Red Hat JBoss Enterprise Application Platform 7.0 Management CLI Guide

---

For Use with Red Hat JBoss Enterprise Application Platform 7

Red Hat Customer Content Services

Red Hat JBoss Enterprise Application Platform 7.0 Management CLI Guide

For Use with Red Hat JBoss Enterprise Application Platform 7

Legal Notice

Abstract

This provides information about how to to use the Management CLI to administer Red Hat JBoss Enterprise Application Platform.

Table of Contents

CHAPTER 1. MANAGEMENT CLI OVERVIEW

The management command-line interface (CLI) is a command-line administration tool for JBoss EAP.

Use the management CLI to start and stop servers, deploy and undeploy applications, configure system settings, and perform other administrative tasks. Operations can be performed in batch mode, allowing multiple tasks to be run as a group.

Many common terminal commands are available, such as **`ls`**, **`cd`**, and **`pwd`**. The management CLI also supports tab completion.

CHAPTER 2. GETTING STARTED WITH THE MANAGEMENT CLI

The management CLI is included with the JBoss EAP distribution. Once you launch the management CLI, you can connect to a running server instance or managed domains to perform management operations.

2.1. LAUNCH THE MANAGEMENT CLI

You can launch the management CLI by running the **jboss-cli** script provided with JBoss EAP.

```
$ EAP_HOME/bin/jboss-cli.sh
```

Note

For Windows Server, use the **EAP_HOME\bin\jboss-cli.bat** script to launch the management CLI.

See Connect to the Server for details on launching the management CLI and connecting to the server in one step using the **--connect** argument.

For a complete listing of all available **jboss-cli** script arguments and their purposes, use the **--help** argument or see the Management CLI Startup Arguments section.

2.2. CONNECT TO THE SERVER

You can connect to a running standalone server or managed domain by using the **connect** command.

```
connect
```

The default host and port configuration is **localhost:9990**. If the server is listening on a different host and port, then these need to be provided to the **connect** command.

```
connect 192.168.0.1:9991
```

You can also launch the management CLI and connect to the server in one step using the **--connect** argument (and the **--controller** argument if necessary).

```
$ EAP_HOME/bin/jboss-cli.sh --connect --controller=192.168.0.1:9991
```

In JBoss EAP 7.0, to connect using the **http-remoting** protocol, use:

```
connect http-remoting://192.168.0.1:9999
```

2.3. CHANGE THE CURRENT PATH

You can change to a different node path by using the **cd** command and providing the desired path.

When the management CLI is first launched, it is at the root level (`/`).

```
cd /subsystem=datasources
cd data-source=ExampleDS
```

You can show the full path of the current node using the **`pwd`** command.

```
pwd
```

```
/subsystem=datasources/data-source=ExampleDS
```

2.4. LIST CONTENTS

You can list the contents of a particular node path by using the **`ls`** command. If the path ends on a node name, that resource's attributes will be listed as well.

The below example navigates the **`standard-sockets`** socket binding group and then lists its contents.

```
cd /socket-binding-group=standard-sockets
ls -l
```

```
ATTRIBUTE         VALUE                                  TYPE
default-interface public                                 STRING
name              standard-sockets                       STRING
port-offset       ${jboss.socket.binding.port-offset:0} INT

CHILD                                       MIN-OCCURS MAX-OCCURS
local-destination-outbound-socket-binding  n/a        n/a
remote-destination-outbound-socket-binding n/a        n/a
socket-binding                              n/a        n/a
```

The same result can be achieved from anywhere in the resource tree hierarchy by specifying the node path to the **`ls`** command.

```
ls -l /socket-binding-group=standard-sockets
```

```
ATTRIBUTE         VALUE                                  TYPE
default-interface public                                 STRING
name              standard-sockets                       STRING
port-offset       ${jboss.socket.binding.port-offset:0} INT

CHILD                                       MIN-OCCURS MAX-OCCURS
local-destination-outbound-socket-binding  n/a        n/a
remote-destination-outbound-socket-binding n/a        n/a
socket-binding                              n/a        n/a
```

You can also use the **`--resolve-expressions`** parameter to resolve the expressions of the returned attributes to their value on the server.

```
ls -l /socket-binding-group=standard-sockets --resolve-expressions
```

```
ATTRIBUTE         VALUE            TYPE
default-interface public           STRING
name              standard-sockets STRING
port-offset       0                INT

CHILD                                        MIN-OCCURS MAX-OCCURS
local-destination-outbound-socket-binding    n/a        n/a
remote-destination-outbound-socket-binding   n/a        n/a
socket-binding                               n/a        n/a
```

In this example, the **`port-offset`** attribute shows its resolved value (**`0`**) instead of the expression (**`${jboss.socket.binding.port-offset:0}`**).

2.5. GETTING HELP

The management CLI provides several ways for you to get help with using the management CLI.

- View general help on using the management CLI.

  ```
  help
  ```

  This provides detailed help with launching, navigating, and generating operation requests.

- View help for a particular command.

  ```
  COMMAND_NAME --help
  ```

  This provides the usage, description, and arguments for the particular command.

- View the list of available commands,

  ```
  help --commands
  ```

  

  Note

  Commands that require a connection to either a standalone server or domain controller will not appear in the list unless the connection has been established.

See the Management CLI Commands section for a listing of management CLI commands.

2.6. QUIT THE MANAGEMENT CLI

You can quit the management CLI by entering the **`quit`** command.

```
quit
```

CHAPTER 3. CREATING AND EXECUTING REQUESTS

JBoss EAP configuration is presented as a hierarchical tree of addressable resources, each offering their own set of operations. Management CLI operation requests allow for low-level interaction with the management model and provide a controlled way to edit server configurations.

Operation requests use the following format:

```
/NODE_TYPE=NODE_NAME:OPERATION_NAME(PARAMETER_NAME=PARAMETER_VALUE)
```

An operation request consists of three parts:

address

The address specifies the resource node on which to perform the operation. *NODE_TYPE* maps to an element name and *NODE_NAME* maps to that element's **name** attribute in the configuration XML. Each level of the resource tree is separated by a slash (`/`).

operation name

The operation to perform on the resource node. It is prefixed with a colon (`:`).

parameters

The set of required or optional parameters that vary depending on the operation. They are contained within parentheses (`()`).

Construct an Operation Request

1. Determine the address

   You can reference the XML configuration files (**`standalone.xml`**, **`domain.xml`**, or **`host.xml`**) to assist in determining the required address. You can also use tab completion to view the available resources.

   Below are several common addresses for resources at the root (`/`) level.

   - **`/deployment=DEPLOYMENT_NAME`** - Deployment configurations.
   - **`/socket-binding-group=SOCKET_BINDING_GROUP_NAME`** - Socket binding configurations.
   - **`/interface=INTERFACE_NAME`** - Interface configurations.
   - **`/subsystem=SUBSYSTEM_NAME`** - Subsystem configuration when running as a standalone server.
   - **`/profile=PROFILE_NAME/subsystem=SUBSYSTEM_NAME`** - Subsystem configuration for the selected profile when running in a managed domain.
   - **`/host=HOST_NAME`** - Server configuration for the selected host when running in a managed domain.

   The below address is for the **`ExampleDS`** datasource.

   ```
   /subsystem=datasources/data-source=ExampleDS
   ```

2. Determine the operation

   The available operations differ for each type of resource node. You can use the **`:read-operation-names`** operation on a resource address to view the available operations. You can also use tab completion.

   Use the **`:read-operation-description`** operation to get information on a particular operation for a resource.

   The below operation (once the appropriate parameters are included) will set the value of an attribute for the **`ExampleDS`** datasource.

   ```
   /subsystem=datasources/data-source=ExampleDS:write-attribute
   ```

3. Determine the parameters

   Each operation has its own set of available parameters. If you attempt to perform an operation without a required parameter, you will receive an error message that the parameter cannot be **`null`**.

   Multiple parameters are separated by commas (**`,`**). If an operation does not have any parameters, then the parentheses are optional.

   Use the **`:read-operation-description`** operation on a resource, passing in the operation name, to determine the required parameters for that operation. You can also use tab completion to list the available parameters.

   The below operation disables the **`ExampleDS`** datasource by setting its **`enabled`** attribute to **`false`**.

