Oracle® Enterprise Manager

Application Configuration Console CLI Reference

Release 5.3.2

E14656-02

September 2009

ORACLE®

Oracle Enterprise Manager Application Configuration Console CLI ReferenceRelease 5.3.2

E14656-02

Primary Author: James Garrison

Contents

Preface

This document is a reference for the Application Configuration Console command line interface (CLI). In addition to command synax and examples, it provides a complete reference for mvPath, a tool for or navigating the data hierarchy in Application Configuration Console.

Audience

This document is intended for technically saavy Application Configuration Console users who prefer command-line interaction with a software application.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at http://www.fcc.gov/cgb/consumerfacts/trs.html, and a list of phone numbers is available at http://www.fcc.gov/cgb/dro/trsphonebk.html.

Related Documents

For more information, see the following documents in the Application Configuration Console documentation set:

- *Release Notes*
- *Getting Started*
- *PCI Compliance*
- *Installation Guide*
- *Performance and Tuning Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Overview

This chapter introduces the Application Configuration Console Command Line Interface (CLI).

1.1 About the Command Line Interface

You can access the commands and data in Application Configuration Console through the graphical user interface in the Client or through the Command Line Interface. This reference document assumes you understand how to use the capabilities in the Client through the graphical user interface.

The command line interface (CLI) provides a low-level, programmatic interface to the same information that is managed through Clients. While you can perform many of the same operations in the CLI, the Client provides a richer command set and more visual feedback for the work you perform in Application Configuration Console.

Use the command line interface if:

- You want to repeat specific steps on numerous occasions. You can script repeated operations with the command line interface and replay them as you need them.
- You cannot access the Client.
- You prefer using command line interfaces over graphical user interfaces.

We recommend that you use the graphical user interface to review the data hierarchy in Application Configuration Console before accessing the data from the command line interface. The CLI lets you navigate the resource hierarchy like a directory tree, but it's easier to visualize the tree after you've seen it presented graphically.

Note: A DOS window cannot handle non-ASCII characters. Therefore, to display multibyte characters properly, run the CLI from a console window within a Client that is UTF-8-enabled, not in standalone mode.

1.2 Accessing the Command Line Interface

The CLI runs as a Python script that is installed in the eclipse directory where you installed the Application Configuration Console software. Use Python version 2.44, 2.54, or 2.61 to run the CLI.

The CLI requires JDK 1.6.x. If you have multiple versions of the JDK in your path and v1.6.x is not the first one, you must set the JDK_HOME property before starting the CLI.

1. Navigate to the following directory under the Client installation location, for example:

   ```
   C:\ProgramFiles\Oracle\oacc\client\runtime
   ```

2. Open a text editor and create a new file that contains this line:

   ```
   JDK_HOME=<your JDK1.6.x home>
   ```

3. Save the file as `cli.properties` in the runtime directory.

You can start the CLI from a command prompt. When you start the CLI, you will be prompted for your Application Configuration Console username and password. You must include your domain name at the beginning of your username, as in `mydomain\myname`.

```
> startcli.py
username: mydomain\myname
password: password
/Data =>>
```

The =>> is the CLI prompt, which also shows you the current context. The default context when you start the CLI is `/Data`. Within the CLI prompt, you can use the up and down arrow keys to cycle through all commands entered during a CLI session.

Note: You will not be able to start the CLI if there is a currently active Client session. There can be only a single CLI instance connected to the Core Server at any given time.

2
Commands

This chapter provides syntax and examples of CLI commands.

cd

Use this command to change the current context (location) in the resource hierarchy. (The command prompt always displays the current context.)

Usage:

`cd` *`newcontext`*

Argument	Definition
`newcontext`	An absolute or relative mvPath that becomes the new working context in the data hierarchy. If context contains spaces, you must enclose it in quotes. context is case-sensitive. All contexts in Application Configuration Console do not share a common root. You can use these root level contexts: ■ /Data - the root context of containers and configuration data. This equates to the My Workspace/Data folder for the user who logged in to the CLI. You cannot access the Public Workspace from the CLI. ■ `/Template`–the root context of templates ■ `/Authpack`–the root context of authentication packs ■ `/Resourcespec`–the root context of resource specifications ■ `/Group`–the root context of groups From a particular root context, you can switch to other root contexts or to relative contexts. In the container context, cd supports mvPath that evaluates to containers. You can use these shortcuts: `cd .. - the parent context`

Note: Application Configuration Console does not require unique container names; For example, you can have two folders named Folder1 in an environment. The CLI `cd` command cannot navigate past duplicate container names.

