Dodeca Shell

Applied OLAP

Table of Contents

Getting Started

Dodeca Shell (Dshell) is a command-line utility that helps you work with a Dodeca repository. It connects directly to the repository database (without going through the Dodeca server itself). Various actions can be performed on a repository, including viewing, exporting, importing, and searching for data.

Dshell is packaged as a self-contained runnable JAR file. In order to run it, you need to have Java 1.8+ installed on your system. It is much more convenient to run when you have `java` available on your system path, so that you only need to type `java` to run it, as opposed to a fully qualified path.

You can quickly test if Java is available on the current path as well as of a new enough version by trying to run the Java version command from a commmand-line:

```
java -version
```

If Java is installed and available on the path, you may see output like the following:

```
java version "1.8.0_25"
Java(TM) SE Runtime Environment (build 1.8.0_25-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.25-b02, mixed mode)
```

You can run dshell by executing the following command:

```
java -Dloader.path=lib/ -jar dodeca-shell.jar
```

This command defines a variable named `loader.path` and sets it to the `lib` subfolder of the current working directory. This setting sets the folder that Dshell should look in for additional JDBC drivers that may be needed to connect to your Dodeca repository. You will likely need additional JAR files for your repository, based on its database technology (such as Oracle, Microsoft SQL Server, DB2, MySQL, and others).

If everything works correctly, you will see Dshell start up similar to the following:

```
       __     __          ____
  ____/ /____/ /_  ___  / / /
 / __  / ___/ __ \/ _ \/ / /
/ /_/ (__  ) / / /  __/ / /
\__,_/____/_/ /_/\___/_/_/
You are connected to an internal repository. Use the connect command to connect to a Dodeca repository
Type 'help connect' for information on how to use the connect command
dshell/:>
```

By default, dshell starts up and is connected to an internal repository that does not have any objects in it. You can use this workspace to facilitate certain types of testing, but most likely you will need and want to connect it to your own repository.

Connecting To Your Dodeca Repository

The easiest way to connect to a repository is to create a properties file with the connection settings in it. This can be as simple as a file with the following contents:

```
dodeca.datasource.url = jdbc:mysql://server/dodeca?noDatetimeStringSync=true
dodeca.datasource.username = dodeca
dodeca.datasource.password = password
```

The preceding example is for a repository inside a MySQL database, therefore the JDBC URL is formatted specifically for MySQL. The URL format for other database types typically varies. You can use the same

setting from your existing `hibernate.properties` file configured for your Dodeca server or attempt to create the URL yourself. The username and password fields are defined with the plaintext version of the password.

For convenience, you can use the suffix `.dodeca.properties` on your connection file, for a total file name such as `mysql-dev.dodeca.properties`. This allows you to use the `list-connections` command in the shell, which will list all of the files in the current directory that have this suffix. You may then use the `connect` command to refer to this short version of the file, such as:

```
connect mysql-dev
```

You may use the fully-qualified name of the file as well. You can switch between different repository connections during a single shell session, even if those connections are different database technologies (so long as the appropriate JDBC drivers are available, of course).

Most shell commands require having specified a tenant to work with by using the `use` command. For example, to set the tenant to `SAMPLE`, you would type:

```
use SAMPLE
```

Assuming that a tenant named `SAMPLE` exists, the shell prompt will update to indicate the current tenant:

```
dshell/SAMPLE:>
```

Using Tab-based Auto-completion

Many commands in the shell support tab-based auto-completion. This can make entering artfiact IDs, tenants, commands, and file names much more quick and less error prone.

For example, consider when there are a number of tenants available, such as by using the `list-tenants` command:

```
dshell/SAMPLE:>list-tenants
+---------------------------+
|Tenant                     |
+---------------------------+
|DEMOWARE                   |
|SAMPLE                     |
+---------------------------+
```

You may then wish to switch to the `DEMOWARE` tenant. You can have the shell provide you a list of items to select from, or a list of items based on what you have already typed in. For example, upon typing `use DE` followed by pressing the tab key, the shell will finish off the text by using the matching tenant (in this case, `DEMOWARE`), and you can then press enter to execute the command.