   ```
   /subsystem=datasources/data-source=ExampleDS:write-
   attribute(name=enabled,value=false)
   ```

Once entered, the management interface will perform the operation request on the server configuration. Depending on the operation request, you will receive output to the terminal that contains the outcome and the result or response of the operation.

The following response from disabling the **`ExampleDS`** datasource shows that the operation was successful and requires a reload of the server in order to take effect.

```
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

You can use the **`read-attribute`** operation to read the value of the **`ExampleDS`** datasource's **`enabled`** attribute.

```
/subsystem=datasources/data-source=ExampleDS:read-
attribute(name=enabled)
```

The following response shows that the operation was successful and that the value of **`enabled`** is **`false`**.

```
{
    "outcome" => "success",
    "result" => false,
}
```

3.1. DISPLAY RESOURCE VALUES

You can use the **`read-resource`** operation to view a resource's attribute values.

```
:read-resource
```

You can specify parameters to provide complete information about child resources, recursively. You can also specify parameters to include runtime attributes, resolve expressions, and include aliases. Use **`read-operation-description(name=read-resource)`** to see the description of all available parameters for **`read-resource`**.

The following example reads the attributes for a deployment. It includes details such as the deployment name, whether it is enabled, and the last time it was enabled.

```
/deployment=DEPLOYMENT_NAME:read-resource
{
    "outcome" => "success",
    "result" => {
        ...
        "enabled" => true,
        "enabled-time" => 1453929902598L,
        "enabled-timestamp" => "2016-01-27 16:25:02,598 EST",
        "name" => "DEPLOYMENT_NAME",
        "owner" => undefined,
        "persistent" => true,
        "runtime-name" => "DEPLOYMENT_NAME",
        "subdeployment" => undefined,
        "subsystem" => {
            "undertow" => undefined,
            "logging" => undefined
        }
    }
}
```

Include Runtime Attributes

The **`include-runtime`** parameter can be used to retrieve runtime attributes.

The following example reads the attributes for a deployment. In addition to persistent attributes, it also includes runtime attributes, such as the deployment status and the last time it was disabled.

```
/deployment=DEPLOYMENT_NAME:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        ...
        "disabled-time" => undefined,
        "disabled-timestamp" => undefined,
        "enabled" => true,
```

```
        "enabled-time" => 1453929902598L,
        "enabled-timestamp" => "2016-01-27 16:25:02,598 EST",
        "name" => "DEPLOYMENT_NAME",
        "owner" => undefined,
        "persistent" => true,
        "runtime-name" => "DEPLOYMENT_NAME",
        "status" => "OK",
        "subdeployment" => undefined,
        "subsystem" => {
            "undertow" => undefined,
            "logging" => undefined
        }
    }
}
```

Read Child Resources Recursively

The **`recursive`** parameter can be used to retrieve attributes recursively from child resources.

The following example reads the attributes for a deployment. In addition to the resource's own attributes, it recursively returns the attributes for its child resources, such as the Undertow subsystem configuration.

```
/deployment=DEPLOYMENT_NAME:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        ...
        "enabled" => true,
        "enabled-time" => 1453929902598L,
        "enabled-timestamp" => "2016-01-27 16:25:02,598 EST",
        "name" => "DEPLOYMENT_NAME",
        "owner" => undefined,
        "persistent" => true,
        "runtime-name" => "DEPLOYMENT_NAME",
        "subdeployment" => undefined,
        "subsystem" => {
            "undertow" => {
                "context-root" => "/test",
                "server" => "default-server",
                "virtual-host" => "default-host",
                "servlet" => undefined,
                "websocket" => undefined
            },
            "logging" => {"configuration" => undefined}
        }
    }
}
```

Exclude Default Values

The **`include-defaults`** parameter can be used to show or hide default values when reading attributes for a resource. This is **`true`** by default, meaning that default values will be shown when using the **`read-resource`** operation.

The following example uses the **`read-resource`** operation on the Undertow subsystem.

```
/subsystem=undertow:read-resource
{
    "outcome" => "success",
    "result" => {
        "default-security-domain" => "other",
        "default-server" => "default-server",
        "default-servlet-container" => "default",
        "default-virtual-host" => "default-host",
        "instance-id" => expression "${jboss.node.name}",
        "statistics-enabled" => false,
        "buffer-cache" => {"default" => undefined},
        "configuration" => {
            "filter" => undefined,
            "handler" => undefined
        },
        "server" => {"default-server" => undefined},
        "servlet-container" => {"default" => undefined}
    }
}
```

The following example also uses the **`read-resource`** operation on the Undertow subsystem, but sets the **`include-defaults`** parameter to **`false`**. Several attributes, such as **`statistics-enabled`** and **`default-server`**, now display **`undefined`** as their value instead of the default value.

```
/subsystem=undertow:read-resource(include-defaults=false)
{
    "outcome" => "success",
    "result" => {
        "default-security-domain" => undefined,
        "default-server" => undefined,
        "default-servlet-container" => undefined,
        "default-virtual-host" => undefined,
        "instance-id" => undefined,
        "statistics-enabled" => undefined,
        "buffer-cache" => {"default" => undefined},
        "configuration" => {
            "filter" => undefined,
            "handler" => undefined
        },
        "server" => {"default-server" => undefined},
        "servlet-container" => {"default" => undefined}
    }
}
```

Resolve Expressions

The **`resolve-expressions`** parameter can be used to resolve the expressions of the returned attributes to their value on the server.

The following example reads the attributes for a deployment. The **`instance-id`** attribute shows its resolved value (**`test-name`**) instead of the expression (**`${jboss.node.name}`**).

```
/subsystem=undertow:read-resource(resolve-expressions=true)
{
    "outcome" => "success",
    "result" => {
        "default-security-domain" => "other",
        "default-server" => "default-server",
        "default-servlet-container" => "default",
        "default-virtual-host" => "default-host",
        "instance-id" => "test-name",
        "statistics-enabled" => false,
        "buffer-cache" => {"default" => undefined},
        "configuration" => {
            "filter" => undefined,
            "handler" => undefined
        },
        "server" => {"default-server" => undefined},
        "servlet-container" => {"default" => undefined}
    }
}
```

3.2. DISPLAY RESOURCE DESCRIPTIONS

You can use the **`read-resource-description`** operation to a description about a resource and its attributes.

```
:read-resource-description
```

You can specify parameters to provide complete descriptions about child resources, recursively. You can also specify parameters to include details of the resource's operations and notifications. Use **`read-operation-description(name=read-resource-description)`** to see the description of all available parameters for **`read-resource-description`**.

The following example displays the attribute details for a buffer cache.

```
/subsystem=undertow/buffer-cache=default:read-resource-description
{
    "outcome" => "success",
    "result" => {
        "description" => "The buffer cache used to cache static
content",
        "attributes" => {
            "buffer-size" => {
                "type" => INT,
                "description" => "The size of an individual buffer",
                "expressions-allowed" => true,
                "nillable" => true,
                "default" => 1024,
                "min" => 0L,
                "max" => 2147483647L,
                "access-type" => "read-write",
                "storage" => "configuration",
                "restart-required" => "resource-services"
            },
            "buffers-per-region" => {
```

```
                "type" => INT,
                "description" => "The numbers of buffers in a region",
                "expressions-allowed" => true,
                "nillable" => true,
                "default" => 1024,
                "min" => 0L,
                "max" => 2147483647L,
                "access-type" => "read-write",
                "storage" => "configuration",
                "restart-required" => "resource-services"
            },
            "max-regions" => {
                "type" => INT,
                "description" => "The maximum number of regions",
                "expressions-allowed" => true,
                "nillable" => true,
                "default" => 10,
                "min" => 0L,
                "max" => 2147483647L,
                "access-type" => "read-write",
                "storage" => "configuration",
                "restart-required" => "resource-services"
            }
        },
        "operations" => undefined,
        "notifications" => undefined,
        "children" => {}
    }
}
```

3.3. DISPLAY AN ATTRIBUTE VALUE

You can use the **`read-attribute`** operation to view the current value of a single attribute.

```
:read-attribute(name=ATTRIBUTE_NAME)
```

The following example displays the log level for the root logger by reading the **`level`** attribute.

