EPAM Cloud Orchestrator

Maestro CLI

Developer’s Guide

April 2021

CSDG-1

Version 4.2

CONTENT

PREFACE

ABOUT THIS GUIDE

Maestro Command Line Interface (CLI) is intended to perform basic Orchestrator commands via remote command line by sending server API requests using Public API without the need to install 3rd party utilities.

This document provides detailed guidelines on use and customization of Maestro CLI client.

AUDIENCE

This guide is designed for EPAM Cloud users and projects which want to use Maestro API in their applications, as well as for EPAM internal ecosystem elements such as UPSA.

THE STRUCTURE OF THE GUIDE

The guide consists of the following chapters:

1. Public API for CLI commands. This chapter gives general information about Public API for CLI commands as well as examples of CLI and API requests with responses.
2. Setup Extras. This chapter gives the most essential information about Maestro CLI installation and customization.
3. CLI Client Authorization. This chapter gives the information about the authorization process and the storage of the authorization data. Authorization token calculation algorithm is also given here.
4. HTTP Client Fine-Tune. This chapter includes information about the HTTP client, its settings and parameters.
5. Log Subsystem. This chapter contains the information about the Apache Log4j 1.2 library which is used as a log subsystem. The details on logging performance and output are given here.
6. Maestro CLI Commands. This chapter gives the details on the CLI commands exit codes.
7. Maestro CLI Output Customization. This chapter describes the ways to customize Maestro CLI output according to the FreeMarker template engine specifics.

DOCUMENTATION REFERENCES

EPAM Orchestration is described in details in a number of documents, oriented on different aspects of Orchestration usage, and on different types of users.

You can find these documents on our Documentation page.

The answers to the most frequently asked questions can be found on the FAQ page.

EPAM Cloud terms and conditions are described in the EPAM Cloud Terms and Conditions. Please take a look at this document in order to avoid misunderstandings and conflicts that may arise during the service usage.

The terminology of EPAM Cloud and the related products can be found on the Glossary page.

Please email your comments and feedback to EPAM Cloud Consulting at

SpecialEPM-CSUPConsulting@epam.com to help us provide you with documentation that is as clear, correct and readable as possible.

1. PUBLIC API FOR CLI COMMANDS

1.1 GENERAL INFORMATION

Maestro CLI is a client for EPAM Cloud Orchestrator, supporting both HTTP and HTTPS as application layer protocols.

Maestro CLI commands are based on respective commands for Amazon AWS. We picked a minimum required set of parameters for each command. This way we were able to uniform the commands for different service providers.

In order to execute CLI commands, the Orchestrator's command controller expects a `POST` request from a client.

Item	Value
CLI entry point	`/api/cli`
General address structure	`https://<orchestrator_host>[:<port>]/<orchestrator_entry_point>/api/cli?action=<action_name>`

Example

The following example shows executing the **describe instances** command on POC Orchestrator instance:

```
https://orchestration.EPAM.com/maestro2/api/cli?action=describe-instances
```

The **Accept** request header can be used to specify a desired server response format (application/json or application/xml). Server response contains a **Content-Type** header, specifying actual response format.

Currently the client side supports automatic processing of xml-answers only.

Depending on the transmitted value of the **action** parameter a certain command is executed.

Controller processes the following **action** values:

"describe-images"	"stop-instances"
"describe-instances"	"terminate-instances"
"reboot-instances"	"attach-volume"
"run-instances"	"create-attach-volume"
"start-instances"	"describe-volumes"
"describe-regions"	"describe-shapes"
"access"	"delete-volume"
"detach-volume"	"create-volume"

The following values for the **action** parameter will be supported in API versions to follow:

"allocate-address"	"reset-snapshot-attribute"
"associate-address"	"reset-image-attribute"
"disassociate-address"	"describe-keypairs"
"release-address"	"create-keypair"
"describe-addresses"	"delete-keypair"
"describe-availability-zones"	"import-keypair"
"create-image"	"get-password"
"deregister-image"	"describe-groups"
"describe-image-attribute"	"create-group"
"describe-instance-attribute"	"delete-group"
"modify-instance-attribute"	"describe-volume-attribute"
"modify-snapshot-attribute"	"describe-snapshot-attribute"
"modify-image-attribute"	"modify-volume-attribute"
"reset-instance-attribute"	"get-console-output"
"create-snapshot"	"delete-snapshot"
"describe-snapshots"	"instance-restore-from-snapshot"

The parameters required to launch a chosen command are transmitted as HTTP POST parameters.

For example, the parameter pointing at a virtualization region required for most of commands looks as follows:

```
region=<region_name>
```

The date of request creation corresponds to RFC822 and has the following layout:

```
Mon, 03 Sep 2012 13:08:07 GMT
```

CLI client version, user ID, and creation dates for request and authorization token are specified using HTTP Request Headers:

Item	Header
API version	Maestro-api-version=<api version>
User ID	Maestro-access-id=<valid user id>
Date	Maestro-date=<RFC822 GMT Date>
Authorization token	Maestro-authorization=<valid authorization token>
Accept	application/json OR application/xml*

*CLI client supports xml-answers only

1.2 MAESTRO CLI AND API REQUEST EXAMPLES

1.2.1 Maestro CLI Command Example:

```
>or2din --project sample --region aws-useast

describe-instances "--project" "sample" "--region" "aws-useast"

Response:


        instance:

            instanceID = i-3d4fb640

            zone = us-east-1a

            state = stopped
```

1.2.2 API Request Example:

Method: POST

URL: https://orchestration.EPAM.com/maestro2/api/cli?action=describe-instances

Request Parameters:

```
region=aws-useast

project-id=sample
```

Request Headers:

```
Maestro-access-id: demo

Maestro-api-version: 1.0

Maestro-date: Fri, 28 Sep 2012 11:50:45 GMT

Maestro-authorization: igwBTAc63wXagPGoQkqWJbSQbswD3d/g9ChwjfNWCQ4=

Accept: application/xml
```

Response example:

```
<?xml version="1.0" encoding="UTF-8"?>

     <describe-instances-response>

           <instance instanceID="i-c9413bb2"

           dnsName="ec2-204-236-250-219.compute-1.amazonaws.com"

           publicIp="204.236.250.219"
```

```
            privateIp="10.192.65.148"

            zone="us-east-1c"

            state="running"/>

            <instance instanceID="i-cf413bb4"

            zone="us-east-1c"

            state="stopped"/>

      </describe-instances-response>
```

2 SETUP EXTRAS

The detailed Maestro-CLI installation and configuration guidelines are contained in Quick Start Guide.

Having downloaded and unpacked the Maestro CLI installation archive, you will get the following subfolders:

2.1 BIN SUBFOLDER

This folder contains scripts for command launching:

- cmd – scripts for Windows
- sh – scripts for *nix operation systems)

