architecture is fundamentally what makes ModeShape so powerful and flexible.

It is also possible to put a different API layer on top of the connectors. For example, the JSR-203 API allows you to build new file system providers. It would be straightforward to put on top of a JCR implementation, but it could be even simpler by putting it on top of a ModeShape connector. In both cases, it would be a trivial mapping from nodes that represent files and folders into JSR-203 files and directories, with events on those nodes translated into JSR-203 watch events. Then, choose a ModeShape connector and configure it to use the source you want to use.

16.58. Connector Terminology

Connector

A connector is the runnable code packaged in one or more JAR files that contains implementations of several interfaces (described below). A Java developer writes a connector to a type of source, such as a particular database management system, LDAP directory, or source code management system. It is then packaged into one or more JAR files (including dependent JARs) and deployed for use in applications that use ModeShape repositories.

Repository Source

The description of a particular source system (e.g., the "Customer" database, or the company LDAP system) is called a repository source. ModeShape provides a RepositorySource interface providing various features (including a method for establishing connections). A connector will have a class that implements this interface and that has JavaBean properties for all of the connector-specific properties required to describe an instance of the system. Use of JavaBean properties is not required, but it is recommended, as it enables reflective configuration and administration. Applications using ModeShape create an instance of the connector's RepositorySource and set the properties for the external source (that the application wants to access) with that connector.

Repository Connection

A RepositorySource instance is then used to establish connections to that source. A connector provides an implementation of the RepositoryConnection interface, which defines methods for

interacting with the external system. In particular, the **`execute(...)`** method takes an **`ExecutionContext`** instance and a **`Request`** object. The **`ExecutionContext`** object defines the environment in which the processing is occurring, while the **`Request`** object describes the requested operations on the content, with different subclasses representing each type of activity. Examples of commands include getting a node, moving a node, creating a node, changing a node, and deleting a node. If the repository source is able to participate in JTA/JTS distributed transactions, then the RepositoryConnection must implement the **`getXaResource()`** method by returning a valid **`javax.transaction.xa.XAResource`** object that can be used by the transaction monitor.

16.59. Example Use of Connector Components

As an example, consider if we wanted ModeShape to give us access through JCR to the information contained in a relational database. We first have to develop a connector that allows us to interact with relational databases using JDBC. That connector would contain a **`JdbcAccessSource`** class that implements RepositorySource , and that has the various JavaBean properties for setting the name of the driver class, URL, username, password, and other properties. If we add a JavaBean property defining the JNDI name, our connector could look in JNDI to find a JDBC **`DataSource`** instance, perhaps already configured to use connection pools.

Our new connector might also have a **`JdbcAccessConnection`** Java class that implements the RepositoryConnection interface. This class would probably wrap a JDBC database connection, and would implement the **`execute(...)`** method such that the nodes exposed by the connector describe the database tables and their contents. For example, the connector might represent each database table as a node with the table's name, with properties that describe the table (e.g., the description, whether it is a temporary table), and with child nodes that represent rows in the table.

To use our connector in an application that uses ModeShape, we would need to create an instance of the **`JdbcAccessSource`** for each database instance that we want to access. If we have 3 MySQL databases, 9 Oracle databases, and 4 PostgreSQL databases, then we'd need to create a total of 16 **`JdbcAccessSource`** instances, each with the properties describing a single database instance. Those sources are then available for use by ModeShape components, including the JCR implementation.

16.60. Provided Connectors

Before you develop a connector, you should check the list of connectors ModeShape already provides.

16.61. Create a Custom Connector

Creating a custom connector involves the following steps:

1. Implement the RepositorySource interface, using JavaBean properties for each bit of information the implementation will need to establish a connection to the source system. Then, implement the RepositoryConnection interface with a class that represents a connection to the source. The **`execute(ExecutionContext, Request)`** method should process any and all requests that may come down the pike, and the results of each request can be put directly on that request. This approach is pretty straightforward, and gives you ultimate freedom in terms of your class structure.

   Alternatively, an easier way to get a complete read-write connector would be to extend one of our two abstract RepositorySource implementations. If the content your connector exposes has unique keys (such as a unique string, UUID or other identifier), consider implementing **`MapRepositorySource`**, subclassing **`MapRepository`**, and using the existing **`MapRepositoryConnection`** implementation. This **`MapRepositoryConnection`** does most of the work already, relying upon

your **MapRepository** subclass for anything that might be source-specific. (See the JavaDoc for details.) Or, if the content your connector exposes is path-based, consider implementing **PathRepositorySource**, subclassing **PathRepository**, and using the existing **PathRepositoryConnection** implementation. Again, **PathRepositoryConnection** class does almost all of the work and delegates to your **PathRepository** subclass for anything that might be source-specific. (See the JavaDoc for details.)

Don't forget unit tests that verify the connector is doing what it is expected to do. (If you'll be committing the connector code to the ModeShape project, please ensure that the unit tests can be run by others that may not have access to the source system. In this case, consider writing integration tests that can be easily configured to use different sources in different environments, and try to make the failure messages clear when the tests can't connect to the underlying source.)

2. Configure ModeShape to use your connector. This may involve just registering the source with the **RepositoryService**, or it may involve adding a source to a configuration repository used by the federated repository.

3. Deploy the JAR file with your connector (as well as any dependencies), and make them available to ModeShape in your application.

16.62. Implementing a Repository Source

Perhaps the most important class that makes up a connector is the implementation of the RepositorySource . This class is analogous to JDBC's DataSource in that it is instantiated to represent a single instance of a system that will be accessed, and it contains enough information (in the form of JavaBean properties) so that it can create connections to the source.

Why is the RepositorySource implementation a JavaBean? Well, this is the class that is instantiated, usually reflectively, and so a no-arg constructor is required. Using JavaBean properties makes it possible to reflect upon the object's class to determine the properties that can be set (using setters) and read (using getters). This means that an administrative application can instantiate, configure, and manage the objects that represent the actual sources, without having to know anything about the actual implementation.

So, your connector will need a public class that implements RepositorySource and provides JavaBean properties for any kind of inputs or options required to establish a connection to and interact with the underlying source. Most of the semantics of the class are defined by the RepositorySource and inherited interface.

16.63. Implementing a Repository Connection

One job of the RepositorySource is to create connections to the underlying sources. Connections are represented by classes that implement the RepositoryConnection interface, and creating this class is the next step in writing a connector. This is what we'll cover in this section.

16.64. RepositoryConnection Interface

```
/**
 * A connection to a repository source.
 *
 * These connections need not support concurrent operations by multiple
threads.
 */
@NotThreadSafe
public interface RepositoryConnection {
```

```
    /**
     * Get the name for this repository source. This value should be the
same as that returned
     * by the same RepositorySource that created this connection.
     *
     * @return the identifier; never null or empty
     */
    String getSourceName();

    /**
     * Return the transactional resource associated with this connection.
The transaction manager
     * will use this resource to manage the participation of this connection
in a distributed transaction.
     *
     * @return the XA resource, or null if this connection is not aware of
distributed transactions
     */
    XAResource getXAResource();

    /**
     * Ping the underlying system to determine if the connection is still
valid and alive.
     *
     * @param time the length of time to wait before timing out
     * @param unit the time unit to use; may not be null
     * @return true if this connection is still valid and can still be used,
or false otherwise
     * @throws InterruptedException if the thread has been interrupted
during the operation
     */
    boolean ping( long time, TimeUnit unit ) throws InterruptedException;

    /**
     * Get the default cache policy for this repository. If none is
provided, a global cache policy
     * will be used.
     *
     * @return the default cache policy
     */
    CachePolicy getDefaultCachePolicy();

    /**
     * Execute the supplied commands against this repository source.
     *
     * @param context the environment in which the commands are being
executed; never null
     * @param request the request to be executed; never null
     * @throws RepositorySourceException if there is a problem loading the
node data
     */
    void execute( ExecutionContext context, Request request ) throws
RepositorySourceException;

    /**
```

```
     * Close this connection to signal that it is no longer needed and that
any accumulated
     * resources are to be released.
     */
    void close();
}
```

While most of these methods are straightforward, a few warrant additional information. The **`ping(...)`** method allows ModeShape to check the connection to see if it is alive. This method can be used in a variety of situations, ranging from verifying that a RepositorySource 's JavaBean properties are correct to ensuring that a connection is still alive before returning the connection from a connection pool.

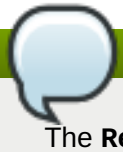
The most important method on this interface, though, is the **`execute(...)`** method, which serves as the mechanism by which the component using the connector access and manipulates the content exposed by the connector. The first parameter to this method is the **`ExecutionContext`**, which contains the information about environment as well as the subject performing the request. This was discussed earlier.

The second parameter, however, represents a **`Request`** that is to be processed by the connector. **`Request`** objects can take many different forms, as there are different classes for each kind of request (see the previous chapter for details). Each request contains the information a connector needs to do the processing, and it also is the place where the connector places the results (or the error, if one occurs).

A connector is technically free to implement the **`execute(...)`** method in any way, as long as the semantics are maintained. But as discussed in the previous chapter, ModeShape provides a **`RequestProcessor`** class that can simplify writing your own connector and at the same time help insulate your connector from new kinds of requests that may be added in the future. The **`RequestProcessor`** is an abstract class that defines a **`process(...)`** method for each concrete Request subclass. In other words, there is a **`process(CompositeRequest)`** method, a **`process(ReadNodeRequest)`** method, and so on.

16.65. Using a Request Processor

Create a subclass of **`RequestProcessor`**, overriding all of the abstract methods and optionally overriding any of the other methods that have a default implementation.



Note

The **`RequestProcessor`** abstract class contains default implementations for quite a few of the **`process(...)`** methods, and these will be sufficient but probably not efficient or optimum. If you can provide a more efficient implementation given your source, feel free to do so. However, if performance is not a big issue, all of the concrete methods will provide the correct behavior. Keep things simple to start out - you can always provide better implementations later.

Also, make sure your **`RequestProcessor`** is properly broadcasting the changes made during execution. The **`RequestProcessor`** class has a **`recordChange(ChangeRequest)`** method that can be called from each of the **`process(...)`** methods that take a **`ChangeRequest`**. The **`RequestProcessor`** enqueues these requests, and when the **`RequestProcessor`** is closed, the default implementation is to send a **`Changes`** to the Observer supplied into the constructor.

Then, in your connector's **`execute(ExecutionContext, Request)`** method, instantiate your **`RequestProcessor`** subclass and call its **`process(Request)`** method, passing in the **`execute(...)`** method's **`Request`** parameter. The **`RequestProcessor`** will determine the appropriate method given the actual **`Request`** object and will then invoke that method:

```
public void execute( final ExecutionContext context,
                     final Request request ) throws
RepositorySourceException {
    String sourceName = // from the RepositorySource
    Observer observer = // from the RepositoryContext
    RequestProcessor processor = new
CustomRequestProcessor(sourceName,context,observer);
    try {
        processor.process(request);
    } finally {
        processor.close(); // sends the accumulated ChangeRequests as a
Changes to the Observer
    }
}
}
```

If you do this, the bulk of your connector implementation may be in the **`RequestProcessor`** implementation methods. This not only is pretty maintainable, it also lends itself to easier testing. And should any new request types be added in the future, your connector may work just fine without any changes. In fact, if the **`RequestProcessor`** class can implement meaningful methods for those new request types, your connector may "just work". Or, at least your connector will still be binary compatible, even if your connector won't support any of the new features.