```
/subsystem=logging/root-logger=ROOT:read-attribute(name=level)
{
    "outcome" => "success",
    "result" => "INFO"
}
```

One advantage of using the **`read-attribute`** operation is the ability to expose the current runtime value of an attribute.

```
/interface=public:read-attribute(name=resolved-address)
{
    "outcome" => "success",
    "result" => "127.0.0.1"
}
```

The **resolved-address** attribute is a runtime attribute. This attribute is not displayed when using the **read-resource** operation on the public interface unless you pass in the **include-runtime** parameter. And even then, it is displayed with the rest of the resource's other attributes.

You can also use the **include-defaults** and **resolve-expressions** parameters. See Display Resource Values for details on these parameters.

3.4. UPDATE AN ATTRIBUTE

You can use the **write-attribute** operation to update the value of an attribute for a resource.

```
:write-attribute(name=ATTRIBUTE_NAME, value=ATTRIBUTE_VALUE)
```

The following example disables the deployment scanner by setting the **scan-enabled** attribute to **false**.

```
/subsystem=deployment-scanner/scanner=default:write-
attribute(name=scan-enabled,value=false)
{"outcome" => "success"}
```

The response from the operation request shows that it was successful. You can also confirm the result by using the **read-attribute** operation to read the **scan-enabled** attribute, which now shows as **false**.

```
/subsystem=deployment-scanner/scanner=default:read-attribute(name=scan-
enabled)
{
    "outcome" => "success",
    "result" => false
}
```

3.5. UNDEFINE AN ATTRIBUTE

You can set the value of an attribute to **undefined**. If this attribute has a default value, that will be used.

The following example undefines the **level** attribute for the root logger.

```
/subsystem=logging/root-logger=ROOT:undefine-attribute(name=level)
```

The default value for the **level** attribute is **ALL**. You can see that this default is used when performing the **read-resource** operation.

```
/subsystem=logging/root-logger=ROOT:read-resource
{
    "outcome" => "success",
    "result" => {
        "filter" => undefined,
        "filter-spec" => undefined,
        "handlers" => [
            "CONSOLE",
            "FILE"
```

```
        ],
        "level" => "ALL"
    }
}
```

To view the resource without the default values being read, you must use set the **`include-defaults`** parameter to **`false`**. You can now see that the value of **`level`** is **`undefined`**.

```
/subsystem=logging/root-logger=ROOT:read-resource(include-
defaults=false)
{
    "outcome" => "success",
    "result" => {
        "filter" => undefined,
        "filter-spec" => undefined,
        "handlers" => [
            "CONSOLE",
            "FILE"
        ],
        "level" => undefined
    }
}
```

3.6. DISPLAY OPERATION NAMES

You can use the **`read-operation-names`** list the available operations for a given resource.

```
:read-operation-names
```

The following example lists the available operations to perform on a deployment.

```
/deployment=DEPLOYMENT_NAME:read-operation-names
{
    "outcome" => "success",
    "result" => [
        "add",
        "deploy",
        "list-add",
        "list-clear",
        "list-get",
        "list-remove",
        "map-clear",
        "map-get",
        "map-put",
        "map-remove",
        "query",
        "read-attribute",
        "read-attribute-group",
        "read-attribute-group-names",
        "read-children-names",
        "read-children-resources",
        "read-children-types",
        "read-operation-description",
        "read-operation-names",
```

```
        "read-resource",
        "read-resource-description",
        "redeploy",
        "remove",
        "undefine-attribute",
        "undeploy",
        "whoami",
        "write-attribute"
    ]
}
```

Use the **`read-operation-description`** operation to display an operation description.

3.7. DISPLAY AN OPERATION DESCRIPTION

You can use the **`read-operation-description`** operation to display a description of a certain operation for a resource. This also includes parameter descriptions and which parameters are required.

```
:read-operation-description(name=OPERATION_NAME)
```

The following example provides the description and parameter information for the **add** operation on a system property.

```
/system-property=SYSTEM_PROPERTY:read-operation-description(name=add)
{
    "outcome" => "success",
    "result" => {
        "operation-name" => "add",
        "description" => "Adds a system property or updates an existing
one.",
        "request-properties" => {"value" => {
            "type" => STRING,
            "description" => "The value of the system property.",
            "expressions-allowed" => true,
            "required" => false,
            "nillable" => true,
            "min-length" => 0L,
            "max-length" => 2147483647L
        }},
        "reply-properties" => {},
        "read-only" => false,
        "runtime-only" => false
    }
}
```

3.8. ADD A VALUE WITH SPECIAL CHARACTERS

Occasionally when creating management CLI requests, you may need to add a value that contains special characters. Certain special characters, such as those used in the syntax of a management CLI request, must be entered in a particular manner.

In many cases, but not all, enclosing the value in double quotes (**""**) is sufficient. If you are unsure whether your special character was accepted properly, be sure to read the attribute or resource after adding the value to verify that it was saved correctly.

See the sections below for information on how to process the following special characters.

- Whitespace
- Quotation Marks
- Commas
- Parentheses
- Braces
- Diacritic Marks

Whitespace

By default, whitespace is trimmed from values added through the management CLI. You can include a space in a value by enclosing the value in double quotes (**""**) or braces (**{}**), or by escaping it using a forward slash (****).

```
/system-property=test1:add(value="Hello World")
/system-property=test2:add(value={Hello World})
/system-property=test3:add(value=Hello\ World)
```

This will set the value to **`Hello World`**.

Quotation Marks

Double quotation marks (**"**) are not allowed in JBoss EAP configuration values. You can use a single quotation mark (**'**) in a value by enclosing the value in double quotes (**""**) or by escaping it using a forward slash (****).

```
/system-property=test1:add(value="It's")
/system-property=test2:add(value=It\'s)
```

This will set the value to **`It's`**.

Commas

You can use a comma (**,**) in a value by enclosing the value in double quotes (**""**).

```
/system-property=test:add(value="Last,First")
```

This will set the value to **`Last,First`**.

Parentheses

You can include parentheses (**()**) in a value by enclosing the value in double quotes (**""**) or braces (**{}**), or by escaping the parenthesis using a forward slash (****).

```
/system-property=test1:add(value="one(1)")
/system-property=test2:add(value={one(1)})
/system-property=test3:add(value=one\(1\))
```

This will set the value to **one(1)**.

Braces

You can include braces (**{}**) in a value by enclosing the value in double quotes (**""**).

```
/system-property=test:add(value="{braces}")
```

This will set the value to **{braces}**.

Diacritic Marks

Diacritic marks, such as **ñ**, **ř**, or **ý**, can be used when adding a value using the management CLI.

```
/system-property=test1:add(value=Año)
```

However, do not enclose the value in double quotes (**""**). This can cause the diacritic mark to be replaced with a question mark (**?**). If the value has whitespace that needs to be maintained, instead enclose the value in braces (**{}**) or escape the space with a forward slash (****).

```
/system-property=test2:add(value={Dos años})
/system-property=test3:add(value=Dos\ años)
```

This will set the value to **Dos años**.

3.9. USE IF-ELSE CONTROL FLOW

The management CLI supports **if**-**else** control flow, which allows you to choose which set of commands and operations to execute based on a condition. The **if** condition is a boolean expression which evaluates the response of the management command or operation specified after the **of** keyword.

Expressions can contain any of the following items:

- Conditional operators (**&&**, **||**)
- Comparison operators (**>**, **>=**, **<**, **<=**, **==**, **!=**)
- Parentheses to group and prioritize expressions

Note

The use of nested **if**-**else** statements is not supported.

The below example attempts to read the system property **test**. If **outcome** is not **success** (meaning that the property does not exist), then the system property will be added and set to **true**.

```
if (outcome != success) of /system-property=test:read-resource
    /system-property=test:add(value=true)
end-if
```

The condition above uses **outcome**, which is returned when the CLI command after the **of** keyword is executed, as shown below:

```
/system-property=test:read-resource
{
    "outcome" => "failed",
    "failure-description" => "JBAS014807: Management resource
'[(\"system-property\" => \"test\")]' not found",
    "rolled-back" => true
}
```

The below example issues the appropriate management CLI command to enable the **ExampleDS** datasource by checking the launch type of the server process (**STANDALONE** or **DOMAIN**).