If you have not started typing the text for an item, the shell may prompt your for the parameter name first, which you can select and press tab again in order to get the list of all available options.

Scripting

The shell offers limited scripting/automation capabilities by way of running the shell with an additional parameter that specifies a list of commands to run sequentially. For instance, you may wish to automate a process that connects to your repository and exports all of the comments for a certain Dodeca tenant named `SAMPLE`. The sequence of commands for accomplishing this would be the following:

```
connect mysql-dev
use SAMPLE
export-comments --file comment_export.xml
```

You can then put these lines into a file named `export_comments.dshell` and then run it with the following command:

```
java -Dloader.path=lib/ -jar dodeca-shell.jar @export_comments.dshell
```

Note that `@` symbol in front of the scripting filename (`@export_comments.dshell`).

Getting Help With Commands

You can get a list of all available commands by typing `help`. You can get help and parameter information by typing `help` followed by the command name. For instance, to get help on the `export-comments` command, type:

```
help export-comments
```

You will then be presented with help like the following:

```
NAME
    export-comments - Export comments

SYNOPSYS
    export-comments [--file] file  [[--author] string]  [[--include-key-spec]
    comment-key-spec]  [[--exclude-key-spec] comment-key-spec]

OPTIONS
    --file or -F  file

        [Mandatory]

    --author  string

        [Optional, default = <none>]

    --include-key-spec  comment-key-spec

        [Optional, default = <none>]

    --exclude-key-spec  comment-key-spec

        [Optional, default = <none>]
```

Common Operations

Display Artifact Contents

Many Dodeca objects are stored as compressed XML. You can decompress and view the contents by using the `print-artifact` command and specifying an artifact. For example, to view the XML conents of version 1 of a certain View artifact named `IncStmt`, you may use the following command:

```
print-artifact IncStmt VIEW 1
```

The contents are then displayed:

```
<View>
  <ViewInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <ID>IncStmt</ID>
    <Name>Income Statement</Name>
    <ViewTypeID>ExcelEssbase</ViewTypeID>
    <NamedPropertySets />
  </ViewInformation>
  <Properties xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <AllowSave>true</AllowSave>
    <AllowSharing>true</AllowSharing>
    <AutoBuildOnOpen>true</AutoBuildOnOpen>
    <CaptionAfterBuild>Income Statement for [T.Market] - [T.Product] -
[T.Scenario]</CaptionAfterBuild>
    <MergeableToolbarsConfigurationID>View Main </MergeableToolbarsConfigurationID>
    <ViewTokens />
    <ViewToolbarsConfigurationID>Essbase View Standard Limited</ViewToolbarsConfigurationID>
    <WindowsViewUIObjectTypeID>WorkbookView</WindowsViewUIObjectTypeID>
    <AllowedLROTypes>2</AllowedLROTypes>
    <EssbaseConnectionID>Sample.Basic</EssbaseConnectionID>

<EssbaseLoginServiceObjectTypeID>EssbaseLoginDialogWithGenericCaption</EssbaseLoginServiceObjectTypeID
>
    <EssProperties>
      <SendZerosAsMissing>true</SendZerosAsMissing>
      <UseAliases>true</UseAliases>
      <MissingTextString>0</MissingTextString>
      ...
```

Note that all artifacts in a Dodeca repository are uniquely identified by a unique combination of four pieces of information: tenant, artifact ID, category, and version. In the previous example, the tenant is already set (by way of the `use` command), and then then three other properties are specified. You do not have to always fully qualify an exact artifact, but if the artifact specification is otherwise ambiguous, the shell will report an error and tell you that you need to be more specific.

For instance, in the demo tenant that we have been working with, there are multiple artifacts with the name `IncStmt`, so attempts to print the artifact contents with the following command:

```
dshell/SAMPLE:>print-artifact IncStmt
```

Will result in the following error message:

```
Ambiguous artifact specification 'IncStmt' as there are 2 artifacts that match (GENERAL/1, VIEW/1)
```

Helpfully enough, the error tells us that the ambiguous name is due to there being an artifact with the same ID but with categories of `GENERAL` and `VIEW`. We can then narrow our artifact specification:

```
dshell/SAMPLE:>print-artifact IncStmt VIEW
```

(Notice that we did not have to specify the version).