Examples

The following example uses the `cd` command to switch from the Data folder to the QA environment in the 401K project and then back to the Data folder.

```
/Data =>> ls
401K
Portal

/Data =>> cd 401K

/Data/401K =>> ls
Dev
QA

/Data/401K =>> cd QA

/Data/401K/QA =>> cd /Data

/Data =>> cd 401K/QA

/Data/401K/QA =>>
```

create

Use this command to create a new Application Configuration Console object in the current context, or to list the object types that can be created in the current context. You will be prompted for the values required to create the object.

Usage:

`create` [*`objectType`*]

Argument	Definition
`objectType`	The type of object to be created. The type must be valid for the current context. If `objectType` is not specified, the valid object types that can be created in the current context are listed.

After specifying a valid object type, you are prompted for any additional information needed to create the object. The UISpec for the object type determines the additional information needed and the text of the prompts.

Examples

The following example creates a new environment under a project.

```
/Data/401K =>> ls
Dev
QA
Staging

/Data/401K =>> create

Allowed type for creation:
Asset
Layer
Resource
Folder
Environment
Project
ResourceView

/Data/401K =>> create Environment
Environment Name: Production

/Data/401K =>> ls
Dev
QA
Staging
Production
```

This example shows the interactive prompting when you create an asset:

```
/Data/Bluefish =>> ls
Redfish
Dev
Asset1

"list" successful.
/Data/Bluefish =>> create asset
```

```
Asset Name:
==>>  Asset2
Auth Pack:(select an index from the following)
1.  solarisv120
2.  solarisv125

==>> 1
Resource spec:(select an index from the following)
1.  AM_IHS
2.  AM_WebSphere_Plugin
3.  AM_WebSphere

==>> 3
Base path:
==>>  /
/Data/Bluefish =>> ls
Redfish
Dev
Asset1
Asset2

"list" successful.
/Data/Bluefish =>>
```

delete

Use this command to delete the object of the specified type and name from the current context.

Usage:

`delete | del [objectType objectName]`

Argument	Definition
objectType	The type of object to be deleted. The type must be valid for the current context. If `objectType` is not specified, the valid object types that can be deleted in the current context are listed. You can delete authentication packs and groups if the current context allows it.
objectName	The name of the object to be deleted.

Examples

The following example deletes an environment from a project.

```
/Data/401K =>> ls
Dev
QA
Staging
Production

/Data/401K =>> delete environment Production

/Data/401K =>> ls
Dev
QA
Staging

/Data/401K =>>
```

exit

Use this command to end the CLI session and close the console view.

Usage:

```
exit
```

help

Use this command to display a list of available commands, or display the syntax and description of a command.

Usage:

`help` *command*

Argument	Definition
command	A command for which you want to see syntax and usage instructions.

Examples

The following example shows the help for the cd command.

```
/Data =>> help cd
  usage: cd [newcontext]

  Changes the CLI's current context.
The "root" level contexts can be
     /Data - the Container root context
     /Template - the Template root context
     /Authpack - the Authpack root context
     /Resourcespec - the Resourcespec root context
     /Group - the Group root context

   From a particular root context, you can switch to other root contexts
   or to relative contexts. In the container context cd supports mvpath that
   evaluates to containers.

  For example:   /Data =>> cd P1/WAS
   /Data/P1/WAS =>> cd ..
   /Data/P1 =>> cd /Authpack
   /Authpack =>> cd /
   /Data =>> cd P1/WAS
   /Data/P1/WAS =>> cd
   /Data =>> cd P1/WAS/config/cells/*/nodes
   /Data/P1/WAS/config/cells/solarisv120Network/nodes =>>
```

history

Use this command to display a list of commands that have been entered.