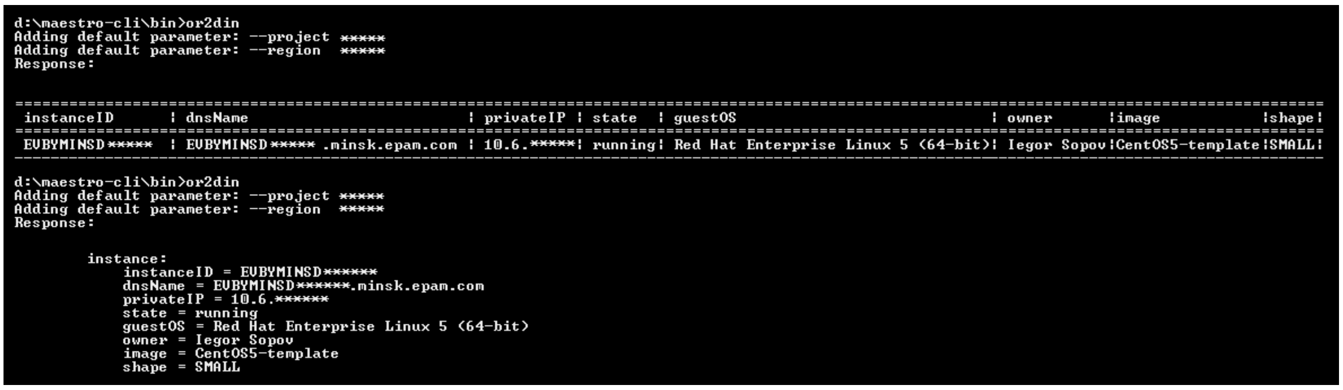
2.2 LIB SUBFOLDER

This folder contains the 'maestro-cli-full.jar' file with all required 3rd party libraries. There are also several configuration files, namely:

- **cli.properties** – common configuration
- **cli-http-client.properties** – HTTP client fine-tuning
  (See Section 4 - HTTP Client Fine-Tune)
- example-template.ftl – Freemarker template example
  (See Section 7 – Maestro CLI Output Customization)

2.3 RESPONSE CUSTOMIZATION

Setting the '**use.table.output'** value within the cli.properties configuration file to '**true'** (default value) allows using table layout for command responses (See the top part of the screenshot below).



```
d:\maestro-cli\bin>or2din
Adding default parameter: --project *****
Adding default parameter: --region  *****
Response:


==========================================================================================================================================
 instanceID       ¦ dnsName                          ¦ privateIP ¦ state   ¦ guestOS                             ¦ owner       ¦image             ¦shape¦
==========================================================================================================================================
 EVBYMINSD*****   ¦ EVBYMINSD***** .minsk.epam.com   ¦ 10.6.*****¦ running¦ Red Hat Enterprise Linux 5 (64-bit)¦ Iegor Sopov¦CentOS5-template¦SMALL¦
------------------------------------------------------------------------------------------------------------------------------------------

d:\maestro-cli\bin>or2din
Adding default parameter: --project *****
Adding default parameter: --region  *****
Response:

        instance:
            instanceID = EVBYMINSD******
            dnsName = EVBYMINSD******.minsk.epam.com
            privateIP = 10.6.******
            state = running
            guestOS = Red Hat Enterprise Linux 5 (64-bit)
            owner = Iegor Sopov
            image = CentOS5-template
            shape = SMALL
```

Figure 1 - Table and Plain layouts

When the property is set to '**false'**, responses will be output as plain text (see bottom part of the screenshot above).

2.4 DEFAULT REGION AND PROJECT

This functionality allows you to configure default region and project so that you don't have to provide **-p/--project** and **-r /--region** parameters each time you run a CLI command.

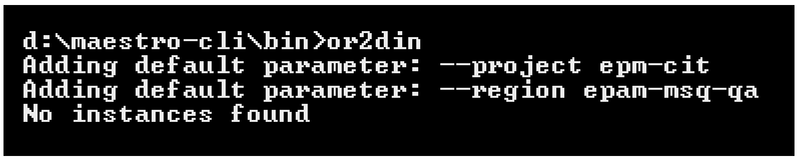
To use default parameters, please follow these steps:

1. Create a text file titled '**default.properties**' within the '**lib**' subfolder (the one containing **maestro-cli-full.jar**).
2. Edit the '**default.properties**' file; add region and project using full parameter names and values in the java properties format (key=value).Multiple entries are allowed. Save and close the file.
   For example:

```
region=EPAM-msq-qa
project=epm-test
```

Please do not use short parameter names here

3. Run a CLI command requiring both region and project to be specified (e.g. **or2din**) without specifying these parameters:



```
d:\maestro-cli\bin>or2din
Adding default parameter: --project epm-cit
Adding default parameter: --region epam-msq-qa
No instances found
```