View Artifact Dependencies

You can view the dependencies (artifacts that a given artifact depends on) with by using the `view-dependencies` command and specifying the artifact:

```
dshell/SAMPLE:>view-dependencies IncStmt VIEW 1
```

Example output:

```
dshell/SAMPLE:>view-dependencies IncStmt VIEW 1
has root (VIEW/1) IncStmt
    has Essbase connection (ESSBASE_CONNECTION/1) Sample.Basic
    has selector (SELECTOR/1) 4_Market
    has selector (SELECTOR/1) 3_Product
    has selector (SELECTOR/1) 5_Scenario
    has selector list (SELECTOR_LIST/1) Market_Tree_AutoOpen
    has selector list (SELECTOR_LIST/1) Product_Tree_AutoOpen
    has selector list (SELECTOR_LIST/1) Scenario_Combo
    has mergeable toolbar (TOOLBARS_CONFIGURATION/1) View Main
    has view toolbar (TOOLBARS_CONFIGURATION/1) Essbase View Standard Limited
    has Excel template (GENERAL/1) IncStmt
```

You may run a reverse dependency search to determine what depends on a given artifact. For example, in the previous example we can see that the `IncStmt` view has a dependency on an Excel binary artifact of the same name (with type `GENERAL`). Therefore, in this case we would expect that if we search for what depends on this Excel template, we will at least see our view.

We can use the `depends-on` command to perform this search, such as the following:

```
dshell/SAMPLE:>depends-on IncStmt GENERAL 1
```

We may see something like the following output:

```
(GENERAL/1) IncStmt
    has Excel template (VIEW/1) IncStmt
```

Export Comments

You can view, export, and import comments. Viewing is useful to get a sense of what comments exist and which intersections an export/import will target. You can export comments to an XML file to be imported to a different tenant, database, or both. You may also specify which intersections should be specifically included or excluded in the import.

To view all comments in the current tenant, you can simply use the `view-comments` command:

```
view-comments
```

With output such as the following:

```
+-----------+-------------------------+-----------------------------------------------+
|Author     |Keys                     |Comment                                        |
+-----------+-------------------------+-----------------------------------------------+
|Jason Jones|Day: 2                   |Comment 1                                      |
|           |Month: December          |                                               |
|           |Row: 1                   |                                               |
|           |Year: 2016               |                                               |
|Jason Jones|Market: New York         |Test comment                                   |
|           |Measures: COGS           |                                               |
|           |Product: Root Beer       |                                               |
|           |Scenario: Variance       |                                               |
|           |Year: Jan                |                                               |
+-----------+-------------------------+-----------------------------------------------+
```

The `view-comments` command (as well as the export and import commands) also allow for specifying an include and exclude "key spec" or key specification. This lets you filter the comments to include/exclude such that all the comments meet **all** of the specified criteria. By way of a simple example, you may wish to export all comments that contain as part of their intersection the key of "Market" with a value of "New York". That can be specified as follows:

```
view-comments --include-key-spec "Market=New York"
```

Note that the whole key spec is in quotes since one of the values contains a space in it.

You may instead decide that you want to view/export all comments that simply *have* a key named Market, but with any value. This can be specified as follows:

```
view-comments --include-key-spec "Market="
```

You may want to specify just comments where the value is "West" but for whatever reason it doesn't matter what the key is:

```
view-comments --include-key-spec "=West"
```

You can also specify that comments should have multiple keys and values, such as those comments that have Market set to West but also where Measures is set to Sales:

```
view-comments --include-key-spec "Market=West,Measures=Sales"
```

Again, not that the exported comments must satisfy **all** of the specified key specs, not just some of them.

You may want to exclude certain combinations as well. This works similarly to the include key spec in that the excluded comments must match all of the specifications. For example:

```
view-comments --include-key-spec "Market" --exclude-key-spec "Market=New York,Year=Jan"
```

The preceding example would include comments that have Market as part of their key set (with any value), while also excluding those comments that have Market set to New York AND have Year set to Jan. Comments that have Market set to New York but that have Year set to something besides Jan will not be excluded.