Usage:

```
history
```

Examples

The following example shows a list of commands previously entered during the session.

```
/Data =>> list
401K
Portal

/Data =>> cd 401K

/Data/401K =>> ls
Dev
QA
Staging

/Data/401K =>> history
list
cd 401K
ls
history

/Data/401K =>>
```

list

Use this command to list the children of the specified context.

Usage:

`list | ls [`*`context`*`]`

Argument	Definition
context	An absolute or relative mvPath of the context for which you want to list children. If context starts with / then it is an absolute path from one of the root contexts. All other paths are relative to the current context. If context is not specified, the contents of the current context are listed.

Examples

An absolute or relative mvPath of the context for which you want to list children. If context starts with / then it is an absolute path from one of the root contexts. All other paths are relative to the current context. If context is not specified, the contents of the current context are listed.

```
/Data =>> list
401K
Portal

/Data =>> cd 401K

/Data/401K =>> ls
Dev
QA
Staging

/Data/401K =>> cd /Groups

/Groups =>> ls
All Users
Administrators
DB Admins
Number of groups = 3
```

log

Use this command to turn CLI logging on or off. When enabled, all standard CLI output is directed to the `mvCli.log` file, located in the following folder:

```
%USERPROFILE%\Application Data\ApplicationConfigurationConsole
```

Where `%USERPROFILE%` defaults to `C:\Documents and Settings\u`*`sername.`*

Logging is off by default.

Usage:

`log` [*`flag`*]

Argument	Definition
`flag`	0 = disable logging 1 = enable logging

provision

Use this command to Writes the assets below the current context out to the external resources, using the authentication packs and resource specifications associated with those assets. It is the equivalent of the Provision command in the Application Configuration Console Client; all assets below the current context will be provisioned, not just assets in the current context.

Usage:

```
provision | pro["comment"]
```

Argument	Definition
comment	Text describing the changes made or the reason for the provision operation. Optional but recommended.

Examples

The following example shows all of the assets under the Development environment that includes the layers Web Server and Application Server being provisioned.

```
/Data/Portal View/Development =>> ls
Web Server
Application Server
"list" successful.
/Data/Portal View/Development =>> provision
com.mvalent.api.container.results.MvOperationSummary@130b993
com.mvalent.api.container.results.MvOperationSummary@1e799e6
com.mvalent.api.container.results.MvOperationSummary@28d853
End of provision.
/Data/Portal View/Development =>>
```

resync

Use this command to synchronize an asset or a configuration with the external resources by overwriting the Application Configuration Console data with the external files. This is the equivalent of the Update command in the Client and is valid for assets and configurations. You must cd to make an asset or a configuration the current context to run this command.

Usage:

```
resync
```

Examples

The following example shows the asset TestApp in the App Server container being updated.

```
/Data/Lab View/Test =>>
/Data/Lab View/Test =>> cd "App Server"
/Data/Lab View/Test/App Server =>> ls
TestApp"
list" successful.
/Data/Lab View/Test/App Server =>> cd TestApp
/Data/Lab View/Test/App Server/TestApp =>> resync
com.mvalent.api.container.results.MvOperationSummary@1c5eee0
End of resync.
/Data/Lab View/Test/App Server/TestApp =>>
```

runscript

Use this command to execute a registered script, either interactively or by reading input from a parameter dictionary.

Usage:

`runscript [-id scriptid | -id` *`scriptid`* `-param` *`paramdictionary`*`]`

- Typing runscript with no parameters displays an indexed list of scripts that can be run in the current context. You can select a script from the list and be prompted for input values.
- You can optionally specify a script ID to bypass the indexed list.
- You also can specify a script ID and a parameter dictionary to run a script without prompting. The keys to the dictionary have to be the control names and the value the data, for example:

```
 runscript -id SampleScript -param "{'sdi_text1':'val1',
'sdi_combo1':['mylabel','labelid1'],'sdi_subdialog1':{'sdi_text2':'val2',
'sdi_combo2':['mylabel','labelid2']}}"
```

One way to construct the param string is to create a PyDictionary object in the console and use its str representation as shown:

```
=>> param = {}
=>> param["sdi_text1"] = "val1"
=>> param["sdi_combo1"] = ["mylabel","labelid1"]
.
.
.
=>> str(param)
```

(You'll see a message after each command, but the input is still saved in the string.) Copy the output of the above line and paste it in the runscript command.