```
if (result == STANDALONE) of /:read-attribute(name=launch-type)
    /subsystem=datasources/data-source=ExampleDS:write-
attribute(name=enabled, value=true)
else
    /profile=full/subsystem=datasources/data-source=ExampleDS:write-
attribute(name=enabled, value=true)
end-if
```

Management CLI commands with **if**-**else** control flow can be specified in a file, with each command on a separate line in the file. You can then pass the file to the **jboss-cli** script to be executed non-interactively using the **--file** parameter.

```
$ EAP_HOME/bin/jboss-cli.sh --connect --file=CLI_FILE
```

3.10. USE TRY-CATCH-FINALLY CONTROL FLOW

The management CLI provides a simplified **try-catch-finally** control flow. It consists of three sets of operations and commands corresponding to the **try**, **catch**, and **finally** blocks. The **catch** and **finally** blocks are optional, but at least one of them should be present and only one catch block can be specified.

The control flow begins with execution of the **try** batch. If the **try** batch completes successfully, then the **catch** batch is skipped and the **finally** batch is executed. If the **try** batch fails, for example, **java.io.IOException**, the **try-catch-finally** control flow will terminate immediately, and the **catch** batch if it is available will be executed. The **finally** batch will always execute at the end of the control flow whether the **try** and **catch** batches succeeds or fails to execute.

There are four commands that define the try-catch-finally control flow:

- **try** command starts the **try** batch. The **try** batch continues until one of **catch** or **finally** command is encountered.
- **catch** command marks the end of the **try** batch. The **try** batch is then held back and the **catch** batch is started.

- **finally** command marks the end of the **catch** batch or the **try** batch and starts the **finally** batch.
- **end-try** is the command that ends either the **catch** or **finally** batch and runs the try-catch-finally control flow.

The following example either creates or re-creates a datasource and enables it:

```
try
/subsystem=datasources/data-source=myds:add(connection-url=xxx,jndi-
name=java:/myds,driver-name=h2)

catch
/subsystem=datasources/data-source=myds:remove
/subsystem=datasources/data-source=myds:add(connection-url=xxx,jndi-
name=java:/myds,driver-name=h2)

finally
/subsystem=datasources/data-source=myds:enable
end-try
```

3.11. QUERY A RESOURCE

The JBoss EAP management CLI provides the **query** operation to query about a resource. You can use the **:read-resource** operation to read all attributes for a resource. If you want to list only selected attributes, you can use the **:query** operation.

For example, to see the list of **name** and **enabled** attributes, use the following command:

```
/deployment=jboss-modules.jar:query(select=["name","enabled"])
```

The following response shows that the operation was successful. The **name** and **enabled** attributes are listed for the **jboss-modules.jar** deployment.

```
{
    "outcome" => "success",
    "result" => {
        "name" => "jboss-modules.jar",
        "enabled" => true
    }
}
```

You can also query across multiple resources, for example, list the **name** and **enabled** attributes of all deployments, using a wildcard:

```
/deployment=*:query(select=["name","enabled"])
```

The following response shows that the operation was successful. The **name** and **enabled** attributes of all deployments are listed.

```
{
     "outcome" => "success",
     "result" => [
         {
```

```
            "address" => [("deployment" => "jboss-helloworld.war")],
            "outcome" => "success",
            "result" => {
                "name" => "jboss-helloworld.war",
                "enabled" => true
            }
        },
        {
            "address" => [("deployment" => "jboss-kitchensink.war")],
            "outcome" => "success",
            "result" => {
                "name" => "jboss-kitchensink.war",
                "enabled" => true
            }
        },
        {
            "address" => [("deployment" => "xyz.jar")],
            "outcome" => "success",
            "result" => {
                "name" => "xyz.jar",
                "enabled" => false
            }
        }
    ]
}
```

The **`:query`** operation also filters relevant objects. For example, to view the **`name`** and **`enabled`** attribute values for deployments with **`enabled`** as **`true`**.

```
/deployment=*:query(select=["name","enabled"],where=["enabled","true"])
```

The following response shows that the operation was successful. The **`name`** and **`enabled`** attribute values for deployments with **`enabled`** as **`true`** are listed.

```
{
    "outcome" => "success",
    "result" => [
        {
            "address" => [("deployment" => "jboss-helloworld.war")],
            "outcome" => "success",
            "result" => {
                "name" => "jboss-helloworld.war",
                "enabled" => true
            }
        },
        {
            "address" => [("deployment" => "jboss-kitchensink.war")],
            "outcome" => "success",
            "result" => {
                "name" => "jboss-kitchensink.war",
                "enabled" => true
            }
        }
    ]
}
```

CHAPTER 4. CONFIGURING THE MANAGEMENT CLI

Certain aspects of the management CLI can be customized in its configuration file, **jboss-cli.xml**. This file must be located either in the **EAP_HOME/bin** directory or in a custom directory specified with the **jboss.cli.config** system property.

The following elements can be configured in the **jboss-cli.xml** file.

default-protocol

The default protocol to use when controller addresses are supplied without one. The default is **http-remoting**. If port 9999 is used and no protocol is specified, the protocol will automatically default to **remoting** unless you set the **use-legacy-override** attribute to **false**.

default-controller

Configuration of the controller to which to connect if the **connect** command is executed without any parameters. If the management CLI is started with the argument **--controller=** or **controller=**, then the value specified in the argument overrides the **default-controller** definition from the configuration.

- **protocol** - Protocol name of the controller. If one is not provided, the value of **default-protocol** will be used.
- **host** - Host name of the controller. The default is **localhost**.
- **port** - Port number on which to connect to the controller. The default value is **9990**.

controllers

You can define connection controller aliases in the **jboss-cli.xml** file. For example:

```
<!-- The default controller to connect to when 'connect' command
is executed w/o arguments -->
<default-controller>
    <host>localhost</host>
    <port>9990</port>
</default-controller>
<!-- CLI connection controller aliases -->
<controllers>
    <controller name="ServerOne">
        <protocol>remoting</protocol>
        <host>192.168.3.45</host>
        <port>9999</port>
    </controller>
    <controller name="ServerTwo">
        <protocol>http-remoting</protocol>
        <host>192.168.3.46</host>
    </controller>
</controllers>
```

The **name** attribute of controller element should be used as a value to **--controller=** argument. For example, **--controller=ServerTwo**.

`validate-operation-requests`

Whether to validate the parameter list of operation requests before sending the requests to the controller for execution. The default is **`true`**.

`history`

The configuration for the CLI command history log.

- **`enabled`** - Whether or not the **`history`** is enabled. The default is **`true`**.
- **`file-name`** - The file name in which the history will be stored. The default is **`.jboss-cli-history`**.
- **`file-dir`** - The directory in which the history is to be stored. The default is the user's home directory.
- **`max-size`** - The maximum size of the history file. The default is **`500`**.

`resolve-parameter-values`

Whether to resolve system properties specified as command argument (or operation parameter) values before sending the operation request to the controller. The default is **`false`**.

`connection-timeout`

The time in milliseconds allowed to establish a connection with the controller. The default is **`5000`**.

`ssl`

The configuration for the key and trust stores used for SSL.

> **Warning**
>
> Red Hat recommends that SSLv2, SSLv3, and TLSv1.0 be explicitly disabled in favor of TLSv1.1 or TLSv1.2 in all affected packages.

- **`vault`** - The vault configuration. If neither **`code`** nor **`module`** is specified, the default implementation will be used. If **`code`** is specified but not **`module`**, it will look for the specified class in the Picketbox module. If **`module`** and **`code`** are specified, it will look for the class specified by code in the module specified by 'module'.
- **`key-store`** - The key store.
- **`key-store-password`** - The key store password.
- **`alias`** - The alias.
- **`key-password`** - The key password.
- **`trust-store`** - The trust store.
- **`trust-store-password`** - The trust store password.

- **`modify-trust-store`** - If set to **`true`**, the CLI will prompt the user when unrecognized certificates are received and allow them to be stored in the truststore. The default is **`true`**.

`silent`

Whether to write informational and error messages to the terminal. The default is **`false`**.

`access-control`

Whether the management-related commands and attributes should be filtered for the current user based on the permissions the user has been granted. For example, if **`true`**, tab completion will hide commands and attributes that the user is not allowed to access. The default is **`true`**.

4.1. PROPERTY SUBSTITUTION

JBoss EAP supports the use of preset element and property expressions in the management CLI. These expressions will be resolved to their defined values during the execution of the command.

You can substitute expressions for the following properties:

- the operation address part of the operation request (for example, node types or names)
- operation name
- operation parameter names
- header names and values
- command names
- command argument names

By default, the management CLI performs property substitution for every line except for argument or parameter values. Argument and parameter values are resolved in the server at runtime. If you require property substitution for argument or parameter values to occur in the management CLI and have it send the resolved values to the server, complete the following steps.

1. Edit the management CLI configuration file: **`EAP_HOME/bin/jboss-cli.xml`**.
2. Set the **`resolve-parameter-values`** parameter to **`true`** (the default is **`false`**).