Finally, how should the connector handle exceptions? As mentioned above, each **`Request`** object has a slot where the connector can set any exception encountered during processing. This not only handles the exception, but in the case of **`CompositeRequest`**s it also correctly associates the problem with the request. However, it is perfectly acceptable to throw an exception if the connection becomes invalid (e.g., there is a communication failure) or if a fatal error would prevent subsequent requests from being processed.

16.66. Broadcasting Events

When your RepositorySource instance is put into the library within a running ModeShape system, the **`initialize( RepositoryContext )`** method will be called on the instance. The supplied RepositoryContext object represents the context in which the RepositorySource is running, and provides access to an **`ExecutionContext`**, a RepositoryConnectionFactory that can be used to obtain connections to other sources, and an Observer of your source that should be called with events describing the **`Changes`** being made within the source, either as a result of **`ChangeRequest`** operations being performed on this source, or as a result of operations being performed on the content from outside the source.

16.67. Cache Policy

Each connector is responsible for determining whether and how long ModeShape is to cache the content made available by the connector. This is referred to as the caching policy, and consists of a time to live value representing the number of milliseconds that a piece of data may be cached. After the TTL has passed, the information is no longer used.

ModeShape allows a connector to use a flexible and powerful caching policy. First, each connection returns the default caching policy for all information returned by that connection. Often this policy can be configured via properties on the RepositorySource implementation. This is optional, meaning the connector can return **`null`** if it does not wish to have a default caching policy.

Second, the connector is able to override its default caching policy on individual requests. Again, this is optional, meaning that a null caching policy on a request implies that the request has no overridden caching policy.

Third, if the connector has no default caching policy and none is set on the individual requests, ModeShape uses whatever caching policy is set up for that component using the connector. For example, the federating connector allows a default caching policy to be specified, and this policy is used should the sources being federated not define their own caching policy.

In summary, a connector has total control over whether and for how long the information it provides is cached.

Note

At this time, not every connector takes advantage of cache policies. However, it is anticipated that this will change.

16.68. Leveraging JNDI

Sometimes it is necessary (or easier) for a RepositorySource implementation to look up an object in JNDI. One example of this is the JBoss Cache connector: while the connector can instantiate a new JBoss Cache instance, more interesting use cases involve JBoss Cache instances that are set up for clustering and replication, something that is generally difficult to configure in a single JavaBean. Therefore the **`JBossCacheSource`** has optional JavaBean properties that define how it is to look up a JBoss Cache instance in JNDI.

This is a simple pattern that you may find useful in your connector. Basically, if your source implementation can look up an object in JNDI, use a single JavaBean String property that defines the full name that should be used to locate that object in JNDI. Usually it is best to include "Jndi" in the JavaBean property name so that administrative users understand the purpose of the property. (And some may suggest that any optional property also use the word "optional" in the property name.)

16.69. Capabilities

Another characteristic of a RepositorySource implementation is that it provides some hint as to whether it supports several features. This is defined on the interface as a method that returns a **`RepositorySourceCapabilities`** object. This class currently provides methods that say whether the connector supports updates, whether it supports same-name-siblings (SNS), and whether the connector supports listeners and events.

Note that these may be hard-coded values, or the connector's response may be determined at runtime by various factors. For example, a connector may interrogate the underlying system to decide whether it can support updates.

The **`RepositorySourceCapabilities`** class can be used as is (the class is immutable), or it can be subclassed to provide more complex behavior. It is important, however, that the capabilities remain constant throughout the lifetime of the RepositorySource instance.

Note

Why a concrete class and not an interface? By using a concrete class, connectors inherit the default behavior. If additional capabilities need to be added to the class in future releases, connectors may not have to override the defaults. This provides some insulation against future enhancements to the connector framework.

16.70. Security and Authentication

As we'll see in the next section, the main method that connectors use to process requests takes an **`ExecutionContext`**, which contains the JAAS security information of the subject performing the request. This means that the connector can use this to determine authentication and authorization information for each request.

Sometimes that is not sufficient. For example, it may be that the connector needs its own authorization information so that it can establish a connection (even if user-level privileges still use the **`ExecutionContext`** provided with each request). In this case, the RepositorySource implementation will probably need JavaBean properties that represent the connector's authentication information. This may take the form of a username and password, or it may be properties that are used to delegate authentication to JAAS . Either way, just realize that it is perfectly acceptable for the connector to require its own security properties.

16.71. ModeShape Sequencing

Many repositories are used (at least in part) to manage files and other artifacts, including service definitions, policy files, images, media, documents, presentations, application components, reusable libraries, configuration files, application installations, databases schemas, management scripts, and so on. Unlocking the information buried within all of those files is what ModeShape sequencing is all about. As files are loaded into the repository, you ModeShape instance can automatically sequence these files to extract from their content meaningful information that can be stored in the repository, where it can then be searched, accessed, and analyzed using the JCR API.

16.72. Sequencers

Sequencers are POJOs that implement a specific interface, and their job is to process a stream of data (supplied by ModeShape) to extract meaningful content that usually takes the form of a structured graph. Exactly what content is up to each sequencer implementation. For example, ModeShape comes with a Compact Node Definition (CND) sequencer that processes the CND files to extract and produce a structured representation of the node type definitions, property definitions, and child node definitions contained within the file.

Sequencers are configured to identify the kinds of nodes that the sequencers can work against. When content in the repository changes, ModeShape looks to see which (if any) sequencers might be able to run on the changed content. If any sequencer configurations do match, those sequencers are run against the content, and the structured graph output of the sequencers is then written back into the repository (at a location dictated by the sequencer configuration). And once that information is in the repository, it can be easily found and accessed via the standard JCR API.

In other words, ModeShape uses sequencers to help you extract more meaning from the artifacts you already are managing, and makes it much easier for applications to find and use all that valuable information. All without your applications doing anything extra.

16.73. Stream Sequencers

The StreamSequencer interface defines the single method that must be implemented by a sequencer:

```
public interface StreamSequencer {

    /**
     * Sequence the data found in the supplied stream, placing the output
```

```
     * information into the supplied map.
     *
     * @param stream the stream with the data to be sequenced; never null
     * @param output the output from the sequencing operation; never null
     * @param context the context for the sequencing operation; never null
     */
    void sequence( InputStream stream, SequencerOutput output,
StreamSequencerContext context );
}
```

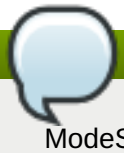
A new instance is created for each sequencing operation, so there is no need for the class to be synchronized or thread-safe. Additionally, when a sequencer configuration includes properties, ModeShape will set those properties on the StreamSequencer implementation using JavaBean-style setter methods. This makes it easy to define sequencer-specific properties on the sequencer configurations, while making it easy to implement with JavaBean-style setter methods.

Implementations are responsible for processing the content in the supplied InputStream content and generating structured content using the supplied SequencerOutput interface. The StreamSequencerContext provides additional details about the information that is being sequenced, including the location and properties of the node being sequenced, the MIME type of the node being sequenced, and a Problems object where the sequencer can record problems that aren't severe enough to warrant throwing an exception. The StreamSequencerContext also provides access to the ValueFactories that can be used to create Path, Name, and any other value objects.

The SequencerOutput interface is fairly easy to use, and its job is to hide from the sequencer all the specifics about where the output is being written. Therefore, the interface has only a few methods for implementations to call. Two methods set the property values on a node, while the other sets references to other nodes in the repository. Use these methods to describe the properties of the nodes you want to create, using relative paths for the nodes and valid JCR property names for properties and references. ModeShape will ensure that nodes are created or updated whenever they're needed.

```
public interface SequencerOutput {

  /**
   * Set the supplied property on the supplied node.  The allowable
   * values are any of the following:
   *   - primitives (which will be autoboxed)
   *   - String instances
   *   - String arrays
   *   - byte arrays
   *   - InputStream instances
   *   - Calendar instances
   *
   * @param nodePath the path to the node containing the property;
   * may not be null
   * @param property the name of the property to be set
   * @param values the value(s) for the property; may be empty if
   * any existing property is to be removed
   */
  void setProperty( String nodePath, String property, Object... values );
  void setProperty( Path nodePath, Name property, Object... values );

  /**
   * Set the supplied reference on the supplied node.
   *
   * @param nodePath the path to the node containing the property;
```

```
     * may not be null
     * @param property the name of the property to be set
     * @param paths the paths to the referenced property, which may be
     * absolute paths or relative to the sequencer output node;
     * may be empty if any existing property is to be removed
     */
    void setReference( String nodePath, String property, String... paths );
}
```



Note

ModeShape will create nodes of type **`nt:unstructured`** unless you specify the value for the **`jcr:primaryType`** property. You can also specify the values for the **`jcr:mixinTypes`** property if you want to add mixins to any node.

16.74. Path Expressions

Each sequencer must be configured to describe the areas or types of content that the sequencer is capable of handling. This is done by specifying these patterns using path expressions that identify the nodes (or node patterns) that should be sequenced and where to store the output generated by the sequencer. We'll see how to fully configure a sequencer in the next chapter, but before then let's dive into path expressions in more detail.

A path expression consist of two parts: a selection criteria (or an input path) and an output path:

```
    inputPath => outputPath
```

The inputPath part defines an expression for the path of a node that is to be sequenced. Input paths consist o '**`/`**' separated segments, where each segment represents a pattern for a single node's name (including the same-name-sibling indexes) and '@' signifies a property name.

16.75. Simple Input Path Examples

Table 16.6. Simple Input Path Examples

Input Path	Description
/a/b	Match node "**b**" that is a child of the top level node "**a**". Neither node may have any same-name-sibilings.
/a/*	Match any child node of the top level node "**a**".
/a/*.txt	Match any child node of the top level node "**a**" that also has a name ending in "**`.txt`**".
/a/*.txt	Match any child node of the top level node "**a**" that also has a name ending in "**`.txt`**".
/a/b@c	Match the property "**c**" of node "**`/a/b`**".
/a/b[2]	The second child named "**b**" below the top level node "**a**".
/a/b[2,3,4]	The second, third or fourth child named "**b**" below the top level node "**a**".
/a/b[*]	Any (and every) child named "**b**" below the top level node "**a**".

Input Path	Description
//a/b	Any node named "**b**" that exists below a node named "**a**", regardless of where node "**a**" occurs. Again, neither node may have any same-name-sibilings.

With these simple examples, you can probably discern the most important rules.

First, the '**`*`**' is a wildcard character that matches any character or sequence of characters in a node's name (or index if appearing in between square brackets), and can be used in conjunction with other characters (e.g., "**`*.txt`**").

Second, square brackets (i.e., '**`[`**' and '**`]`**') are used to match a node's same-name-sibiling index. You can put a single non-negative number or a comma-separated list of non-negative numbers. Use '0' to match a node that has no same-name-sibilings, or any positive number to match the specific same-name-sibling.