Examples

This example shows selecting a script from the indexed list and providing input values.

```
/Data/401K =>> runscript
List of available scripts:(select an index from the following)
1. SampleScript
2. AnotherScript

==>> 1

Sample Script

This sample script creates an organizational container.

Choose type of container to create:
1. Project
2. Environment
3. Layer
4. Folder

==>> 2
Name for container:

==>> QA

Succeeded

/Data/401K =>> ls
QA
```

This example runs the same script and passes in the values in a param dictionary. The CLI command prompt is not shown because this format would most likely be used in a script.

```
runscript -id SampleScript -param "{'containerType':'environment',
'containerName':'QA'}"
```

suppresswarning

Use this command to suppress the following unrecognized command warning that is displayed before passing the command to the Python shell:

```
Not a recognized mValent Integrity command. Passing it to the python shell. If you
see a secondary prompt and if you intended to type an mValent command press enter
to get the primary prompt else continue typing.
```

Usage:

```
suppresswarning
```

unlock

Use this command to force an unlock of a locked asset or configuration in the current context.

Usage:

```
unlock [commit | rollback]
```

Argument	Definition
commit	Save changes before unlocking the asset or configuration.
rollback	Discard changes before unlocking the asset or configuration. (Default)

3

mvPath

This chapter contains information on how to use mvPath. It includes complete mvPath syntax and copious examples of its use.

3.1 Using mvPath

Several CLI commands use mvPath expressions to refer to containers and properties.

mvPath is similar to XPath, but is designed specifically for navigating the data hierarchy in Application Configuration Console. If you're not familiar with XPath but have experience typing directory and file names at a DOS or UNIX command prompt, you should find mvPath intuitive.

All mvPath expressions refer to a context, which you can usually think of as the current location in the data hierarchy.

3.2 mvPath Reference

mvPath is an expression language for referring to objects in Application Configuration Console such as containers and properties. In Application Configuration Console you use mvPath when specifying variables and constraints, when using the command line interface, and when writing your own scripts.

The mvPath view of configuration data is simple: everything is a container except for properties and metadata. Projects, environments, layers, folders, assets, configurations, views, configuration elements -- they're all containers. Both containers and properties can have metadata, and properties and metadata can have values.

3.2.1 mvPath Constructors

You use mvPath to navigate up and down the hierarchy of containers. All mvPath expressions are evaluated in relation to a context. A context as a container in the data hierarchy (a contexts is always a container). The context for an mvPath expression is usually the current container. However, if an expression starts with / then the context is the data root.

mvPath expressions are constructed from a fairly short list of words and punctuation.

Container names = Refer to containers directly by name, with no special punctuation. For example, Asset1 would refer to a container named "Asset1" that is located in the current context.
Property names = Precede a property name with @. For example, @port refers to a property named "port."

Metadata names = Precede metadata names with a tilde (~). For example, ~http refers to a metadata named "http." Note that both containers and properties can have metadata.
Property and Metadata values = Set or test the values of properties and metadata with = (equals). For example, @port="80" refers to a property named "port" with the value "80."
. (period) = the current context. This is optional, except when preceding //
.. = go up to the parent container. Note that you cannot use this to navigate from one top-level project to another, such as ../P2 from P1.
/ = root or separator. If used at the beginning of an expression, the slash indicates the root of the Data folder. If used elsewhere in an expression, it separates the components of the path.
/MV_CONFIG = root of System Configuration folder. To address objects under the System Configuration folder, you must start the path with /MV_CONFIG.
// = recurse all descendants.
//- = recurse descendants, but don't recurse below a positive result.
^ = Match limiter. Will not allow the expression to match inside configurations. The search will only be in containers above configurations.
[] = contains or has. For example, E2[L2] refers to a container named "E2" that contains a child named "L2." E2[def()="D1"] refers to a container named "E2" that has a definition of "D1."
***** = wildcard for a complete container name, property name, or metadata name. Cannot be used to complete a partial name.
+ = concatenate mvPaths (must be strings, including name(), value() and literal strings). Do not use spaces on either side of the plus sign. Concatenation cannot be used inside predicates, for example: C1[@P1/value()="foo"+@P2/value]
value() = the value of a property or metadata
id() = the id of a container, property or metadata
name() = the name of a container, property or metadata
list("regex") = returns a list string that matches a pattern found in the supplied name() or value() operator.