```
<resolve-parameter-values>true</resolve-parameter-values>
```

This element only affects operation request parameter values and command argument values. It does not impact the rest of the command line. This means system properties present on the command line will be resolved during the parsing of the line regardless of what the value of **`resolve-parameter-values`** element is, unless it is a parameter/argument value.

System property values used in management CLI commands must have already been defined in order to be resolved. You must either pass in a properties file (**`--properties=/path/to/file.properties`**) or property value pairs (**`-Dkey=value`**) when starting your management CLI instance. The properties file uses a standard **`KEY=VALUE`** syntax.

Property keys are denoted in your management CLI commands using the **`${MY_VAR}`** syntax, for example:

```
/host=${hostname}/server-config=${servername}:add(group=main-server-
group)
```

See Configuring the Management CLI for other **`jboss-cli.xml`** configuration options.

4.2. CREATING ALIASES

You can define aliases for the CLI commands and operations during a CLI session using the **`alias`** command.

The following example creates a new CLI command alias named **`read_undertow`** to read the resources in the undertow subsystem using the **`alias`** command:

```
alias read_undertow='/subsystem=undertow:read-resource'
```

To test the creation of **`read_undertow`** alias, type the alias name in the management CLI:

```
read_undertow
```

The result will be:

```
{
    "outcome" => "success",
    "result" => {
        "default-security-domain" => "other",
        "default-server" => "default-server",
        "default-servlet-container" => "default",
        "default-virtual-host" => "default-host",
        "instance-id" => expression "${jboss.node.name}",
        "statistics-enabled" => false,
        "buffer-cache" => {"default" => undefined},
        "configuration" => {
            "filter" => undefined,
            "handler" => undefined
        },
        "server" => {"default-server" => undefined},
        "servlet-container" => {"default" => undefined}
    }
}
```

To view a list of all available aliases, use the **`alias`** command:

```
alias
```

The result will be:

```
alias read_undertow='/subsystem=undertow:read-resource'
```

To remove an alias, use the **`unalias`** command:

```
unalias read_undertow
```

Note

The aliases are stored in the **.aesh_aliases** file within the user's home folder.

4.3. .JBOSSCLIRC CONFIGURATION FILE

JBoss EAP contains the runtime configuration **.jbossclirc** file, which helps you to initialize the environment when a new session is launched. This file is located in **EAP_HOME/bin/** directory. The example provided in the file can be used as a template for user-specific environment setup. The **.jbossclirc** file is ideal for storing global CLI variables.

The content of the **.jbossclirc** file is a list of CLI supported commands and operations. This file is executed when a new management CLI session is launched but before the control is given to the user. If there are system properties specified with **--properties** argument, then the **.jbossclirc** file is executed after the properties have been set.

Example .jbossclirc File

```
set console=/subsystem=logging/console-handler=CONSOLE
```

Note

When using the **--connect** or **-c** argument, **.jbossclirc** is executed before the client is actually connected to the server.

The following locations will be checked for the presence of the **.jbossclirc** file in the following order:

1. If the system property **jboss.cli.rc** is defined, its value will be considered a path to the file.
2. User's working directory as defined by the **user.dir** system property.
3. The **EAP_HOME/bin** directory.

4.4. USING VARIABLES

Using the Set Command

You can define a certain path of the server model to a variable using the **set** command. For example:

```
set s1=/host=master/server=server-one
```

This is useful in a managed domain as you can include references to host and profiles using variables to easily replicate scripts on different servers. For example:

```
$s1/subsystem=datasources/data-source=ExampleDS:test-connection-in-pool
```

Note

The variables are referenced using **$**.

Using the Unset Command

You can remove the variable by using the **unset** command:

```
unset prod_db
```

Using the jbossclirc File

To use the variable across CLI session, you can include these variables in the **.jbossclirc** file. This file is located in the **EAP_HOME/bin/** directory.

For example:

```
set s1=/host=master/server=server-one
set s2=/host=master/server=server-two
```

Now, restart the management CLI and issue a **set** command to check the available variables:

```
set
```

The output will be:

```
s1=/host=master/server=server-one
s2=/host=master/server=server-two
```

The variables may appear in any part of a command line and resolved during the command line parsing phase. In this example, the **prod_db** variable will be resolved to a datasource:

```
$prod_db/statistics=jdbc:read-resource
```

Using the Echo Command

Use the **echo** command to check the value of a variable:

```
echo $prod_db
```

The output will be:

```
/subsystem=datasources/data-source=ExampleDS
```

Example

The following general examples show where the variables may appear and that the entire command line may consist of variables:

```
$prod_db:$op($param=$param_value)
$cmd --$param=$param_value
```

Note

The variables help you in CLI scripting.

CHAPTER 5. MANAGEMENT CLI COMMAND HISTORY

The management CLI features a command history functionality that is enabled by default in the application server installation. The history is kept both as a record in the volatile memory of the active CLI session, and appended to a log file that saves automatically in the user's home directory as **`.jboss-cli-history`**. This history file is configured by default to record up to a maximum of **`500`** CLI commands. The history file location and maximum history entries can be customized in the **`EAP_HOME/bin/jboss-cli.xml`** file.

The **`history`** command by itself will return the history of the current session, or with additional arguments will disable, enable, or clear the history from the session memory. The management CLI also features the ability to use your keyboard's arrow keys to go back and forth in the history of commands and operations.

View the Management CLI Command History

Display the CLI command history stored in memory since the management CLI startup or the history clear command.

```
history
```

Clear the Management CLI Command History

Clear the history of CLI commands from the session memory and from the **`.jboss-cli-history`** file saved to the user's home directory.

```
history --clear
```

Enable the Management CLI Command History

Record CLI commands in the session memory and in the **`.jboss-cli-history`** file saved to the user's home directory.

```
history --enable
```

Disable the Management CLI Command History

Do not record CLI commands in the session memory or in the **`.jboss-cli-history`** file saved to the user's home directory.

```
history --disable
```

CHAPTER 6. BATCH PROCESSING

Batch processing allows multiple operation requests to be grouped in a sequence and executed together as a unit. If any of the operation requests in the sequence fail, the entire group of operations is rolled back.

Note

Batch mode does not support conditional statements.

1. Enter batch mode with the **`batch`** management CLI command.

   ```
   batch
   ```

   Batch mode is indicated by the hash symbol (#) in the prompt.

2. Add operation requests to the batch.

   Once in batch mode, enter operation requests as normal. The operation requests are added to the batch in the order they are entered.

   You can edit and reorder batch commands. You can also store a batch for processing at a later time. See Batch Mode Commands for a full list of commands available for working with batches.

3. Run the batch.

   Once the entire sequence of operation requests is entered, run the batch with the **`run-batch`** command.

   ```
   run-batch
   ```

   The entered sequence of operation requests is completed as a batch and prints the result to the terminal: **`The batch executed successfully.`**

Batch Commands in External Files

Frequently-run batch commands can be stored in an external text file and can be loaded either by passing the full path to the file as an argument to the **`batch`** command or executed directly by being passed as an argument to the **`run-batch`** command.

You can create a batch command file by using a text editor and placing each command on its own line.

The following command will load the **`myscript.txt`** file in batch mode. The commands from this file can then be edited or removed. New commands can be inserted. Changes made in this batch session do not persist to the **`myscript.txt`** file.