Figure 2 - Running a command with default parameters

3 CLI CLIENT AUTHORIZATION

3.1 AUTHORIZATION DATA ALLOCATION

Running a command using Maestro CLI requires provision of user authorization data. For your convenience, there are several authorization data presentation models:

- Create a text file containing authorization data. Use the '**-cr**' CLI parameter to provide full path to it:

  ```
  or2din -r aws-useast -p sample -cr D:/user1.cr
  ```

  The file should contain a key identifier ('**id**' property) and the key token. For example, the file with user token can contain the following:

  ```
  id=demo_id

  token=fe89cb2d191600e600ed95920759c0
  ```

- In case you didn't provide a path to the file with authorization data, Maestro CLI will try to use authorization data from the '**default.cr**' file, which should be placed to the same folder as '**maestro-cli-full.jar**'.

  You can create the '**default.cr**' file on your own. To do this execute the '**or2access**' command and provide your PMC login and password when prompted. Currently this is the only way you can use your PMC credentials to authorize in Maestro CLI, as it makes Orchestrator check validity of user credentials and create a list of PMC projects available to you.

Detailed description of this command is contained in Maestro CLI User Guide.

The second method is preferable.

3.2 AUTHORIZATION TOKEN CALCULATION ALGORITHM

A Java code example illustrating generation of Authorization token for the current API version is always available from our Knowledge Base.

Maestro CLI client performs the following actions to calculate the '**Maestro-authorization**' token:

1. Obtain valid user **id** and **token** (user access token is generated on the server side when you run **or2-get-access** command and is present in the default.cr file).

2. Form a hashString:

```
hashString=<HTTP_METHOD>:<Maestro-access-id>:<Maestro-date>:<Query>
```

The '**Query**' element contains action name and other URL parameters.

These parameters should be sorted in alphabetical order by parameter name.

Query element example:

```
“action=describe-instances:instances=EVBYMINSD******,
EVBYMINSD******:project-id=sample:region=sample”
```

hashString example:

```
“POST:john_doe:Thu, 31 Jan 2013 00:00:00 GMT:action=describe-
instances:describe-instances:instances=EVBYMINSD******,
EVBYMINSD******:project-id=sample:region=sample”
```

3. Generate **key** from the credentials **token** concatenated with the **Maestro-date** header value using the **'javax.crypto.spec.SecretKeySpec'** class.

```
keyString = token.concat(<Maestro-date>)

key = new SecretKeySpec(keyString.getBytes(“UTF-8”), “HmacSHA256”)
```

4. Calculate the signature, following the *SHA256* algorithm (http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf), using the '**javax.crypto.Mac**' class.

```
signature = mac(“HmacSHA256”, hashString, key)
```

Some Java invocations example:

```
Mac mac = Mac.getInstance(“HmacSHA256”);

mac.init(key);

byte[] bytes = hashString.getBytes(“UTF-8”);

            byte[] signature = mac.doFinal(bytes);
```

5. Transform the calculated signature value into a string after *Base64* encoding:

```
Maestro-authorization = new String (Base64(signature))
```

After that, the ‘**Maestro-authorization**’ value is set for **HTTP Request Headers**.

The server side performs similar conversions to check validity of the received ‘**Maestro-authorization**’ header using the date from ‘**Maestro-date**’. The time of request creation is also checked. If the value contained in **Maestro-date**, different from current server time by more than 5 minutes (taking into account conversion to GMT), the request is rejected.

4 HTTP CLIENT FINE-TUNE

Maestro CLI uses Apache HTTP Client 4 as a HTTP request organization client, which allows fine-tuning networking parameters.

HttpClient supports a preference API based on HttpParams. All major components of the HttpClient toolkit (agents, host configurations, methods, connections, connection managers) contain a collection of HTTP parameters, which determine the runtime behavior of those components.

During execution of an HTTP request HttpParams of the HttpRequest object are linked with HttpParams of an HttpClient instance, used to execute the request. This enables parameters set at the HTTP request level to take precedence over HttpParams set at the HTTP client level.