Examples:

If a returned name() or value() is “SIBusMember_114678997643" then:

```
value()/list("^[a-zA-Z]*") returns "SIBusMember"
value()/list("[0-9]*$") returns "114678997643"
value()/list("M[a-z]*") returns "Member"
```

/slice(start:end) = returns a part of a string, where start is the (zero-based) number of the first character to return and end is the number of the last character to return. If end is a negative number, then it is counted from the right end of the string.

Examples:

If a returned value() is "ABCDEFG" then:

```
value()/slice("2:5") returns "CDE"
value()/slice("2:-2") returns "CDE"
value()/slice("2:") returns "CDEFG"
value()/slice(":5") returns "ABCDE
value()/slice("-3:") returns "EFG"
```

def() = the container or property definition. Possible values are the strings represented by the constants defined in `mvcontainer.ContainerDefinitions`. To get the actual string, type the following in the CLI console view:

```
import mvcontainer
 mvcontainer.ContainerDefinitions.<CONSTANT>
```

where *<CONSTANT>* is a containerDefinitions constant, for example:

```
mvcontainer.ContainerDefinitions.PROJECT
```

Possible constant values:

```
ASSET = http://mvalent.com/2003/container#asset
ASSET_RESOURCE_VIEW =
http://mvalent.com/2003/container#asset_resource_view
AUTHPACK_FOLDER = http://mvalent.com/2003/container#authpack_
folder
CONFIGURATION =
http://mvalent.com/2003/container#configuration
CONFIGURATION_VIEW =
http://mvalent.com/2003/container#configuration_view
CUSTOM_VIEW = http://mvalent.com/2003/container#custom_view
CUSTOM_VIEW_FOLDER =
http://mvalent.com/2003/container#custom_view_folder
ELEMENT_TREE = http://mvalent.com/2003/container#element_tree
ENVIRONMENT = http://mvalent.com/2003/container#environment
FILE_FOLDER = http://mvalent.com/2003/container#file_folder
FILE_TEMPLATE = http://mvalent.com/2003/container#file_
template
FOLDER = http://mvalent.com/2003/container#folder
HOST_FOLDER = http://mvalent.com/2003/container#host_folder
LAYER = http://mvalent.com/2003/container#layer
MV_CONFIG_ASSET = http://mvalent.com/2003/container#mv_
config_asset
PROJECT = http://mvalent.com/2003/container#project
RESOURCESPEC_FOLDER =
http://mvalent.com/2003/container#resourcespec_folder
```

type() = container type. Possible values are the strings represented by the constants defined in `mvcontainer.Type`. To get the actual string, type the following in the CLI console view:

```
regex(".*.xml")
```

where *<CONSTANT>* is the `mvcontainer.Type`, for example:

```
mvcontainer.Type.ORGANIZATION.uri()
```

Possible constant values:

```
ASSET = http://mvalent.com/2003/container#asset
ASSET_VIEW = http://mvalent.com/2003/container#asset_view
```

```
AUTHPACK_FOLDER = http://mvalent.com/2003/container#authpack_
folder
CONFIGURATION =
http://mvalent.com/2003/container#configuration
CONFIGURATION_VIEW =
http://mvalent.com/2003/container#configuration_view
ELEMENT_TREE = http://mvalent.com/2003/container#element_tree
FILE_TEMPLATE = http://mvalent.com/2003/container#file_
template
HOST_FOLDER = http://mvalent.com/2003/container#host_folder
MV_CONFIG = http://mvalent.com/2003/container#mv_config
ORGANIZATION = http://mvalent.com/2003/container#organization
RESOURCESPEC_FOLDER =
http://mvalent.com/2003/container#resourcespec_folder
```

regex() = regular expression. You can use regular expressions for pattern matching, such as:

```
import mvcontainer
 mvcontainer.Type.<CONSTANT>.uri()
```

to match all configuration names that end in .xml. Regular expressions can be quite powerful, but are beyond the scope of this reference page. For more information, look at regex sites on the Web, such as http://www.regular-expressions.info/.