```
batch --file=myscript.txt
```

The following will immediately run the batch commands stored in the file **`myscript.txt`**

```
run-batch --file=myscript.txt
```

-

The entered sequence of operation requests is completed as a batch.

CHAPTER 7. EMBEDDING A SERVER FOR OFFLINE CONFIGURATION

You can embed a JBoss EAP standalone server or host controller process inside the management CLI process. This allows you to configure the server without it being visible on the network. A common use of this feature is for initial configuration of the server, such as managing security-related settings or avoiding port conflicts, prior to the server being online.

This direct, local administration of a JBoss EAP installation through the management CLI does not require a socket-based connection. You can use the management CLI with the embedded server in a way that is consistent with interacting with a remote JBoss EAP server. All of the standard management CLI commands that you can use to administer a remote server are available.

Start an Embedded Standalone Server

You can launch a standalone server locally using the management CLI to modify standalone configuration without launching an additional process or opening network sockets.

The following procedure launches the management CLI, starts an embedded standalone server, modifies configuration, and then stops the embedded server.

1. Launch the management CLI.

```
$ EAP_HOME/bin/jboss-cli.sh
```

2. Launch the embedded standalone server.

   Passing in the **`--std-out=echo`** parameter prints the standard output to the terminal.

```
embed-server --std-out=echo
```

3. Perform the desired operations.

```
/socket-binding-group=standard-sockets/socket-binding=management-
http:write-attribute(name=port,value=9991)
```

4. Stop the embedded server.

```
stop-embedded-server
```

   This stops the embedded server and returns you to your management CLI session. If you want to exit the management CLI session as well, you can use the **`quit`** command.

Specifying the Server Configuration

By default, the embedded server will use the **`standalone.xml`** configuration file. You can use the **`--server-config`** parameter to specify a different configuration file to use.

```
embed-server --server-config=standalone-full-ha.xml
```

Starting in Admin-Only Mode

By default, the embedded server is started in **admin-only** mode, which will start services related to server administration, but will not start other services or accept end-user requests. This is useful for the initial configuration of the server.

You can start the embedded server in the normal running mode by setting the **--admin-only** parameter to false.

```
embed-server --admin-only=false
```

You can also use the reload command to toggle the **admin-only** setting for the server.

```
reload --admin-only=false
```

Controlling Standard Output

You can control how to handle standard output from the embedded server. By default, standard output is discarded, but you could find the output in the server log. You can pass in **--std-out=echo** to have server output appear with the management CLI output.

```
embed-server --std-out=echo
```

Boot Timeout

By default, the **embed-server** command blocks indefinitely waiting for the embedded server to fully start. You can specify the time to wait in seconds using the **--timeout** parameter. A value less than **1** will return as soon as the embedded server reaches a point where it can be managed by the CLI.

```
embed-server --timeout=30
```

Starting with a Blank Configuration

When starting an embedded server, you can specify to start with an empty configuration. This is useful if you want to build the entire server configuration using management CLI commands.

```
embed-server --server-config=my-config.xml --empty-config
```

This command will fail if the file already exists, which helps to avoid the accidental deletion of a configuration file. You can specify to remove any existing configuration by passing in the **--remove-existing** parameter.

```
embed-server --server-config=my-config.xml --empty-config --remove-
existing
```

Start an Embedded Host Controller

You can launch a host controller locally using the management CLI to modify domain and host controller configuration without launching additional processes or opening network sockets.

An embedded host controller does not start any of its servers. Additionally, you can not use the **--admin-only** parameter when starting an embedded host controller. It will always be launched as if it is in **admin-only** mode.

The following procedure launches the management CLI, starts an embedded host controller, modifies configuration, and then stops the embedded host controller.

1. Launch the management CLI.

```
$ EAP_HOME/bin/jboss-cli.sh
```

2. Launch the embedded host controller.

   Passing in the **`--std-out=echo`** parameter prints the standard output to the terminal.

```
embed-host-controller --std-out=echo
```

3. Perform the desired operations.

```
/host=HOST_NAME:write-attribute(name=name,value=NEW_HOST_NAME)
```

4. Stop the embedded host controller.

```
stop-embedded-host-controller
```

Specifying the Host Controller Configuration

By default, the embedded host controller will use **`domain.xml`** for domain configuration and **`host.xml`** for host configuration. You can use the **`--domain-config`** and **`--host-config`** parameters to specify different configuration files to use.

```
embed-host-controller --domain-config=other-domain.xml --host-
config=host-slave.xml
```

Note

Depending on which alternative configuration file you use, you may need to set certain properties when launching the management CLI. For example,

```
$ EAP_HOME/bin/jboss-cli.sh -
Djboss.domain.master.address=127.0.0.1
```

Controlling Standard Output

You can control how to handle standard output from the embedded host controller. By default, standard output is discarded, but you could find the output in the host controller's log. You can pass in **`--std-out=echo`** to have host controller output appear with the management CLI output.

```
embed-host-controller --std-out=echo
```

Boot Timeout

By default, the **embed-host-controller** command blocks indefinitely waiting for the embedded host controller to fully start. You can specify the time to wait in seconds using the **--timeout** parameter. A value less than **1** will return as soon as the embedded host controller reaches a point where it can be managed by the CLI.

```
embed-host-controller --timeout=30
```

Non-Modular Class Loading with the Management CLI

Using the **EAP_HOME/bin/jboss-cli.sh** script to launch the management CLI uses a modular class loading environment. If you use the **EAP_HOME/bin/client/jboss-cli-client.jar** to run the management CLI in a non-modular class loading environment, you will need to specify the root JBoss EAP installation directory.

1. Launch the management CLI.

   ```
   $ java -jar EAP_HOME/bin/client/jboss-cli-client.jar
   ```

2. Start the embedded server, specifying the root installation directory.

   ```
   embed-server --jboss-home=/path/to/EAP_HOME
   ```

   

   Note

   To embed a host controller, use the **embed-host-controller** command.

   The embedding logic will set up an appropriate modular class loading environment for the server. The module path for the modular class loader will have a single element: **EAP_HOME/modules**.

No matter which way you launch the management CLI, the embedded server will run in a modular class loading environment.

CHAPTER 8. HOW TO…

The following CLI commands and operations provide basic examples on how to accomplish certain tasks. For detailed instructions, see the appropriate section of the *Configuration Guide* or other JBoss EAP guide.

Unless specified otherwise, the examples apply when running as a standalone server. Use the **`--help`** argument on a command to get usage for that command. Use the **`read-operation-description`** to get information on a particular operation for a resource.

8.1. ADD A DATASOURCE

```
data-source add --name=DATASOURCE_NAME --jndi-name=JNDI_NAME --driver-
name=DRIVER_NAME  --connection-url=CONNECTION_URL
```

8.2. ADD A JMS QUEUE

```
jms-queue add --queue-address=QUEUE --entries=java:/jms/queue/QUEUE
```

8.3. ADD A JMS TOPIC

```
jms-topic add --topic-address=TOPIC --entries=java:/jms/topic/TOPIC
```

8.4. ADD A MODULE

```
module add --name=MODULE_NAME --resources=PATH_TO_RESOURCE --
dependencies=DEPENDENCIES
```

8.5. ADD A SERVER

Add a new server to a host in a managed domain.

```
/host=HOST_NAME/server-config=SERVER_NAME:add(group=SERVER_GROUP_NAME)
```

8.6. ADD A SERVER GROUP

Add a new server group in a managed domain.

```
/server-group=SERVER_GROUP_NAME:add(profile=PROFILE_NAME, socket-
binding-group=SOCKET_BINDING_GROUP_NAME)
```

8.7. ADD A SYSTEM PROPERTY

```
/system-property=PROPERTY_NAME:add(value=PROPERTY_VALUE)
```

8.8. CLONE A PROFILE

Clone a profile in a managed domain.

```
/profile=PROFILE_TO_CLONE:clone(to-profile=NEW_PROFILE_NAME)
```

8.9. CREATE A HIERARCHICAL PROFILE

Create a new profile that inherits from other profiles.