Third, combining two delimiters (e.g., "**`//`**") matches any sequence of nodes, regardless of what their names are or how many nodes. Often used with other patterns to identify nodes at any level matching other patterns. Three or more sequential slash characters are treated as two.

Many input paths can be created using these simple rules. However, input paths can be more complicated.

16.76. Advanced Input Path Examples

Table 16.7. Advanced Input Path Examples

Input Path	Description
/a/(b|c|d)	Match children of the top level node "**a**" that are named "**b**", "**c**" or "**d**". None of the nodes may have same-name-sibling indexes.
/a/b[c/d]	Match node "**b**" child of the top level node "**a**", when node "**b**" has a child named "**c**", and "**c**" has a child named "**d**". Node "**b**" is the selected node, while nodes "**c**" and "**d**" are used as criteria but are not selected.
/a(/(b|c|d|)/e)[f/g/@something]	Match node "**`/a/b/e`**", "**`/a/c/e`**", "**`/a/d/e`**", or "**`/a/e`**" when they also have a child "**`f`**" that itself has a child "**g**" with property "**`something`**". None of the nodes may have same-name-sibling indexes.

These examples show a few more advanced rules. Parentheses (i.e., '**`(`**' and '**`)`**') can be used to define a set of options for names, as shown in the first and third rules. Whatever part of the selected node's path appears between the parentheses is captured for use within the output path. Thus, the first input path in the previous table would match node "**`/a/b`**", and "b" would be captured and could be used within the output path using "**`$1`**", where the number used in the output path identifies the parentheses.

Square brackets can also be used to specify criteria on a node's properties or children. Whatever appears in between the square brackets does not appear in the selected node.

16.77. Input Paths with Source and Workspace Names

There are times when it is desirable to configure sequencers to only work against content in a specific source and/or specific workspace. In these cases, it is possible to specify the repository name and workspace names before the path.

Table 16.8. Input Paths with Source and Workspace Names

Input Path	Description
source:default:/a/(b|c|d)	Match nodes in the "**`default`**" workspace within the "**`source`**" source that are children of the top level node "**a**" and named "**b**", "**c**" or "**d**". None of the nodes may have same-name-sibling indexes.
:default:/a/(b|c|d)	Match nodes in the "**`default`**" workspace within any source source that are children of the top level node "**a**" and named "**b**", "**c**" or "**d**". None of the nodes may have same-name-sibling indexes.
source::/a/(b|c|d)	Match nodes in any workspace in the "**`source`**" source that are children of the top level node "**a**" and named "**b**", "**c**" or "**d**". None of the nodes may have same-name-sibling indexes.
::/a/(b|c|d)	Match nodes in any within any source source that are children of the top level node "**a**" and named "**b**", "**c**" or "**d**". None of the nodes may have same-name-sibling indexes. (This is equivalent to the path "**`/a/(b|c|d)`**".)

Again, the rules are pretty straightforward. You can leave off the repository name and workspace name, or you can prepend the path with "**`{sourceNamePattern}:{workspaceNamePattern}:`**", where "**`{sourceNamePattern}`** is a regular-expression pattern used to match the applicable source names, and "**`{workspaceNamePattern}`** is a regular-expression pattern used to match the applicable workspace names. A blank pattern implies any match, and is a shorthand notation for "**`.*`**". Note that the repository names may not include forward slashes (e.g., '/') or colons (e.g., ':').

Consider the following code fragment:

```
//(*.(jpg|jpeg|gif|bmp|pcx|png)[*])/jcr:content[@jcr:data] => /images/$1
```

This matches a node named "**`jcr:content`**" with property "**`jcr:data`**" but no siblings with the same name, and that is a child of a node whose name ends with "**`.jpg`**", "**`.jpeg`**", "**`.gif`**", "**`.bmp`**", "**`.pcx`**", or "**`.png`**" that may have any same-name-sibling index. These nodes can appear at any level in the repository. Note how the input path capture the filename (the segment containing the file extension), including any same-name-sibling index. This filename is then used in the output path, which is where the sequenced content is placed.

16.78. Creating Custom Sequencers

Creating a custom sequencer involves the following steps:

1. Implement the StreamSequencer interface with your own implementation, and create unit tests to verify the functionality and expected behavior;
2. Add the sequencer configuration to the ModeShape **`SequencingService`** in your application and
3. Deploy the JAR file with your implementation (as well as any dependencies), and make them available to ModeShape in your application.

Chapter 17. Using ModeShape

17.1. Using ModeShape Within Your Application

Using ModeShape within your application is actually quite straightforward, and with JCR 2.0 it is possible for your application to do everything using only the JCR 2.0 API . Your application will first obtain a **`javax.jcr.Repository`** instance, and will use that object to create sessions through which your application will read, modify, search, or monitor content in the repository.

17.2. ModeShape Configuration Options

The following options are available for configuring ModeShape:

- **`Loading configuration from a file`** is conceptually the most straightforward and requires the least amount of Java code, but it does requires having a configuration file. This is easy, allows one to manage configurations in version control, enables your application to use only the standard JCR API , and will likely be the best approach for most applications. If you're not sure, use this approach.
- **`Loading configuration from a repository`** is an advanced option allowing multiple **`JcrEngine`** instances (usually in different processes perhaps on different machines) to easily access a (shared) configuration.

Each of these approaches has different advantages.

17.3. Loading Your Configuration from a File

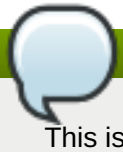
The **`modeshape-config.xml`** file in ***`SOA_ROOT`*`/jboss-as/server/`*`PROFILE`*`/deploy/modeshape-services.jar`** is used by default.

Here is an example configuration file used in the repository example covered in the *ModeShape Getting Started Guide* document, though it has been simplified for clarity:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:mode="http://www.modeshape.org/1.0"
xmlns:jcr="http://www.jcp.org/jcr/1.0">
  <!--
  Define the JCR repositories
  -->
  <mode:repositories>
      <!--
      Define a JCR repository that accesses the 'Cars' source directly.
      This of course is optional, since we could access the same content
through 'vehicles'.
      -->
      <mode:repository jcr:name="car repository" mode:source="Cars">
          <mode:options jcr:primaryType="mode:options">
              <mode:option jcr:name="jaasLoginConfigName"
mode:value="modeshape-jcr"/>
          </mode:options>
          <mode:descriptors>
            <!--
              This adds a JCR Repository descriptor named "myDescriptor" with
a value of "foo".
```

```
             So this code:
             Repository repo = ...;
             System.out.println(repo.getDescriptor("myDescriptor");

             Will now print out "foo".
            -->
            <myDescriptor mode:value="foo" />
          </mode:descriptors>
          <!--
                Import the custom node types defined in the named files. The
values
                can be an absolute path to a classpath resource, an absolute
file system
                path, a relative path on the file system (relative to where
the process was
                started from), or a resolvable URL. If more than one node
type definition
                file is needed, the files can be listed as a single comma-
delimited string
                in the 'mode:resource' attribute of the 'jcr:nodeTypes'
element, or listed
                individually using multiple mode:resource child elements (as
shown below).
            -->
          <jcr:nodeTypes>
            <mode:resource>/org/example/my-node-types.cnd</mode:resource>
            <mode:resource>/org/example/additional-node-
types.cnd</mode:resource>
          </jcr:nodeTypes>
      </mode:repository>
  </mode:repositories>
   <!--
   Define the sources for the content. These sources are directly accessible
using the
   ModeShape-specific Graph API.
   -->
   <mode:sources jcr:primaryType="nt:unstructured">
       <mode:source jcr:name="Cars"

mode:classname="org.modeshape.graph.connector.inmemory.InMemoryRepositorySou
rce"
             mode:retryLimit="3" mode:defaultWorkspaceName="workspace1">

<mode:predefinedWorkspaceNames>workspace2</mode:predefinedWorkspaceNames>

<mode:predefinedWorkspaceNames>workspace3</mode:predefinedWorkspaceNames>
       </mode:source>
   </mode:sources>
   <!--
   Define the sequencers. This is an optional section. For this example,
we're not using any sequencers.
   -->
   <mode:sequencers>
       <!--mode:sequencer jcr:name="Image Sequencer">
           <mode:classname>
            org.modeshape.sequencer.image.ImageMetadataSequencer
```

```
            </mode:classname>
            <mode:description>Image metadata sequencer</mode:description>
            <mode:pathExpression>/foo/source =>
/foo/target</mode:pathExpression>
            <mode:pathExpression>/bar/source =>
/bar/target</mode:pathExpression>
        </mode:sequencer-->
    </mode:sequencers>
    <mode:mimeTypeDetectors>
        <mode:mimeTypeDetector jcr:name="Detector"
                               mode:description="Standard extension-based
MIME type detector"/>
    </mode:mimeTypeDetectors>
</configuration>
```



Note

This is the recommended approach if your application uses the standard and implementation-independent RepositoryFactory mechanism to obtain the JCR Repository reference.

17.4. Loading Your Configuration from a Repository

The first step is to create and configure the RepositorySource instance that we'll use to access the repository where the configuration is stored. Then, create a **`JcrConfiguration`** instance and load from this source:

```
RepositorySource configSource = ...
JcrConfiguration config = new JcrConfiguration();
configuration.loadFrom(configSource);
```

The **`loadFrom(...)`** method can be called any number of times, but each time it is called it completely wipes out any current notion of the configuration and replaces it with the configuration found in the file.

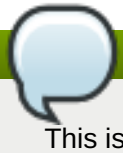
There is an optional second parameter that defines the name of the workspace in the supplied source where the configuration content can be found. It is not needed if the workspace is the source's default workspace. There is an optional third parameter that defines the Path within the configuration repository identifying the parent node of the various configuration nodes. If not specified, it assumes "/". This makes it possible for the configuration content to be located at a different location in the hierarchical structure. (This is not often required, but it is very useful if you ModeShape configuration file is embedded within another XML file.)

Once the **`JcrConfiguration`** has been loaded from a RepositorySource , the **`JcrConfiguration`** instance can be used to modify the configuration and then save those changes back to the repository. This technique can be used to place a configuration into a repository (such as a database) for the first time:

```
RepositorySource configSource = ... // a RepositorySource to an empty source
JcrConfiguration config = new JcrConfiguration();

// Bind the configuration to the repository source (which is initially
empty)...
configuration.loadFrom(configSource);

// Now load a configuration from a file (or construct one programmatically)
...
```

```
String pathToFile = ...
configuration.loadFrom(pathToFile);

// Now save the configuration into the source ...
configuration.save();
```

Now you can load this configuration in multiple processes, using the approach mentioned above.



Note

This is an advanced way of defining your configuration, so this is recommended only for those that are already very comfortable with ModeShape and its lower-level graph API and connector API.

17.5. JCR Repository Options

ModeShape JCR repositories have a number of behaviors that can be controlled from within the configuration. These are known as repository options, and all have sensible defaults. However, they do allow you to better configure the JCR repository instances to best suit your needs.

As mentioned earlier, these options can be set programmatically or within the configuration file. When setting up the configuration programmatically, the actual enum literal values must be used, and all values are String literals:

```
JcrConfiguration config = ...
config.repository("repository A")
     .setOption(JcrRepository.Option.JAAS_LOGIN_CONFIG_NAME, "modeshape-
jcr");
```

When using a configuration file, you set the option within the "**`mode:options`**" fragment under the "**`mode:repository`**" section. Each option fragment looks similar to the following:

```
<mode:option jcr:name="jaasLoginConfigName" mode:value="modeshape-jcr"/>
```

where the "**`jcr:name`**" XML attribute value contains the lower-camel-case form of the option literal, and the "**`mode:value`**" XML attribute value contains the repository option value. In the example above, the "jaasLoginConfigName" is the option name, and "modeshape-jcr" is the option value. An alternative representation is to set the name using the XML element name and set the primary type with an XML attribute. Thus, this fragment is equivalent to the previous listing:

```
<jaasLoginConfigName jcr:primaryType="mode:option" mode:value="modeshape-
jcr"/>
```

The following table describes all of the current repository options.

Table 17.1. JCR Repository Options

Option	Description

Option	Description
jaasLoginConfigName	The JAAS JAAS application configuration name that specifies which login module should be used to validate credentials. By default, "modeshape-jcr" is used. Set the option with an empty (zero-length) value to completely turn off JAAS authentication. The enumeration literal is **`Option.JAAS_LOGIN_CONFIG_NAME`**.
systemSourceName	The name of the source (and optionally the workspace in the source) where the "/jcr:system" branch should be stored. The format is "name of workspace@name of source", or "name of source" if the default workspace is to be used. If this option is not used, a transient in-memory source will be used. Note that all leading and trailing whitespaces is removed for both the source name and workspace name. Thus, a value of "@" implies a zero-length workspace name and zero-length source name. Also, any use of the '@' character in source and workspace names must be escaped with a preceding backslash. The enumeration literal is **`Option.SYSTEM_SOURCE_NAME`**.
anonymousUserRoles	A comma-delimited list of default roles provided for anonymous access. A null or empty value for this option means that anonymous access is disabled. The enumeration literal is **`Option.ANONYMOUS_USER_ROLES`**.
exposeWorkspaceNamesInDescription	A boolean flag that indicates whether a complete list of workspace names should be exposed in the custom repository descriptor "**`org.modeshape.jcr.api.Repository.REPOSITORY_WORKSPACES`**". If this option is set to true, then any code that can access the repository can retrieve a complete list of workspace names through the javax.jcr.Repository.getDescriptor(String) method without logging in. The default value is 'true', meaning that the descriptor is populated. Since some ModeShape installations may consider the list of workspace names to be restricted information and limit the ability of some or all users to see a complete list of workspace names, this option can be set to "false" to disable this capability. If this option is set to "false", the "**`org.modeshape.jcr.api.Repository.REPOSITORY_WORKSPACES`**" descriptor will not be set. The enumeration literal is **`Option.EXPOSE_WORKSPACE_NAMES_IN_DESCRIPTOR`**.
repositoryJndiLocation	A string property that when specified tells the **`JcrEngine`** where to put the Repository in JNDI. Assumes that you have write access to the JNDI tree. If no value set, then the Repository will not be bound to JNDI. The enumeration literal is **`Option.REPOSITORY_JNDI_LOCATION`**.

Option	Description
queryExecutionEnabled	A boolean flag that specifies whether this repository is expected to execute searches and queries. If client applications will never perform searches or queries, then maintaining the query indexes is an unnecessary overhead, and can be disabled. Note that this is merely a hint, and that searches and queries might still work when this is set to 'false'. The default is 'true', meaning that clients can execute searches and queries. The enumeration literal is **`Option.QUERY_EXECUTION_ENABLED`**.
queryIndexDirectory	The system may maintain a set of indexes that improve the performance of searching and querying the content. These size of these indexes depend upon the size of the content being stored, and thus may consume a significant amount of space. This option defines a location on the file system where this repository may (if needed) store indexes so they don't consume large amounts of memory. If specified, the value must be a valid path to a writable directory on the file system. If the path specifies a non-existant location, the repository may attempt to create the missing directories. The path may be absolute or relative to the location where this VM was started. If the specified location is not a readable and writable directory (or cannot be created as such), then this will generate an exception when the repository is created. The default value is null, meaning the search indexes may not be stored on the local file system and, if needed, will be stored within memory. The enumeration literal is **`Option.QUERY_INDEX_DIRECTORY`**.

Option	Description
queryIndexesUpdatedSynchronously	An advanced boolean flag that specifies whether updates to the indexes (if used) should be made synchronously, meaning that a call to Session.save() will not return until the search indexes have been completely updated. The benefit of synchronous updates is that a search or query performed immediately after a save() will operate upon content that was just changed. The downside is that the save() operation will take longer. With asynchronous updates, however, the only work done during a save() invocation is that required to persist the changes in the underlying repository source, while changes to the search indexes are made in a different thread that may not run immediately. In this case, there may be an indeterminate lag before searching or querying after a save() will operate upon the changed content. The default is value 'false', meaning the updates are performed asynchronously. The enumeration literal is **`Option.QUERY_INDEXES_UPDATED_SYNCHRONOUSLY`**.
queryIndexesRebuiltSynchronously	An advanced boolean flag that specifies whether the indexes should be rebuilt synchronously when the repository restarts. If this flag is set to 'true', query indexes for each workspace in the repository will be rebuilt synchronously the first time that the repository is accessed (e.g., at the first login). If this flag is set to 'false', the query indexes for each workspace in the repository will be rebuilt asynchronously. Rebuilding the indexes synchronously can cause very significant latency in the initial repository access if the repository contains a significant amount of content that must be reindexed. Updating the indexes asynchronously eliminates this latency, but repository queries may generate inconsistent results while the indexes are being updated. That is, query results may refer to content that is no longer in the repository or may fail to include appropriate results for nodes that had been added to the repository. The default is value 'true', meaning the rebuilds are performed synchronously. The enumeration literal is **`Option.QUERY_INDEXES_REBUILT_SYNCHRONOUSLY`**.

Option	Description
rebuildQueryIndexOnStartup	An advanced setting that specifies the strategy used to determine which query indexes need to be rebuilt when the repository restarts. ModeShape currently supports two strategies: - A value of "always" dictates that the query index for every workspace in the repository will be rebuilt each time that the repository restarts. This can sharply increase the startup time for the repository, particularly if the **`queryIndexesRebuiltSynchronously`** option is set to 'true' (the default). However, this strategy ensures that any repository content that was modified outside of the repository (e.g., files in a **`FileSystemSource`** that were directly modified on the file system) are properly indexed. - A value of "ifMissing" indicates that indexes should only be rebuilt if they do not currently exist or are obviously invalid. This strategy is always the most appropriate strategy for non-clustered repositories with repository sources that provide exclusive control over content (e.g., the **`InfinispanSource`**, the **`JpaSource`**) as it greatly reduces repository startup time for repositories with significant amounts of content. Note that repositories that do not configure the **`queryIndexDirectory`** option will always use an in-memory index. This type of index will not be persisted across repository restarts and will require ModeShape to rebuild the indexes each time the repository starts up even if the "ifMissing" strategy is specified. The "always" strategy is used by default and in cases where the option's value does not case-independently match the one of these two values. This was the only strategy available prior to ModeShape 2.8.1.GA. The enumeration literal is **`Option.QUERY_INDEXES_REBUILT_SYNCHRONOUSLY`**, and the values are **`RebuildQueryIndexOnStartupOption.ALWAYS`** and **`RebuildQueryIndexOnStartupOption.IF_MISSING`**.
projectNodeTypes	An advanced boolean flag that defines whether or not the node types should be exposed as content under the "/jcr:system/jcr:nodeTypes" node. Value is either "true" or "false" (default). The enumeration literal is **`Option.PROJECT_NODE_TYPES`**.
readDepth	An advanced integer flag that specifies the depth of the subgraphs that should be loaded from the connectors during normal read operations. The default value is 1. The enumeration literal is **`Option.READ_DEPTH`**.
indexReadDepth	An advanced integer flag that specifies the depth of the subgraphs that should be loaded from the connectors during indexing operations. The default value is 4. The enumeration literal is **`Option.INDEX_READ_DEPTH`**.

Option	Description
tablesIncludeColumnsForInheritedProperties	An advanced boolean flag that dictates whether the property definitions inherited from supertypes should be represented in the corresponding queryable table with columns. The JCR specification gives implementations some flexibility, so ModeShape allows this to be controlled. When this option is set to "false", then each table has only those columns representing the (single-valued) property definitions explicitly defined by the node type. When this option is set to "true" (the default), each table will contain columns for each of the (single-valued) property definitions explicitly defined on the node type and inherited by the node type from all of the supertypes. The enumeration literal is **`Option.TABLES_INCLUDE_COLUMNS_FOR_INHERITED_PROPERTIES`**.
performReferentialIntegrityChecks	An advanced boolean flag that specifies whether referential integrity checks should be performed upon Session.save(). If set to "true" (the default), referential integrity checks are performed to ensure that nodes referenced by other nodes cannot be removed. If the value is set to "false", then these referential integrity checks will not be performed when removing nodes. Many people generally discourage the use of REFERENCE properties because of the overhead and the need for referential integrity. These concerns are somewhat mitigated by the introduction in JCR 2.0 of the WEAKREFERENCE property type, which are excluded from referential integrity checks. This option is available for those cases where REFERENCE properties are not used within your content, and thus the referential integrity checks will never find violations. In these cases, you may disable these checks to slightly improve performance of delete operations. The enumeration literal is **`Option.PERFORM_REFERENTIAL_INTEGRITY_CHECKS`**.

Option	Description
versionHistoryStructure	An advanced flag that specifies the structure used to store version histories under the "**`/jcr:system/jcr:versionStorage`**" branch. The JCR 2.0 specification does not predefine any particular structure, but ModeShape supports two types: - A value of "flat" dictates that all "**`nt:versionHistory`**" nodes are stored with a name matching the UUID of the versioned node and directly under the "**`/jcr:system/jcr:versionStorage`**" node. For example, given a "**`mix:versionable`**" node with the UUID **`fae2b929-c5ef-4ce5-9fa1-514779ca0ae3`**, the corresponding " **`nt:versionHistory`**" node will be at "**`/jcr:system/jcr:versionStorage/fae2b929-c5ef-4ce5-9fa1-514779ca0ae3`**". - A value of "hierarchical" dictates that all "**`nt:versionHistory`**" nodes are stored under a hierarchical structure created by the first 8 characters of the UUID string. For example, given a "**`mix:versionable`**" node with the UUID **`fae2b929-c5ef-4ce5-9fa1-514779ca0ae3`**, the corresponding "**`nt:versionHistory`**" node will be at "**`/jcr:system/jcr:versionStorage/fa/e2/b9/29/c5ef-4ce5-9fa1-514779ca0ae3`**. The "hierarchical" structure is used by default and in cases where the option's value does not case-independently match the one of these two values. The enumeration literal is **`Option.VERSION_HISTORY_STRUCTURE`**, and the values are **`VersionHistoryOption.FLAT`** and **`VersionHistoryOption.HIERARCHICAL`**.
removeDerivedContentWithOriginal	An advanced boolean flag that dictates whether content derived from other content (e.g., that output by sequencers) should be automatically (re)moved when the content from which it was derived is (re)moved from the repository. For example, consider that a file is uploaded and sequenced, and that the content derived from the file is stored in the repository. When that file is (re)moved, this option dictates whether the derived content should also be (re)moved automatically. By default this option has a value of "true", ensuring that all derived content is deleted whenever the original content is deleted. A value of "false" will leave the derived content. The enumeration literal is **`Option.REMOVE_DERIVED_CONTENT_WITH_ORIGINAL`**.

Option	Description
useAnonymousAccessOnFailedLogin	A boolean flag that indicates whether any failed, non-anonymous login attempts will automatically cause the Session to be created using the anonymous context. If anonymous logins are not enabled (with the anonymousUserRoles option), then the login will still fail. By default this option has a value of "false", ensuring that non-anonymous login attempts either succeed as the requested user or fail. The enumeration literal is **`Option.USE_ANONYMOUS_ACCESS_ON_FAILED_LOGIN`**.
useSecurityContextCredentials	Older versions of ModeShape allowed client applications to pass in Credentials implementations that had a **`getSecurityContext()`** method that returned a SecurityContext object, which ModeShape would then use for authorization. However, since ModeShape now provides support for customized authentication and authorization modules, this is no longer needed and has been deprecated. If, however, your applications were written to use this **`SecurityContextCredentials`** implementation, then you can enable this option to turn the old behavior back on. Note, however, that this option will be removed in the next major release. Value is either "true" or "false" (default). The enumeration literal is **`Option.USE_SECURITY_CONTEXT_CREDENTIALS`**.

Warning

Setting the useAnonymousAccessOnFailedLogin option to "true" and setting the anonymousUserRoles to a valid value means that all login attempts will succeed, but named login attempts may actually succeed in an anonymous context. You can programattically determine which context is being used by checking the value of **`Session`**.getUserID().

17.6. Repository System Content

Each JCR repository contains information about the system in the "**`/jcr:system`**" area of the repository content. All of this system content applies to the whole repository (e.g., namespaces, node types, locks, versions, etc.) and therefore every session for each workspace sees the exact same "**`/jcr:system`**" content.

ModeShape implements this behavior by storing all "**`/jcr:system`**" content in a separate workspace, and then using federation to project that content into each workspace. This ensures that all workspaces see the same content, without having to duplicate the "**`/jcr:system`**" content in each workspace and ensure those copies stay in sync. Federation is better than duplication.

By default, ModeShape creates this separate system workspace in a transient, in-memory store. This works great for some simplistic cases, but this does not work when using clustering, , or dynamically registering namespaces or adding or changing node types. This is because these features all rely upon changing or adding content in the "**`/jcr:system`**" area. For example, version histories are stored under

"**/jcr:system/jcr:versionStorage**", node types under "**/jcr:system/jcr:versionStorage**", and namespaces under "**/jcr:system/mode:namespaces**".

In these situations, it is necessary to persist the system content in a repository source, and if clustering is enabled this source needs to be accessible to all members of the cluster. Many times, the easiest approach is to define an extra workspace in your repository source where the system content can be stored. It's also possible to define a separate repository source with a separate workspace for each repository's system content. (Using a separate source is required when the repository is using a single repository source that can only store limited kinds of nodes, like the file system connector or Subversion connector that can only store **nt:file** and **nt:folder** nodes.)

You should always configure each ModeShape repository with a source for its system workspace by using the **SYSTEM_SOURCE_NAME** repository option with a value that defines the name of source and name of the workspace in that source where the system content should be stored, in the format:

```
workspaceName@sourceName
```

This specifies the system content should be stored in the workspace named "**workspaceName**" in the "**sourceName**" repository source.

The system content can be stored in any repository source capable of storing any content and, in the case of clustering, that is accessible across multiple processes. For most people, this will mean a relational database.

17.7. Example: Defining a Source for System Content

The following is an abbreviated example of an XML configuration that defines a source for the system content (in a MySQL database) and a repository that uses it:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:mode="http://www.modeshape.org/1.0"
               xmlns:jcr="http://www.jcp.org/jcr/1.0">
  <mode:repositories>
    <mode:repository jcr:name="car repository" mode:source="Cars">
      <mode:options jcr:primaryType="mode:options">
        <!-- Explicitly specify the "system" workspace in the "SystemStore"
source. -->
        <systemSourceName jcr:primaryType="mode:option"
                          mode:value="system@SystemStore"/>
        ...
      </mode:options>
      ...
    </mode:repository>
    ...
  </mode:repositories>
  <mode:sources jcr:primaryType="nt:unstructured">
    <!-- One source for the "/jcr:system" content ... -->
    <mode:source jcr:name="SystemStore"

mode:classname="org.modeshape.connector.store.jpa.JpaSource"
                 mode:description="The database store for our system
content"
                 mode:dialect="org.hibernate.dialect.MySQLDialect"
                 mode:dataSourceJndiName="java:/MyDataSource"
                 mode:defaultWorkspaceName="system"
                 mode:autoGenerateSchema="validate"/>
```

```
    </mode:sources>
    <!-- An another source for the regular content ... -->
    <mode:source jcr:name="Cars"

mode:classname="org.modeshape.connector.store.jpa.JpaSource"
                 mode:description="The database store for our system
content"
                 mode:dialect="org.hibernate.dialect.MySQLDialect"
                 mode:dataSourceJndiName="java:/MyDataSource"
                 mode:defaultWorkspaceName="workspace1"
                 mode:autoGenerateSchema="validate">

<mode:predefinedWorkspaceNames>workspace1</mode:predefinedWorkspaceNames>

<mode:predefinedWorkspaceNames>workspace2</mode:predefinedWorkspaceNames>

<mode:predefinedWorkspaceNames>workspace3</mode:predefinedWorkspaceNames>
    </mode:sources>
    ...
  </mode:sources>
  ...
</configuration>
```

Of course, you can always use a separate workspace in your primary source, too:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:mode="http://www.modeshape.org/1.0"
xmlns:jcr="http://www.jcp.org/jcr/1.0">
  <mode:repositories>
    <mode:repository jcr:name="car repository" mode:source="Cars">
      <mode:options jcr:primaryType="mode:options">
        <!-- Explicitly specify the "system" workspace in the "Cars" source.
-->
        <systemSourceName jcr:primaryType="mode:option"
mode:value="system@Cars"/>
        ...
      </mode:options>
      ...
    </mode:repository>
    ...
  </mode:repositories>
  <mode:sources jcr:primaryType="nt:unstructured">
    <!--
    Define one source for the regular content with a special workspace for
the system content.
    -->
    <mode:source jcr:name="Cars"

mode:classname="org.modeshape.connector.store.jpa.JpaSource"
                 mode:description="The database store for our system
content"
                 mode:dialect="org.hibernate.dialect.MySQLDialect"
                 mode:dataSourceJndiName="java:/MyDataSource"
                 mode:defaultWorkspaceName="workspace1"
                 mode:autoGenerateSchema="validate">
```

```
<mode:predefinedWorkspaceNames>workspace1</mode:predefinedWorkspaceNames>

<mode:predefinedWorkspaceNames>workspace2</mode:predefinedWorkspaceNames>

<mode:predefinedWorkspaceNames>workspace3</mode:predefinedWorkspaceNames>
      <mode:predefinedWorkspaceNames>system</mode:predefinedWorkspaceNames>
    </mode:sources>
    ...
  </mode:sources>
  ...
</configuration>
```

17.8. Query Index Directory

ModeShape maintains a set of index files that are used to process queries and searches, using the Lucene search engine. By default, these indexes are kept in memory (primarily because it is easy to configure). But most production configurations should not store them in-memory but should instead store these index files on the local file system.

Each ModeShape repository can be configured where the indexes should be stored, using the "QUERY_INDEX_DIRECTORY" repository option (see **`JcrRepository.Option`**) when using the programmatic API or the "queryIndexDirectory" repository option in a ModeShape configuration file. The value of this setting should be the absolute or relative path to the folder where the indexes should be stored. In this directory, ModeShape will store the index files for each workspace in a folder named similarly to the workspace. Note that ModeShape will dynamically create these workspace folders as required.

For example, here is part of a ModeShape configuration file that specifies these index files should be stored in the "**`data/car_repository/indexes`**" folder, relative to the folder where the JVM process was started:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:mode="http://www.modeshape.org/1.0"
               xmlns:jcr="http://www.jcp.org/jcr/1.0">
  <mode:repositories>
    <mode:repository jcr:name="car repository" mode:source="Cars">
      <mode:options jcr:primaryType="mode:options">
        <!-- Explicitly specify the directory where the index files should
be stored. -->
        <queryIndexDirectory jcr:primaryType="mode:option"
                             mode:value="data/car_repository/indexes"/>
        ...
      </mode:options>
      ...
    </mode:repository>
    ...
  </mode:repositories>
  ...
</configuration>
```

17.9. Security Index Modules

ModeShape has pluggable authentication and authorization modules. Several modules are included and configured out-of-the-box, but it is now possible to implement and configure customized authentication and authorization logic. This section describes how these modules work, what's there out-of-the-box, and how to implement and add your own modules.

The **`AuthenticationProvider`** interface defines a single method:

```
public interface AuthenticationProvider {

  /**
   * Authenticate the user that is using the supplied credentials. If the
supplied
   * credentials are authenticated, this method should construct an
ExecutionContext
   * that reflects the authenticated environment, including the context's
valid
   * SecurityContext that will be used for authorization throughout the
Session.
   * <p>
   * Note that each provider is handed a map into which it can place name-
value
   * pairs that will be used in the Session attributes of the Session that
results
   * from this authentication attempt. ModeShape will ignore any attributes
if
   * this provider does not authenticate the credentials.
   * </p>
   *
   * @param credentials the user's JCR credentials, which may be an
   *  AnonymousCredentials if authenticating as an anonymous user
   * @param repositoryName the name of the JCR repository; never null
   * @param workspaceName the name of the JCR workspace; never null
   * @param repositoryContext the execution context of the repository, which
   * may be wrapped by this method
   * @param sessionAttributes the map of name-value pairs that will be
placed
   *  into the Session's attributes; never null
   * @return the execution context for the authenticated user, or null if
   * this provider could not authenticate the user
   */
  ExecutionContext authenticate( Credentials credentials,
                                 String repositoryName,
                                 String workspaceName,
                                 ExecutionContext repositoryContext,
                                 Map<String,Object> sessionAttributes );

}
```

When a client calls one of the Repository **`login`** methods, ModeShape calls the **`authenticate`** method on each of the **`AuthenticationProvider`** implementations registered with the Repository. As soon as one provider returns a non-null **`ExecutionContext`**, the caller is authenticated and ModeShape uses that **`ExecutionContext`** within the resulting Session .