\ = Escape character. You must precede the following special characters with a backslash if you need to use them in an mvPath expression: / ([ ] = +

Although there are only a few constructors, they are quite powerful when combined in mvPath expressions as shown in the examples below. When constructing an mvPath, you should be as specific as possible to increase speed and efficiency. Keep these recommendations in mind:

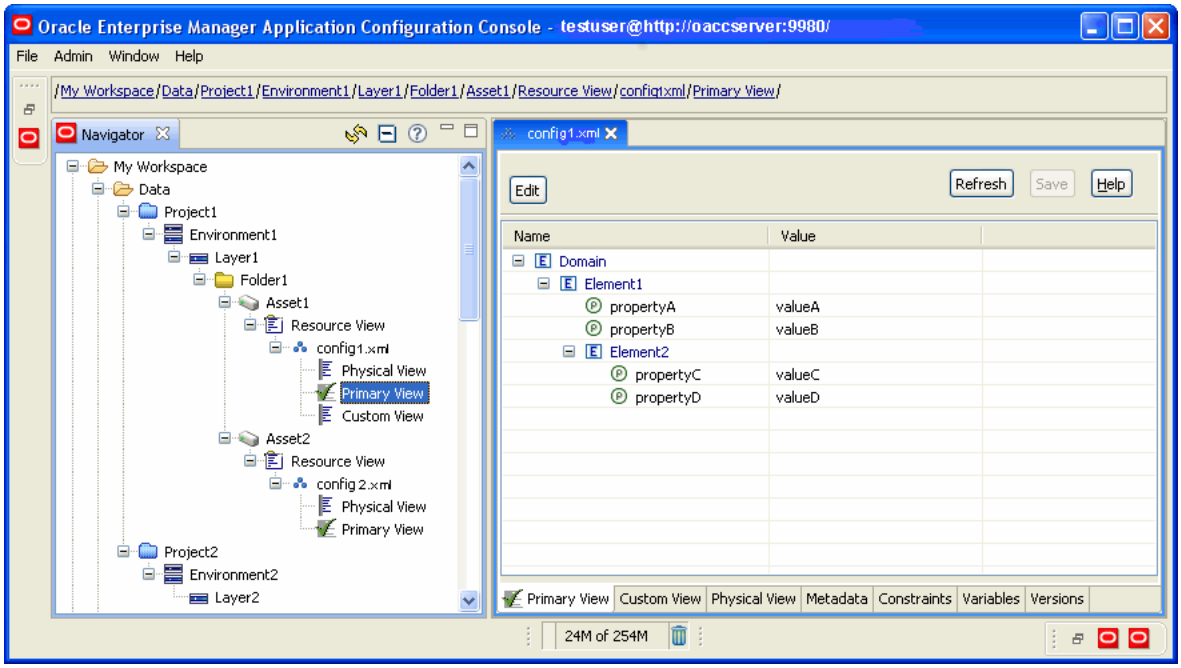
If the expression is not meant to find elements or properties, use //^ instead of //.

If the expression is looking for something under a specific child container, then use that child container name in the path instead of searching all child containers; that is, use: ./c1/c2 instead of .//c2.

Only start expressions with // if you don't know any part of the path to the desired object.

3.2.2 mvPath Examples

The examples below use a simple data hierarchy, represented graphically here in the Client Navigator view:



In addition to what is visible here, both config1.xml and propertyA have metadata.The context for all examples will be Folder1 that is highlighted in Project1.

3.2.2.1 Containers

Refer to a container by location:

Asset1/*
Returns all child containers of Asset1. In this example, it would just return Resource View.

.//*
Returns the current container and all its descendant containers.

..
Returns Layer1, the parent of the current container.

../../*
Returns Environment1 (and any siblings if it had any).

Refer to a container by name:

Asset1
Returns Asset1 under the current context, which is Folder1.

Asset1/Resource View/config1.xml
Returns config1.xml under Asset1.

Asset1/Resource View/config1.xml/Primary View/Element1
Returns Element1 in the Primary View of config1.xml, which is under Resource View of Asset1. Note that both views and elements are containers in the hierarchy. You must specify a view to access the elements and properties in a configuration.

//config1.xml
Returns all of the containers named "config1.xml" in the entire system.