See the full list of supported parameters at: http://hc.apache.org/httpcomponents-client-ga/tutorial/html/.

In order to edit default properties of HTTP client provide required values for parameters within the '**cli-http-client.properties'** file, which should be placed to the same folder as '**maestro-cli-full.jar'**. If the '**cli-http-client.properties**' file is missing from the folder, HTTP Client uses default parameters.

To check parameter values within the '**cli-http-client.properties**' configuration file (if it contains specified values), run the '**or2info**' command.

The '**or2info**' command description is contained in Maestro CLI User Guide.

#	Parameter Name	Description	Value Type
1	**http.socket.timeout**	Defines the socket timeout (`SO_TIMEOUT`) in milliseconds, which is the maximum inactivity period between two consecutive data packets. A timeout value of zero is interpreted as an infinite timeout. See **`java.net.SocketOptions#SO_TIMEOUT`**	Integer
2	**http.tcp.nodelay**	Determines whether *Nagle*'s algorithm is to be used. The Nagle's algorithm tries to conserve bandwidth by minimizing the number of segments that are sent. When applications wish to decrease network latency and increase performance, they can disable Nagle's algorithm (that is enable `TCP_NODELAY`). Data will be sent earlier, at the cost of an increase in bandwidth consumption. See **`java.net.SocketOptions#TCP_NODELAY`**	Boolean
3	**http.socket.buffer-size**	Determines the size of the internal socket buffer used to buffer data while receiving / transmitting HTTP messages.	Integer
4	**http.socket.linger**	Sets `SO_LINGER` with the specified linger time in seconds. The maximum timeout value is platform specific. Value **0** implies that the option is disabled. Value **-1** implies that the JRE default is used. The setting only affects the socket close operation. See **`java.net.SocketOptions#SO_LINGER`**	Integer

#	Parameter Name	Description	Value Type
5	**http.socket.reuseaddr**	Defines whether the socket can be bound even though a previous connection is still in a timeout state. See **`java.net.Socket#setReuseAddress(boolean)`** ⚠ *Since version 4.1*	Boolean
6	**http.connection.timeout**	Determines the timeout in milliseconds until a connection is established. A timeout value of zero is interpreted as an infinite timeout. ⚠ Please note this parameter can only be applied to connections that are bound to a particular local address.	Integer
7	**http.connection.stalecheck**	Determines whether stale connection check is to be used. The stale connection check can cause up to 30 millisecond overhead per request and should be used only when appropriate. For performance critical operations this check should be disabled.	Boolean
8	**http.connection.max-line-length**	Determines the maximum line length limit. If set to a positive value, any HTTP line exceeding this limit will cause an `IOException`. A negative or zero value will effectively disable the check.	Integer
9	**http.connection.max-header-count**	Determines the maximum HTTP header count allowed. If set to a positive value, the number of HTTP headers received from the data stream exceeding this limit will cause an `IOException`. A negative or zero value will effectively disable the check.	Integer
10	**http.connection.min-chunk-limit**	Defines the size limit below which data chunks should be buffered in a session I/O buffer in order to minimize native method invocations on the underlying network socket. The optimal value of this parameter can be platform specific and defines a trade-off between performance of memory copy operations and that of native method invocation. ⚠ Since version 4.1	Integer

11	**http.protocol.version.major**	Defines the `ProtocolVersion` used per default.
		Default:
		`HttpVersion.HTTP_1_1`
		`Set http.protocol.version.major=1, http.protocol.version.minor=1 for HTTP_1_1`
			Integer

12	**http.protocol.version.minor**		
13	**http.protocol.element-charset**	Defines the charset to be used for encoding HTTP protocol elements.	String
14	**http.protocol.content-charset**	Defines the charset to be used per default for encoding content body.	String
15	**http.useragent**	Defines the content of the `User-Agent` header.	String
16	**http.origin-server**	Defines the content of the `Server` header.	String
17	**http.protocol.strict-transfer-encoding**	Defines whether responses with an invalid `Transfer-Encoding` header should be rejected.	Boolean

5 LOG SUBSYSTEM

Maestro CLI uses Apache Log4j 1.2 library as a log subsystem http://logging.apache.org/log4j/1.2/.