When the client uses the Session and attempts to perform actions on the content, ModeShape uses the **`ExecutionContext`**'s SecurityContext to determine whether the user has the necessary privileges. If the SecurityContext object implements the **`AuthorizationProvider`** interface, then ModeShape will call the **`hasPermission(...)`** method, passing in the ExecutionContext, the repository name, the name of the source used for the repository, the workspace name, the path of the node upon which the actions are being applied, and the array of actions:

```
public interface AuthorizationProvider {

  /**
   * Determine if the supplied execution context has permission for all of
the
   * named actions in the named workspace. If not all actions are allowed,
the
   * method returns false.
   *
   * @param context the context in which the subject is performing the
   *        actions on the supplied workspace
   * @param repositoryName the name of the repository containing the
   *        workspace content
   * @param repositorySourceName the name of the repository's source
   * @param workspaceName the name of the workspace in which the path exists
   * @param path the path on which the actions are occurring
   * @param actions the list of ModeShapePermissions actions to check
   * @return true if the subject has privilege to perform all of the named
   *         actions on the content at the supplied path in the
   *         given workspace within the repository, or false otherwise
   */
  boolean hasPermission( ExecutionContext context,
                         String repositoryName,
                         String repositorySourceName,
                         String workspaceName,
                         Path path,
                         String... actions );
}
```

If the SecurityContext does not implement **`AuthorizationProvider`**, then ModeShape uses role-based authorization by mapping the actions into roles and then for each role calling the **`SecurityContext.hasRole(...)`** method on SecurityContext. Only if all of these invocations returns **`true`** will the operation be allowed to continue.

17.10. Available Security Providers

ModeShape comes with several **`AuthorizationProvider`** implementations that are automatically configured with every Repository, depending upon other settings and options. These providers are as follows:

- **`JaasProvider`** uses JAAS for all authentication and role-based authorization. This provider authenticates clients that login to the Repository with a **`SimpleCredentials`** object, where the username and password match that in the JAAS policy, or a **`JaasCredentials`** constructed with a specific and already-authenticated JAAS **`LoginContext`**. This provider can be disabled by setting the jaasLoginConfigName configuration options to an empty (i.e., zero-length) value; otherwise, the option defines the name of the JAAS login configuration and will default to "modeshape-jcr" if not explicitly set. (This provider also works in some J2EE containers, in which the JAAS Subject is not available via the standard JAAS API and instead requires use of the JACC API, which many J2EE containers support)
- **`SeamSecurityProvider`** delegates all authentication and role-based authorization to the Seam Security framework. This provider authenticates clients that login to the Repository with no need to pass a **`Credentials`** object. Note this does require obtaining a session for each servlet request, which is actually how the JCR API was intended to be used within web applications. This provider is automatically enabled when the Seam Security Identity class is found on the classpath.

- **`ServletProvider`** delegates all authentication and role-based authorization to the servlet framework. This provider authenticates clients that login to the Repository with a **`ServletCredentials`** object, which can be constructed with the **`HttpServletRequest`**. Note this does require obtaining a session for each servlet request, which is actually how the JCR API was intended to be used within web applications. This provider is automatically enabled when the HttpServletSession class is found on the classpath.
- **`AnonymousProvider`** will allow clients without Credentials to operate upon the repository, and will use role-based authorization based upon the roles defined by the anonymousUserRoles configuration option. This provider authenticates clients that provide an **`AnonymousCredentials`** to the Repository 's **`login(...)`** methods or use one of the **`login(...)`** methods that does not take a Credentials object.

Note

The **`SecurityContextProvider`** is also configured only when the useSecurityContextCredentials configuration option is set to 'true'. This provider authenticates clients that pass a **`SecurityContextCredentials`** object, and delegates all authentication to the embedded SecurityContext. This deprecated approach not enabled by default, and will be removed in the next major release of ModeShape. It remains in place to enable applications that use this approach to upgrade to ModeShape 2.6 (or later) without breaking their authentication mechanism.

17.11. Custom Providers

It is possible to provide your own authentication and authorization logic by providing one (or more) classes that implements the **`AuthorizationProvider`** interface, specifying the names of these classes in the configuration (see below), and making the classes available on the correct classpath.

Implementing the **`AuthorizationProvider`** interface is pretty straightforward. Your class needs a no-arg constructor, and the **`authenticate`** method must authenticate the credentials for the named repository and workspace. If the credentials are not authenticated, return null. Otherwise, create an **`ExecutionContext`** instance (from the **`ExecutionContext`** supplied in the **`repositoryContext`** parameter) to contain an appropriate SecurityContext instance for the authenticated user. As mentioned above, the SecurityContext should also implement the **`AuthorizationProvider`** interface for non-role-based authorization.

17.12. Example: Implement a Custom Provider

For example, let's imagine that our JCR application has its own authentication and authorization system. We can integrate with that by creating a new Credentials implementation called **`MyAppCredentials`** to encapsulate any information needed by the authentication/authorization system, which we'll assume is accessed by a singleton class **`SecurityService`**. We can then implement **`AuthenticationProvider`** as follows:

```
public class MyAppAuthorizationProvider implements AuthorizationProvider {

  private String appName;

  /**
   * Any public JavaBean properties can be set in the configuration
   */
  public void setApplicationName( String appName ) {
    this.appName = appName;
  }
```

```
  /**
   * Authenticate the user that is using the supplied credentials. If the
supplied
   * credentials are authenticated, this method should construct an
ExecutionContext
   * that reflects the authenticated environment, including the context's
valid
   * SecurityContext that will be used for authorization throughout the
Session.
   * <p>
   * Note that each provider is handed a map into which it can place name-
value
   * pairs that will be used in the Session attributes of the Session that
results
   * from this authentication attempt. ModeShape will ignore any attributes
if
   * this provider does not authenticate the credentials.
   * </p>
   *
   * @param credentials the user's JCR credentials, which may be an
   *  AnonymousCredentials if authenticating as an anonymous user
   * @param repositoryName the name of the JCR repository; never null
   * @param workspaceName the name of the JCR workspace; never null
   * @param repositoryContext the execution context of the repository, which
   * may be wrapped by this method
   * @param sessionAttributes the map of name-value pairs that will be
placed
   *  into the Session's attributes; never null
   * @return the execution context for the authenticated user, or null if
   * this provider could not authenticate the user
   */
  public ExecutionContext authenticate( Credentials credentials,
                                        String repositoryName,
                                        String workspaceName,
                                        ExecutionContext
repositoryContext,
                                        Map<String,Object>
sessionAttributes );
    if ( credentials instanceof MyAppCredentials ) {
      // Try to authenticate ...
      MyAppCredentials appCreds = (MyAppCredentials)credentials;
      String user = appCreds.getUser();
      Object token = appCreds.getToken();
      AppCreds creds = SecurityService.login(appName,user,token);
      if ( creds != null ) {
        // We're in ...
        SecurityContext securityContext = new MyAppSecurityContext(creds);
        return repositoryContext.with(securityContext);
      }
    }
    return null;
  }
}
```

where the **`MyAppSecurityContext`** is as follows:

```
public class MyAppSecurityContext
            implements SecurityContext, AuthorizationProvider {
  private final AppCreds creds;
  public MyAppSecurityContext( AppCreds creds ) {
    this.creds = creds;
  }

  /**
   * {@inheritDoc SecurityContext#getUserName()}
   *
   * @see SecurityContext#getUserName()
   */
  public final String getUserName() {
      return creds.getUser();
  }

  /**
   * {@inheritDoc SecurityContext#hasRole(String)}
   *
   * @see SecurityContext#hasRole(String)
   */
  public final boolean hasRole( String roleName ) {
      // shouldn't be called since we've implemented AuthorizationProvider
      return false;
  }

  /**
   * {@inheritDoc}
   *
   * @see org.modeshape.graph.SecurityContext#logout()
   */
  public void logout() {
      creds.logout();
  }

  /**
   * {@inheritDoc}
   *
   * @see org.modeshape.jcr.security.AuthorizationProvider.hasPermission
   */
  public boolean hasPermission( ExecutionContext context,
                            String repositoryName,
                            String repositorySourceName,
                            String workspaceName,
                            Path path,
                            String... actions ) {
    // This is imaginary and simplistic, but you'd implement any
authorization logic here ...
    return this.creds.isAuthorized(repositoryName,workspaceName,path);
  }
}
```

Then we just need to configure the Repository to use this provider. In the ModeShape configuration files, there is an optional "**`mode:authenticationProviders`**" child element of "**`mode:repository`**", and within this fragment you can define zero or more authentication providers by specifying a name, the class, an optional description, and optionally any bean properties that should be called upon instantiation. (Note that the

class will be instantiated only once per Repository instance). Here's an example configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:mode="http://www.modeshape.org/1.0"
               xmlns:jcr="http://www.jcp.org/jcr/1.0">
  <mode:repositories>
    <mode:repository jcr:name="MyApp Repository" mode:source="Store">
      ...
      <mode:authenticationProviders>
        <!-- Specify the providers in a manner similar to sequencer
             definitions are defined -->
        <mode:authenticationProvider jcr:name="CustomProviderA"

mode:classname="org.example.MyAppAuthorizationProvider">
          <mode:description>My authentication provider</mode:description>
          <!-- Set JavaBean properties on provider if needed -->
          <mode:appName>MyAppName</mode:appName>
        </mode:authenticationProvider>
        ...
      </mode:authenticationProviders>
      ...
    </mode:repository>
    ...
  </mode:repositories>
  ...
</configuration>
```

17.13. Clustering with ModeShape

ModeShape has the ability to have a cluster of JcrEngine instances distributed across multiple processes while behaving as though everything was happening in a single process. With clusters, the workload can be distributed across multiple machines, increasing tolerance against failure while allowing ModeShape to scale out to handle more workload.

ModeShape clustering uses the powerful, flexible and mature JGroups library to handle all network communication within the cluster. JGroups provides a wealth of capabilities, including automatically detecting new engines in the cluster (called discovery), reliable multicast communication, and automatic determination of the master node in the cluster. JGroups has a flexible protocol stack, works across firewalls, WANs and LANs, and supports multiple transport protocols, failure detection, reliable unicast and multicast message transmission, and encryption.

By default, clustering is not enabled. This means that each JcrEngine instance is self-contained and will not be aware of changes made in other JcrEngine instances. This is perfect in many lightweight or embedded scenarios, because it does not introduce any overhead associated with network communication.

However, clustering ModeShape is very easy and requires only a few simple steps:

1. Enable clustering in the ModeShape configuration (more on this in a bit).
2. Include the modeshape-clustering module in your application by JAR file.
3. Start (or deploy) multiple JcrEngine instances using the same configuration. For embedded scenarios, this means instantiating multiple JcrEngine instances in multiple processes. In other cases, this means deploying ModeShape to multiple servers (either using the WebDAV server, REST server, or into JNDI and using with your own applications).

Your JCR-based application does not need to change in any other ways. Any EventListener implementations registered in Sessions on any of the engines will be notified of all events, regardless of whether those events were due to changes in the local or remote engines.

It also does not matter how many Repository instances are defined in the configuration and managed by each JcrEngine instance: each engine in the cluster can manage multiple named repositories. ModeShape ensures that all Sessions for a named repository see the changes made to that repository, regardless of where those sessions are located in the cluster. Likewise, those same changes will not be visible to the sessions for any other named repository.

17.14. Enabling Clustering in ModeShape

A ModeShape configuration can have a "clustering" fragment that defines the name of the cluster and the JGroups configuration:

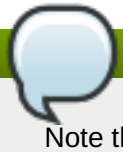
```
<mode:clustering clusterName="modeshape-cluster" configuration="jgroups-
modeshape.xml" />
```

The "clusterName" is a string that is a logical name of the cluster; all engines connecting to the same name form a cluster. Any messages multicast from one engine in the cluster will be received by all other members of the cluster. Again, the cluster name is independent of the repositories managed by th

The "**`configuration`**" value is a string that is one of:

- the absolute file system path to the file containing the JGroups XML configuration;
- the relative file system path to the file containing the JGroups XML configuration, relative to the current working directory of the Java process;
- the name of a resource on the classpath containing the JGroups XML configuration;
- the URL that can be resolved to the JGroups XML configuration; or
- the string representation of JGroups configuration, either in XML format or the older string format.

If the "**`configuration`**" property is not given, ModeShape will use the default JGroups configuration (as defined by the specific JGroups version).



Note

Note that all engines in the cluster must have the same JGroups configuration. In fact, all engines in the cluster will almost always have exactly the same ModeShape configuration.

Here is an example of a "**`clustering`**" fragment defining a cluster named "modeshape-cluster" using the JGroups configuration defined in the "jgroups-modeshape.xml" file at the supplied URL:

```
<clustering clusterName="modeshape-cluster"
   configuration="file://some/path/jgroups-modeshape.xml" />
```

This next example uses the JGroups configuration defined in the "jgroups-modeshape.xml" resource file on the classpath (or as an absolute path on a *nix system):

```
<clustering clusterName="modeshape-cluster"
   configuration="/some/path/jgroups-modeshape.xml" />
```

Next is an example that specifies the JGroups configuration using the older string representation of the form:

```
<clustering clusterName="modeshape-cluster"
   configuration="PROTOCOL(param=value;param=value):PROTOCOL:PROTOCOL" />
```

Of course, the "**`configuration`**" property can be specified as a child element, too (line breaks added for readability):

```
<clustering clusterName="modeshape-cluster">
      <configuration>UDP(max_bundle_size="60000":max_bundle_timeout="30"):
                     PING(timeout="2000"):...</configuration>
</clustering>
```

And finally an example that specifies the JGroups configuration using the newer XML representation (line breaks added for readability):

```
<clustering clusterName="modeshape-cluster">
  <configuration><![CDATA[<config><UDP max_bundle_size="60000"
       max_bundle_timeout="30".../><PING timeout="2000"/>...</config>]]>
  </configuration>
</clustering>
```

Note that the this example uses a child XML element for the "**`configuration`**", along with a CDATA section, so that the XML configuration can be nested within the ModeShape configuration.

Warning

Remember to specify the system workspace name for each repository that is clustered.

17.15. JGroups Configuration

The JGroups configuration defines a protocol stack that is used for messaging, starting with the bottom-most protocol and ending with the top-most protocol.

An example of the recommended JGroups XML format follows:

```
<config>
   <UDP
        mcast_addr="${jgroups.udp.mcast_addr:228.10.10.10}"
        mcast_port="${jgroups.udp.mcast_port:45588}"
        discard_incompatible_packets="true"
        max_bundle_size="60000"
        max_bundle_timeout="30"
        ip_ttl="${jgroups.udp.ip_ttl:2}"
        enable_bundling="true"
        thread_pool.enabled="true"
        thread_pool.min_threads="1"
        thread_pool.max_threads="25"
        thread_pool.keep_alive_time="5000"
        thread_pool.queue_enabled="false"
```

```
        thread_pool.queue_max_size="100"
        thread_pool.rejection_policy="Run"
        oob_thread_pool.enabled="true"
        oob_thread_pool.min_threads="1"
        oob_thread_pool.max_threads="8"
        oob_thread_pool.keep_alive_time="5000"
        oob_thread_pool.queue_enabled="false"
        oob_thread_pool.queue_max_size="100"
        oob_thread_pool.rejection_policy="Run"/>
    <PING timeout="2000"
            num_initial_members="3"/>
    <MERGE2 max_interval="30000"
            min_interval="10000"/>
    <FD_SOCK/>
    <FD timeout="10000" max_tries="5" />
    <VERIFY_SUSPECT timeout="1500"  />
    <BARRIER />
    <pbcast.NAKACK
                  use_mcast_xmit="false" gc_lag="0"
                  retransmit_timeout="300,600,1200,2400,4800"
                  discard_delivered_msgs="true"/>
    <UNICAST timeout="300,600,1200,2400,3600"/>
    <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
                  max_bytes="400000"/>
    <VIEW_SYNC avg_send_interval="60000"   />
    <pbcast.GMS print_local_addr="true" join_timeout="3000"
                view_bundling="true"/>
    <FC max_credits="20000000"
                   min_threshold="0.10"/>
    <FRAG2 frag_size="60000"  />
    <pbcast.STATE_TRANSFER  />
</config>
```

For more details on how to configure the JGroups stack, see the JGroups Manual.

Note

JGroups is also used in Infinispan, JBoss EAP, and other open source projects, and many of the JGroups configurations will work with ModeShape deployed in those same environments. For example, this blog post describes how to configure JGroups with three autodiscovery options available on Amazon EC2.

17.16. Using ModeShape in Web Applications

Your web application or JBoss service can use one of the JCR Repository instances running inside the ModeShape service with a URL such as:

```
jndi:jcr/local?repositoryName=repository
```

Be sure to use the correct repository name.

Since the JCR API JAR is on the global classpath, your web application can use the JCR API without having to include the JAR file in your application's WAR file. In fact, your application will likely get

ClassCastExceptions if it does include the JCR API in its WAR file. Plus, if needed, your application can use ModeShape's "**`org.modeshape.jcr.api`**" extensions to the JCR API (again, on the global classpath), and should not need or use any of the classes or interfaces in the ModeShape implementation.

17.17. Configuring a Predefined Node Hierarchy

The ***`SOA_ROOT`*`/jboss-as/server/`*`PROFILE`*`/deploy/modeshape-services.jar/modeshape-initial-content.xml`** file is an optional XML file which can be added to the main ModeShape configuration file:

```
<mode:initialContent mode:workspaces="default"
mode:applyToNewWorkspaces="true" mode:content="modeshape-initial-
content.xml"/>
```

Its purpose is to allow users to configure, at repository startup, a predefined node hierarchy with which the repository will be pre-populated. In other words, once the repository has started up, the node hierarchy from the XML file will be already present in the repository. The name of the XML element will be the name of the node, while the XML structure itself (the nested elements) will define the hierarchy.

To define a specific JCR type for a node (or for that matter any other valid JCR property), one needs to define the JCR namespace:

```
<files xmlns:jcr="http://www.jcp.org/jcr/1.0" jcr:primaryType="nt:folder"
jcr:mixinTypes="mode:publishArea">
```

This shows the definition of a "files" node, of type "nt:folder" and which has the mixin "mode:publishArea".

17.18. The ModeShape REST Server

Metadata Repository provides a RESTful interface to its JCR implementation that allows HTTP-based access and updating of content.

The REST server is deployed in /modeshape-rest.

17.19. Supported Resources and Methods

The REST server currently supports the URIs and HTTP methods described below.

Table 17.2. Supported URIs for the Metadata Repository REST Server

URI Pattern	Description	HTTP Methods
http://<host>:<port>/modeshape-rest	Returns a list of accessible repositories	GET
http://<host>:<port>/modeshape-rest/*{repositoryName}*>	Returns a list of accessible workspaces within that repository	GET
http://<host>:<port>/modeshape-rest/*{repositoryName}*/*{workspaceName}*	Returns a list of available operations within the workspace	GET

URI Pattern	Description	HTTP Methods
http://<host>:<port>/modeshape-rest/*{repositoryName}*/*{workspaceName}*/item/*{path}*	Accesses the item (node or property) at the path	GET, POST, PUT, DELETE
http://<host>:<port>/modeshape-rest/*{repositoryName}*/*{workspaceName}*/query	Executes the query in the request body	POST

17.20. Return a List of Accessible Repositories

A typical conversation might start with a request to the server to dynamically discover the available repositories.

```
GET http://www.example.com/modeshape-rest
```

This request would generate a response that mapped the names of the available repositories to metadata information about the repositories:

```
{
   "eds" : {
      "repository" :
      {
         "name" : "eds",
         "resources" : {  "workspaces":"/modeshape-rest/eds" }
      }
   }
}
```

The actual response would not be pretty-printed like the example, but the format would be the same. The name of the repository ("repository" URL-encoded) is mapped to a repository object that contains a name (the redundant "repository") and a list of available resources within the repository and their respective URIs. Note that Metadata Repository supports deploying multiple JCR repositories side-by-side on the same server, so this response could easily contain multiple repositories in a real deployment.

17.21. Return a List of Workspaces for a Repository

Once you know the name of an accessible repository, you can retrieve a list of its workspaces:

```
GET http://www.example.com/modeshape-rest/eds
```

This request (and all of the following requests) actually create a JCR session (**`javax.jcr.Session`**) to service the request and require that security be configured. The response looks similar to the following:

```
{
   "default" : {
      "workspace" : {
         "name" : "default",
         "resources" : {
            "items":"/modeshape-rest/eds/default/items",
            "query":"/modeshape-rest/eds/default/query"
```

```
         },
      }
   }
}
```

Like the first response, this response consists of a list of workspace names mapped to metadata about the workspaces. The example above only lists one workspace for simplicity, but there could be many different workspaces returned in a real deployment. Note that the "items" resource builds the full URI to the root of the items hierarchy, including the encoding of the repository name and the workspace name and the "query" resource builds the full URI needed to execute queries.

17.22. Access a Repository Item

Once you know the path to a repository's workspaces, you can retrieve the root item of the repository:

```
GET http://www.example.com/modeshape-rest/eds/default/items
```

Any other item in the repository could be accessed by appending its path to the URI above. In a default repository with no content, this would return the following response:

```
{
   "properties": {
      "jcr:primaryType": "mode:root",
      "jcr:uuid": "97d7e2ef-996e-4d99-8ec2-dc623e6c2239"
   },
   "children": ["jcr:system"]
```

The response contains a mapping of property names to their values and an array of child names. Had one of the properties been multi-valued, the values for that property would have been provided as an array as well, as will shortly be shown.

The items resource also contains an option query parameter: **`mode:depth`**. This parameter, which defaults to 1, controls how deep the hierarchy of returned nodes should be. Had the request had the parameter:

```
GET http://www.example.com/modeshape-rest/eds/default/items?mode:depth=2
```

Then the response would have contained details for the children of the root node as well.

```
{
   "properties": {
      "jcr:primaryType": "mode:root",
      "jcr:uuid": "163bc5e5-3b57-4e63-b2ae-ededf43d3445"
   },
   "children": {
      "jcr:system": {
         "properties": {"jcr:primaryType": "mode:system"},
         "children": ["mode:namespaces"]
      }
   }
}
```

17.23. Modify Repository Content

It is also possible to use the RESTful API to add, modify and remove repository content. Removal is simple - a DELETE request with no body returns a response with no body.