Asset1//^Primary View
Returns all containers named "Primary View" that are descendants of Asset1 but are not configurations or under configurations. There are no matches in the example, so nothing would be returned.

/Project1
Returns the root project named "Project1."

//Primary View
Returns all the containers named "Primary View" in the entire system.

.//Primary View
Returns all the descendants of the current container named "Primary View" plus the container itself if it is named "Primary View."

../Layer1
Returns the parent container.

./"Asset"+"2"
Returns Asset2.

//-Layer1
Returns Layer1, but does not look below Layer1 for any other containers named Layer1.

Refer to a container by something it contains:

Asset1[Resource View/config1.xml]
Returns Asset1 if it has a child container named "config1.xml" in its Resource View.

Asset1/Resource View/config1.xml[CustomView]
Returns config1.xml if it is a child of Asset1's Resource View and if it has a child container named "CustomView."

Asset1[Resource View/config1.xml/CustomView]
Returns Asset1 if it has a child container named "config1.xml" in its Resource View that has a child container named "CustomView."

Asset1[Resource View/config1.xml/~metadata1]
Returns Asset1 if it has a child container named "config1.xml" in its Resource View that has metadata named "metadata1."

Asset1/Resource View/config1.xml[~metadata1]
Returns config1.xml if it has metadata named "metadata1."

Asset1/Resource View/config1.xml[~metadata1="value1"]
Returns config1.xml if it has metadata named "metadata1" with a value of "value1."

Asset1/Resource View/config1.xml[Primary View/@propertyA]
Returns config1.xml if it is a child of Asset1 and has a child container named "Primary View" which contains a property named "propertyA."

Asset1/Resource View/*[Primary View/@propertyA]
Returns any containers that are children of Asset1's Resource View and have a child container named "Primary View" which contains a property named "propertyA."

Asset1/*[Primary View/@propertyA="valueA"]
Returns any containers that are children of Asset1's Resource View and have a child container named "Primary View" which contains a property named "propertyA" with value "valueA."

./*/*/*[Primary View]
Returns both config1.xml and config2.xml because they both match the wildcard path and they both contain a container named "Primary View."

Asset1//[@propertyA="valueA"]
Returns the container Element1, because it contains "propertyA" with value "valueA". Note that this returns the container (which is an element in this case), not the property.

Asset1//[@propertyA="valueA"][@propertyB="valueB"]
Returns the container Element1, because it contains "propertyA" with value "valueA" and "propertyB" with "valueB."

Refer to a container by type:

Asset1/*[type()="organization"]
Returns the child containers of Asset1 that have a container type of "organization."

Refer to a container by definition:

Asset1/*[def()="D1"]
Returns all containers that are children of Asset1 and have a definition of "D1."

Refer to the name of a container:

Asset1/name()
Returns the name of Asset1 under the current context.

Asset1/*/name()
Returns the names of all containers directly under Asset1.

Refer to the type of a container:

Asset1/type()
Returns the container type of Asset1.

Refer to the ID of a container:

Asset1/id()
Returns the id of Asset1.

3.2.2.2 Properties

Refer to a property by location:

Asset1/Resource View/config1.xml/Primary View/Element1/@*
Returns all properties that are in Element1 of the Primary View of config1.xml.

Asset1/Resource View/config1.xml/Primary View/*/@*
Returns all properties that are in the Primary View of config1.xml.

Refer to a property by name:

Asset1/Resource View/config1.xml/Primary View/Element1/@propertyA
Returns propertyA in Element1 of the configuration. Note that a view is considered a container in the hierarchy and must be included in the mvPath to reach elements and properties. In other words, properties are contained in elements (which are containers) which are contained in views.

Asset1//@propertyA
Returns all properties named "propertyA" underneath Asset1. With the sample data, this returns the same property as the previous example because there aren't any other properties with that name.

Refer to a property by value:

Asset1/Resource View/config1.xml/Primary View/Element1/@*[value()="valueA"]
Returns all properties in Element1 with value "valueA."

Refer to a property by name and value:

Asset1//@propertyA="valueA"
Returns all properties named "propertyA" with value "valueA" that are descendants of Asset1.