```
/profile=NEW_PROFILE_NAME:add(includes=[PROFILE_1,PROFILE_2])
```

8.10. DEPLOY AN APPLICATION TO A MANAGED DOMAIN

Deploy an Application to All Server Groups.

```
deploy /path/to/DEPLOYMENT.war --all-server-groups
```

Deploy an Application to One or More Server Groups.

```
deploy /path/to/DEPLOYMENT.war --server-
groups=SERVER_GROUP_1,SERVER_GROUP_2
```

8.11. DEPLOY AN APPLICATION TO A STANDALONE SERVER

```
deploy /path/to/DEPLOYMENT.war
```

8.12. DISPLAY THE ACTIVE USER

Command:

```
:whoami
```

Output:

```
{
    "outcome" => "success",
    "result" => {"identity" => {
        "username" => "$local",
        "realm" => "ManagementRealm"
    }}
}
```

8.13. DISPLAY SCHEMA INFORMATION

To show the schema information for the **`:product-info`** command:

```
:read-operation-description(name=product-info)
```

To display the schema version, execute an **`ls`** command at the management CLI root and look for the **`management-*-version`** values:

```
...
management-major-version=4
management-micro-version=0
management-minor-version=1
...
```

8.14. DISPLAY SYSTEM AND SERVER INFORMATION

Command:

```
:product-info
```

Output:

```
{
    "outcome" => "success",
    "result" => [{"summary" => {
        "host-name" => "HOST_NAME",
        "instance-identifier" => "INSTANCE_ID",
        "product-name" => "JBoss EAP",
        "product-version" => "7.0.0.GA",
        "product-community-identifier" => "Product",
        "product-home" => "EAP_HOME",
        "standalone-or-domain-identifier" => "OPERATING_MODE",
        "host-operating-system" => "OS_NAME",
        "host-cpu" => {
            "host-cpu-arch" => "CPU_ARCH",
            "host-core-count" => CORE_COUNT
        },
        "jvm" => {
            "name" => "JAVA_VM_NAME",
            "java-version" => "JAVA_VERSION",
            "jvm-version" => "JAVA_VM_VERSION",
            "jvm-vendor" => "JAVA_VM_VENDOR",
            "java-home" => "JAVA_HOME"
        }
    }}]
}
```

Similarly, for a managed domain, you can display the information for a particular JBoss EAP host or server:

```
/host=HOST_NAME:product-info
```

```
/host=HOST_NAME/server=SERVER_NAME:product-info
```

8.15. RELOAD THE SERVER

```
reload
```

8.16. RELOAD THE SERVER IN ADMIN-ONLY MODE

```
reload --admin-only=true
```

8.17. SHUT DOWN A HOST CONTROLLER

Shut down a host controller in a managed domain.

```
shutdown --host=HOST_NAME
```

8.18. SHUT DOWN THE SERVER

Shut down a standalone server.

```
shutdown
```

8.19. START A SERVER

Start a server in a managed domain.

```
/host=HOST_NAME/server-config=SERVER_NAME:start
```

8.20. START ALL SERVERS IN A SERVER GROUP

Start all servers in a certain server group in a managed domain.

```
/server-group=SERVER_GROUP_NAME:start-servers
```

8.21. STOP A SERVER

Stop a server in a managed domain.

```
/host=HOST_NAME/server-config=SERVER_NAME:stop
```

8.22. STOP ALL SERVERS IN A SERVER GROUP

Stop all servers in a certain server group in a managed domain.

```
/server-group=SERVER_GROUP_NAME:stop-servers
```

8.23. TAKE A CONFIGURATION SNAPSHOT

Take a snapshot of the current configurations.

```
:take-snapshot
```

8.24. UNDEPLOY AN APPLICATION FROM A MANAGED DOMAIN

Undeploy an application from all server groups with that deployment.

```
undeploy DEPLOYMENT.war --all-relevant-server-groups
```

Undeploy an application from a specific server group. The **`--keep-content`** parameter is required in order to keep the content in the repository for other server groups with that deployment.

```
undeploy DEPLOYMENT.war --server-groups=SERVER_GROUP_NAME --keep-
content
```

8.25. UNDEPLOY AN APPLICATION FROM A STANDALONE SERVER

```
undeploy DEPLOYMENT.war
```

8.26. UPDATE A HOST NAME

Update the name of a host in a managed domain. The host must be reloaded in order for the changes to take effect.

```
/host=EXISTING_HOST_NAME:write-attribute(name=name,value=NEW_HOST_NAME)
reload --host=EXISTING_HOST_NAME
```

8.27. VIEW A SERVER LOG

```
/subsystem=logging/log-file=SERVER_LOG_NAME:read-log-file
```

APPENDIX A. REFERENCE MATERIAL

A.1. MANAGEMENT CLI STARTUP ARGUMENTS

The following table lists the arguments that can be passed into the **`jboss-cli`** script to launch the management CLI.

Table A.1. Management CLI Arguments

Argument	Description
`--bind`	Specifies to which address the CLI is going to be bound to. If none is provided then the CLI will choose one automatically as needed.
`--command`	Specifies a single command or an operation that should be executed in the CLI session. The CLI will terminate the session immediately after the command or the operation has been executed.
`--commands`	Specifies a comma-separated list (must not contain whitespace) of commands and operations that should be executed in the CLI session. The CLI session will terminate after the last command has been executed or if a command fails.
`--connect`, **`-c`**	Instructs the CLI to connect to the controller on start-up. This avoid having to issue a separate **`connect`** command later.
`--controller`	The default controller host and port to connect to when the **`--connect`** option is specified or when the **`connect`** command is issued without arguments. The default controller host is **`localhost`** and the default port is **`9990`**.
`--error-on-interact`	Disables prompts for security-related input in non-interactive mode. If an input is required to proceed, the CLI process will terminate abruptly with an error.
`--file`	Specifies the path to a file which contains commands and operations (one per line) that should be executed non-interactively. The CLI will terminate after the last command has been executed or if a command or operation fails.

Argument	Description
--gui	Launches a GUI that is built on top of the command-line interface. In this mode, the CLI will automatically connect during start-up.
--help, **-h**	Displays the help message.
--no-local-auth	Disables the local authentication mechanism which allows the CLI to demonstrate that it is being executed locally to the server being managed through an exchange of tokens using the file system.
--password, **-p**	Specifies the password for authentication while connecting to the controller. If the argument is not specified and authentication is required, then the user will be prompted to enter the password when the **connect** command is issued.
--timeout	Specifies the time in milliseconds to wait for a given command to return. The default is **5000**.
--user, **-u**	Specifies the user name if the controller requires user authentication. If the argument is not specified and authentication is required, then the user will be prompted to enter the user name when the **connect** command is issued. Local authentication is automatically disabled if a user is specified.
--version	Displays the application server version and environment information.

A.2. MANAGEMENT CLI BATCH MODE COMMANDS

This table provides a list of commands that can be used with the management CLI to work with batches.

Table A.2. Management CLI Batch Mode Commands

Command Name	Description
clear-batch	Removes all the existing command lines from the currently active batch.

Command Name	Description
`discard-batch`	Discards the currently active batch and exits batch mode.
`edit-batch-line`	Edit a line in the current batch by providing the line number to edit and the edited command. For example: **`edit-batch-line 2 data-source disable --name=ExampleDS`**.
`holdback-batch`	Postpone or store a current batch. Using this command without arguments creates an unnamed heldback batch. To return to this heldback batch, simply type **`batch`** again at the CLI command line. There can be only one unnamed heldback batch. You can optionally provide a name under which to store the batch by using the **`holdback_name`** argument. To return to the named batch, pass the **`heldback_name`** to the **`batch`** command. Use the **`batch -l`** command to see a list of all heldback batches.
`list-batch`	Lists all commands in the currently active batch.
`move-batch-line`	Re-order the lines in the batch by specifying the line number you want to move as the first argument and its new position as the second argument. For example: ` move-batch-line 3 1`.
`remove-batch-line`	Removes the batch command at the specified line. For example: **`remove-batch-line 3`**.
`run-batch`	Runs the currently active batch. If the batch executed successfully, the batch will be discarded and the CLI will exist batch mode.

A.3. MANAGEMENT CLI COMMANDS

The following table lists management CLI commands and their purposes. For more usage and argument details, use the **`--help`** argument on a specific command.

Table A.3. Management CLI Commands

Command	Description
alias	Define an alias with the format **NAME=VALUE**. When no arguments are specified, the list of aliases are displayed.
batch	Starts batch mode by creating a new batch. If there is an unnamed held back batch, it will be reactivated. If there are named held back batches, reactivate by specifying the **heldback_name**.
cd	Changes to the specified path.
clear	Clears the screen.
command	Allows you to add, remove, and list existing generic type commands. A generic type command is a command that is assigned to a specific node type and which allows you to perform any operation available for an instance of that type. It can also modify any of the properties exposed by the type on any existing instance.
connect	Connects to the controller on the specified host and port using the specified protocol when the management CLI was launched. If not specified, the default host is **localhost**, the default port is **9990**, and the default protocol is **http**.
connection-factory	Manages connection factories in the messaging-activemq subsystem.
connection-info	Displays information about the current connection to the server.
data-source	Manages datasource configuration in the datasource subsystem.
deploy	Deploys an application designated by **file_path** or enables a pre-existing application in the repository. If no arguments are specified, all existing deployments are listed.
deployment-info	Displays information about an individual deployment or about multiple deployments.

Command	Description
`deployment-overlay`	Manage deployment overlays. If no arguments are specified, all existing deployment overlays are listed.
`echo`	Outputs the specified text to the console.
`echo-dmr`	Builds a DMR request for the command or operation passed in as the argument and echos in its **`toString()`** format.
`help`	Displays the help message. Can be used with the **`--commands`** argument to provide a list of available commands.
`history`	Displays the CLI command history in memory and displays a status of whether the history expansion is enabled or disabled. Can be used with arguments to clear, disable and enable the history as required.
`if`	Starts if-else control flow.
`jdbc-driver-info`	Displays information about the installed JDBC drivers.
`jms-queue`	Manage JMS queues in the messaging-activemq subsystem.
`jms-topic`	Manage JMS topics in the messaging-activemq subsystem.
`ls`	Lists the contents of the node path. Use the **`-l`** switch to print the result one per line.
`module`	Add and remove modules.
`patch`	Apply or roll back a patch to the server.
`pwd`	Prints the full node path of the current working node.

Command	Description
`quit`	Terminates the command line interface.
`read-attribute`	Prints the value and, depending on the arguments, the description of the attribute of a managed resource.
`read-operation`	Displays the description of a specified operation, or lists all available operations if none is specified.
`reload`	Sends the **`:reload`** operation request to the server/domain controller and waits for the controller to close the connection and then it returns the control back to the client.
`rollout-plan`	Manage stored rollout plans.
`run-batch`	Runs the currently active batch while in batch mode. While not in batch mode, can use with the **`--file`** argument to execute the contents of the file as a batch.
`set`	Initializes variables with the given names with the specified values.
`shutdown`	Sends the **`:shutdown`** operation request to the server/domain controller and waits for the controller to close the connection.
`try`	Starts a try-catch-finally control flow.
`unalias`	Remove the specified alias.
`undeploy`	Undeploys an application with the specified name and, depending on the arguments, may remove its contents from the content repository. If no arguments are specified, all existing deployments are listed.
`unset`	Removes an existing variable with the specified name.
`version`	Prints the application server version and environment information.

Command	Description
xa-data-source	Manages XA datasource configuration in the datasource subsystem.

A.4. MANAGEMENT CLI OPERATIONS

The following table lists management CLI operations that are available at the root level (`/`). The actual available operations for a particular resource will vary per resource and also depend on the operating mode (standalone server or managed domain).

Operations are invoked using a colon (`:`). The available operations for a resource can be exposed by using the **read-operation-names** operation or by using tab completion after a colon. Operation descriptions can be displayed by using the **read-operation-description** operation. For example:

```
:read-operation-description(name=write-attribute)
```

Table A.4. Management CLI Operations

Operation Name	Description
add-namespace	Adds a namespace prefix mapping to the **namespaces** attribute's map.
add-schema-location	Adds a schema location mapping to the **schema-locations** attribute's map.
clean-obsolete-content	Clean contents that are no longer referenced from the content repository.
delete-snapshot	Deletes a snapshot of the server configuration from the snapshots directory.
full-replace-deployment	Adds previously uploaded deployment content to the list of content available for use, replace existing content of the same name in the runtime, and remove the replaced content from the list of content available for use.

Operation Name	Description
`list-add`	Add an entry to a list attribute.
`list-clear`	Clear all entries from a list attribute.
`list-get`	Get an entry from a list attribute.
`list-remove`	Remove an entry from a list attribute.
`list-snapshots`	Lists the snapshots of server configurations saved in the snapshots directory.
`map-clear`	Clear all entries from a map attribute.
`map-get`	Get an entry from a map attribute.
`map-put`	Add an entry to a map attribute.
`map-remove`	Remove an entry from a map attribute.
`product-info`	Returns a summary of the current server installation.
`query`	Query a resource.
`read-attribute`	Displays the value of an attribute for the selected resource.
`read-attribute-group`	Displays the value of attributes for the selected group.
`read-attribute-group-names`	Displays the names of all the attribute groups under the selected resource.

Operation Name	Description
`read-children-names`	Displays the names of all children under the selected resource with the given type.
`read-children-resources`	Displays information about all of a resource's children that are of a given type.
`read-children-types`	Displays the type names of all the children under the selected resource.
`read-config-as-xml`	Displays the current configuration in XML format.
`read-operation-description`	Displays the details of an operation for the given resource.
`read-operation-names`	Displays the names of all available operations for the given resource.
`read-resource`	Displays a resource's attribute values along with either basic or complete information about any child resources.
`read-resource-description`	Displays the description of a resource's attributes, types of children and operations.
`reload`	Reloads the server by shutting all services down and restarting.
`reload-servers`	Reloads all servers currently running in the domain.
`remove-namespace`	Removes a namespace prefix mapping from the **`namespaces`** attribute map.
`remove-schema-location`	Removes a schema location mapping from the **`schema-locations`** attribute map.

Operation Name	Description
`replace-deployment`	Replace existing content in the runtime with new content. The new content must have been previously uploaded to the deployment content repository.
`resolve-expression`	Accepts an expression as input (or a string that can be parsed into an expression), and resolves it against the local system properties and environment variables.
`resolve-expression-on-domain`	Accepts an expression as input (or a string that can be parsed into an expression) and resolves it against the local system properties and environment variables on all servers in the domain.
`resolve-internet-address`	Takes a set of interface resolution criteria and finds an IP address on the local machine that matches the criteria, or fails if no matching IP address can be found.
`restart-servers`	Restarts all servers currently running in the domain.
`resume`	Resumes normal operations in a suspended server.
`resume-servers`	Resumes processing on all servers in the domain.
`shutdown`	Shuts down the server with a call to **`System.exit(0)`**.
`start-servers`	Starts all configured servers in the managed domain that are not currently running.
`stop-servers`	Stops all servers currently running in the managed domain.
`suspend`	Suspends server operations gracefully. All current requests will complete normally, however no new requests will be accepted.
`suspend-servers`	Suspends all servers in the domain. All current operations will finish and no new operations will be allowed.

Operation Name	Description
`take-snapshot`	Takes a snapshot of the server configuration and saves it to the snapshots directory.
`undefine-attribute`	Sets the value of an attribute of the selected resource to **`undefined`**.
`upload-deployment-bytes`	Indicates that the deployment content in the included byte array should be added to the deployment content repository. Note that this operation does not indicate the content should be deployed into the runtime.
`upload-deployment-stream`	Indicates that the deployment content available at the included input stream index should be added to the deployment content repository. Note that this operation does not indicate the content should be deployed into the runtime.
`upload-deployment-url`	Indicates that the deployment content available at the included URL should be added to the deployment content repository. Note that this operation does not indicate the content should be deployed into the runtime.
`validate-address`	Checks whether a resource with the specified address exists.
`validate-operation`	Validates that an operation is valid according to its description. Any errors present will be shown in the operation's **`failure-description`**.
`whoami`	Returns the identity of the currently authenticated user.
`write-attribute`	Sets the value of an attribute for the selected resource.

Revised on 2016-09-19 12:21:59 EDT