```
DELETE http://www.example.com/modeshape-
rest/eds/default/items/path/to/deletedNode
```

Adding content requires a POST to the name of the *relative* root node of the content that you wish to add and a request body in the same format as the response from a GET. Adding multiple nodes at once is supported, as shown below.

```
POST http://www.example.com/modeshape-rest/eds/default/items/newNode
{
   "properties": {
      "jcr:primaryType": "nt:unstructured",
      "jcr:mixinTypes": "mix:referenceable",
      "someProperty": "foo"
   },
   "children": {
      "newChildNode": {
         "properties": {"jcr:primaryType": "nt:unstructured"}
      }
   }
}
```

Note that protected properties like **`jcr:uuid`** are not provided but that the primary type and mixin types are provided as properties. The REST server will translate these into the appropriate calls behind the scenes. The JSON-encoded response from the request will contain the node that you just posted, including any autocreated properties and child nodes.

If you do not need this information, add **`mode:includeNode=false`** as a query parameter to your URL.

```
POST http://www.example.com/modeshape-rest/eds/default/items/newNode?
mode:includeNode=false

{
   "properties": {
      "jcr:primaryType": "nt:unstructured",
      "jcr:mixinTypes": "mix:referenceable",
      "someProperty": "foo"
   },
   "children": {
      "newChildNode": {
         "properties": {"jcr:primaryType": "nt:unstructured"}
      }
   }
}
```

This will instruct the REST server to only return the path of the newly-created node in the response.

The PUT method allows for updates of nodes and properties. If the URI points to a property, the body of the request must contain the new JSON-encoded value for the property, which includes the property name, allowing proper determination of whether the values are binary.

```
PUT http://www.example.com/modeshape-
rest/eds/default/items/some/existing/node/someProperty
```

```
{
   "someProperty" : "bar"
}
```

Setting multiple properties at once can be performed by providing a URI to a node instead of a property. The body of the request should then be a JSON object that maps property names to their new values.

```
PUT http://www.example.com/modeshape-
rest/eds/default/items/some/existing/node

{
   "someProperty": "foobar",
   "someOtherProperty": "newValue"
}
```

The JSON request can even contain a properties container:

```
PUT http://www.example.com/modeshape-
rest/eds/default/items/some/existing/node

{
   "properties": {
      "someProperty": "foobar",
      "someOtherProperty": "newValue"
   }
}
```

A subgraph can be updated all at once using a PUT against a URI of the top node in the subgraph. Note that, in this case, every node in the subgraph must be provided in the JSON request (any node not in the request will be removed). This method will attempt to set all of the properties to the new values as specified in the JSON request, plus any descendant node in the JSON request that does not reflect an existing node will be created while any existing node not reflected in the JSON request will be removed. Any specifications of **`jcr:primaryType`** are ignored if the node already exists. In other words, the request only needs to contain the properties that are changed. Of course, if a node is being added, all of its properties need to be included in the request.

Here is an example:

```
PUT http://www.example.com/modeshape-
rest/eds/default/items/some/existing/node

{
   "properties": {
      "jcr:primaryType": "nt:unstructured",
      "jcr:mixinTypes": "mix:referenceable",
      "someProperty": "foo"
   },
   "children": {
      "childNode": {
         "properties": {"jcr:primaryType": "nt:unstructured"}
      }
   }
}
```

This will update the existing node at /some/existing/node with the specified properties, and ensure that it contains one child node named **`childNode`**. Note that the body of this request is identical in structure to that of the POST requests.

17.24. Query the Content Repository

Queries can be executed through the REST interface by sending a POST request to the query URI with the query statement in the body of the request. The query language *must* be specified by setting the appropriate MIME type.

All queries for a given workspace are posted to the same URI and the request body is not JSON-encoded.

```
POST http://www.example.com/modeshape-rest/eds/default/query

/a/b/c/d[@foo='bar']
```

Assuming that the above request was a POST with a content type of **`application/jcr+xpath`**, a response would be generated that consisted of a JSON object that contained a property named **`rows`**. The **`rows`** property would contain an array of rows with each element being a JSON object that represented one row in the query result set.

```
{
   "types": {
      "someProperty": "STRING",
      "someOtherProperty": "BOOLEAN",
      "jcr:path": "STRING",
      "jcr:score": "DECIMAL"
   },
   "rows": {
      {
         "someProperty": "foobar",
         "someOtherProperty": "true",
         "jcr:path" : "/a/b/c/d",
         "jcr:score" : 0.9327
      },
      {
         "someProperty": "localValue",
         "someOtherProperty": "false",
         "jcr:path" : "/a/b/c/d[2]",
         "jcr:score" : 0.8143
      }
   }
}
```

The JSON object in the response also contains a **`types`** property. The value of the **`types`** property is a JSON object that maps column names to their JCR type.

17.25. Query Content Types

Table 17.3. Query Content Types for the Metadata Repository REST Server

Query Language	Content Type
XPath	application/jcr+xpath

Query Language	Content Type
JCR-SQL	application/jcr+sql
JCR-SQL2	application/jcr+sql2
Full Text Search	application/jcr+search

If no content type is specified or the content type for the request is not one of the content types listed above, the request will generate a response code of 400 (BAD REQUEST).

17.26. Binary Properties

Binary property values are included in any of the responses or requests, but are represented as string values containing the Base64 encoding of the binary content. Any such property is explicitly annotated such that /base64/ is appended to the property name. This makes it very clear to the client and service which properties are encoded, allowing them to properly decode the values before use. The /base64/ suffix cannot be used in a real property name without escaping.

Here's an example of a node containing a **`jcr:primaryType`** property with a single string value, a **`jcr:uuid`** property with another single UUID value, another **`options`** property that has two integer values, and a fourth **`content`** property that has a single binary value:

```
{
   "properties": {
      "jcr:primaryType": "nt:unstructured",
      "jcr:uuid": "163bc5e5-3b57-4e63-b2ae-ededf43d3445"
      "options": [ "1", "2" ]
      "content/base64/":

"TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSB0aGl
z
IHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBhIGx1c3Qgb2Yg
dGhlIG1pbmQsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2Ugb2YgZGVsaWdodCBpbiB0aGUgY29udGlu
dWVkIGFuZCBpbmRlZmF0aWdhYmxlIGdlbmVyYXRpb24gb2Yga25vd2xlZGdlLCBleGNlZWRzIHRo
ZSBzaG9ydCB2ZWhlbWVuY2Ugb2YgYW55IGNhcm5hbCBwbGVhc3VyZS4="
   },
}
```

All values of a property will always be Base64 encoded if at least one of the values is binary. If there are multiple values, then they will be separated by commas and will appear within brackets (**`[ ]`**) like other properties.

17.27. ModeShape REST Client API

The ModeShape REST Client API provides a way of using the ModeShape REST web service to publish (upload) and unpublish (delete) files from ModeShape repositories. Java objects open the HTTP connection, create the HTTP request URLs, attach the payload associated with **`PUT`** and **`POST`** requests, parse the HTTP JSON response back into Java objects, and close the HTTP connection.

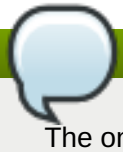
The **`eds/modeshape/client/modeshape-client.jar`** JAR file contains the ModeShape REST Client API.

The **`org.modeshape.web.jcr.rest.client.domain`** package contains the following required objects:

- **`Server`** - hosts one or more ModeShape JCR repositories,

- **`Repository`** - a ModeShape JCR repository containing one or more workspaces, and
- **`Workspace`** - a ModeShape JCR repository workspace.

Publish and unpublish operations are performed using a class that implements the **`org.modeshape.web.jcr.rest.client.IRestClient`** interface.



Note

The only included implementation of **`IRestClient`** is **`org.modeshape.web.jcr.rest.client.json.JsonRestClient`** which uses JSON as its data format.

17.28. Publish a File Using the REST Client API

```
// Setup POJOs
Server server = new Server("http://localhost:8080", "username", "password");
Repository repository = new Repository("repositoryName", server);
Workspace workspace = new Workspace("workspaceName", repository);

// Publish
File file = new File("/path/to/file");
IRestClient restClient = new JsonRestClient();
Status status = restClient.publish(workspace, "/workspace/path/", file);

if (status.isError()
{
   // Handle error here
}
```

Successfully executing the above code results in the creation of a JCR folder node (**`nt:folder`**) for each segment of the workspace path (if the folder didn't already exist). Also, a JCR file node (a node with primary type **`nt:file`**) is created or updated under the last folder node and the file contents are encoded and uploaded into a child node of that file node.

Refer to the quickstart example in the ***`SOA_ROOT`*`/jboss-as/samples/quickstarts/modeshape_helloworld_publish/`** directory.

17.29. Repository Providers

The ModeShape REST and ModeShape WebDAV servers can provide access to other JCR repositories by implementing the org.modeshape.web.jcr.spi.RepositoryProvider interface.

There are four methods defined by the RepositoryProvider interface: **`startup`**, **`getJcrRepositoryNames`**, **`getSession`** and **`shutdown`**. When **`org.modeshape.web.jcr.ModeShapeJcrDeployer`** starts, it will call the **`RepositoryProvider startup`** method which will load the configuration (for example, from a **`web.xml`** file) and initialize the repository.

As an example, here's the ModeShape JCR provider implementation of this method with exception handling omitted for brevity.

```
public void startup( ServletContext context ) {
    String configFile = context.getInitParameter(CONFIG_FILE);

     InputStream configFileInputStream =
getClass().getResourceAsStream(configFile);
     jcrEngine = new
JcrConfiguration().loadFrom(configFileInputStream).build();
     jcrEngine.start();
}
```

The name of configuration file for the **`JcrEngine`** is read from the servlet context and used to initialize the engine. Once the repository has been started, it is ready to accept the main methods that provide the interface to the repository.

The first method returns the set of repository names supported by this repository.

```
public Set<String> getJcrRepositoryNames() {
    return new HashSet<String>(jcrEngine.getRepositoryNames());
}
```

The ModeShape JCR repository does support multiple repositories on the same server. Other JCR implementations that don't support multiple repositories are free to return a singleton set containing any string from this method.

The other required method returns an open JCR Session for the user from the current request in a given repository and workspace. The provider can use the **`HttpServletRequest`** to get the authentication credentials for the HTTP user.

```
public Session getSession( HttpServletRequest request,
                           String repositoryName,
                           String workspaceName ) throws
RepositoryException {
    Repository repository = getRepository(repositoryName);

 SecurityContext context = new ServletSecurityContext(request);
 Credentials credentials = new SecurityContextCredentials(context);
    return repository.login(credentials, workspaceName);
}
```

The **`getSession(...)`** method is used by most of the REST server methods to access the JCR repository and return results as needed.

Finally, the **`shutdown()`** method signals that the web context is being undeployed and the JCR repository should shutdown and clean up any resources that are in use.

Appendix A. Revision History

Revision 6.3.0-06	Mon Oct 21 2016	David Le Sage
Updates for 6.3.		