Refer to a property by metadata name:

Asset1/Resource View/config1.xml/Primary View/Element1/@propertyA[~metadataA]
Returns propertyA in Element1if it has metadata named "metadataA."

Asset1//*/@*[~metadataA]
Returns all properties that are descendants of Asset1 and that have metadata named "metadataA."

Refer to a property by metadata name and value:

Asset1//*/@*[~metadataA="valueA"]
Returns all properties that have metadata named "metadataA" with value "valueA" that are descendants of Asset1.

Refer to a property by definition:

Asset1/Resource View/config1.xml/Primary View/Element1/@*[def()="port"]
Returns all properties in Element1 that have a definition of "port."

Refer to a property by another property in the same element:

Asset1/Resource View/config1.xml/Primary View/Element1[@propertyB="valueB"]/@propertyA
Returns propertyA in Element1 if Element1 contains propertyB with value "valueB."

Refer to the name of a property:

Asset1/Resource View/config1.xml/Primary View/Element1/@propertyA/name()
Returns the name of propertyA.

Asset1/Resource View/config1.xml/Primary View/Element1/@*/name()
Returns the names of all properties in Element1.

Refer to the value of a property:

Asset1/Resource View/config1.xml/Primary View/Element1/@propertyA/value()
Returns the value of propertyA in Element1.

Refer to the definition of a property:

Asset1/Resource View/config1.xml/Primary View/Element1/@propertyA/def()
Returns the property definition of the property named "propertyA." (Properties are linked to definitions in the property dictionaries.)

Refer to the ID of a property:

Asset1/Resource View/config1.xml/Primary View/Element1/@propertyA/id()
Returns the id of propertyA.

3.2.2.3 Metadata

Refer to metadata by location:

Asset1/Resource View/config1.xml/~*
Returns all container metadata of the config1.xml container.

Asset1/Resource View/config1.xml/Primary View/Element1/propertyA/~*
Returns all property metadata of the property named "propertyA."

Refer to metadata by name:

Asset1/Resource View/config1.xml/Primary View/Element1/@propertyA/~metadataA
Returns metadataA of propertyA in Element1 of the configuration.

Asset1/Resource View/config1.xml/~metadata1
Returns metadata1 of the config1.xml container.

Refer to metadata by value:

Asset1/Resource View/config1.xml/~[value()="value1"]
Returns all metadata in the config1.xml container with value "value1."

Refer to metadata by name and value:

Asset1//~metadata1="value1"
Returns all metadata named "metadata1" with value "value1" that are descendants of Asset1.

Refer to metadata names:

Asset1/Resource View/config1.xml/Primary View/Element1/@propertyA/~metadataA/name()
Returns the name of metadataA on propertyA.

Asset1/Resource View/config1.xml/~*/name()
Returns the names of all metadata in config1.xml.

Refer to metadata values:

Asset1/Resource View/config1.xml/Primary View/Element1/@propertyA/~metadataA/value()
Returns the value of metadataA on propertyA in Element1.

Asset1/Resource View/config1.xml/~metadata1/value()
Returns the value of metadata1 in config1.xml.

Refer to metadata ID:

Asset1/Resource View/config1.xml/Primary View/Element1/@propertyA/~metadataA/id()
Returns the id of metadataA.

3.2.2.4 Connectors (Hosts, Authentication Packs, Resource Specifications)

Syntax:

```
[objectPath]operator("objectName")
```

Where:

`objectPath` is the optional path to `objectName`. The path references the root, that is, it begins with `/`, as in `/F1/F2/F3/…`

`operator` is one of the following:

```
authpack
host
resourcespec
```

`objectName` is the optional name of the object to resolve. If no name is specified, all objects of the target type are returned for the specified context.

Refer to a connector by name:

authpack("passport")
Returns an authentication pack named passport from the global name space.

Refer to a connector by name and location:

/F1/F2/F3/resourcespec("xmlConfig")
Returns a resource specification named xmlConfig under the folder hierarchy `/F1/F2/F3/`.

Refer to all connectors of a target type:

host()
Returns all hosts in the global name space.

/F1/F2/F3/host()
Returns all hosts under the folder hierarchy `/F1/F2/F3/`.

3.2.2.5 Strings

Return a string:

"Your text string here."
Returns the string "Your text string here."