With log4j it is possible to enable logging at runtime without modifying the application binary. The log4j package is designed so that these statements can remain in shipped code without incurring a heavy performance cost. Logging behavior can be controlled by editing a configuration file, without touching the application binary.

Logging equips the developer with detailed context for application failures. On the other hand, testing provides quality assurance and confidence in the application. Logging and testing should not be confused. They are complementary. When logging is used wisely, it can prove to be an essential tool.

One of the distinctive features of log4j is the notion of inheritance in loggers. Using a logger hierarchy it is possible to control which log statements are output at arbitrarily fine granularity but also great ease. This helps to reduce the volume of logged output and the cost of logging.

The target of the log output can be a file, an OutputStream, a java.io.Writer, a remote log4j server, a remote Unix Syslog daemon, or many other output targets.

In some cases, it may be useful to see the exact requests and responses being sent and received by Maestro CLI. This logging should not be enabled in production systems since writing out large requests or responses can significantly slow down an application. If you really need access to this information, you can temporarily enable it through Apache HttpClient's logger. Enabling the DEBUG level on the '**org.apache.http.wire**' logger enables logging for all request and response data.

The following '**log4j.xml**' file entry turns on full wire logging in Apache HttpClient and should only be turned on temporarily since it can have a significant performance impact on your application.

```
<!-- HttpClient logger -->

<logger name="org.apache.http.wire" additivity="false">

            <level value="debug"/>

            <appender-ref ref="console"/>

      </logger>
```

The configuration above sends all logged HTTP Client's actions into CLI. If you configure the Log4j FileAppender, you can send logs to a file. For example:

```
<appender name="fileAppender" class="org.apache.log4j.FileAppender">

      <param name="File" value="sample.log"/>

      <layout class="org.apache.log4j.PatternLayout">

            <param name="ConversionPattern" value= value="[%p,%c{1}] %m%n" />

      </layout>

</appender>
```

You should also change configuration of HttpClient's logger, specifying the '**appender-ref "fileAppender"**' instead of '**appender-ref "console"**':

```
<!-- HttpClient logger -->

      <logger name="org.apache.http.wire" additivity="false">

            <level value="debug"/>

            <appender-ref ref="fileAppender"/>

      </logger>
```

For you convenience Maestro CLI supports management of the aforementioned mode via the '**cli.properties**' configuration file.

Enabling HTTP Client wire logs requires setting the '**http.client.verbose.mode**' property to '**true**'.

Receiving Maestro CLI debug level logging requires setting the '**maestro.debug.mode**' property to '**true**'.

In order to determine user Log4j conversion pattern set the '**log4j.conversion.pattern**' property.

Conversion pattern format is defined as per the following document:

http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html

You can also fully reconfigure Log4j by specifying the name of the valid configuration file, which should be placed to the same folder as '**maestro-cli-full.jar**'. The file should correspond to XML configuration file formatting rules for Lo4j version, used in Maestro-CLI (currently - 1.2.16). Provide the name of Log4j user configuration file as a value for the '**custom.log4j.config.file**' parameter and place the file itself to the 'lib' folder (the one with '**maestro-cli-full.jar**'.

To check parameter values within the configuration file, run the '**or2info**' command, described in

Below is a part of the '**cli.properties**' file, dealing with logging subsystem fine tuning. The example contains commented property values examples.

```
# Turns on debug mode for all com.maestro log4j loggers

maestro.debug.mode=false
```

```
# Turns on debug mode for the org.apache.http.wire log4j logger

http.client.verbose.mode=false

# Custom Log4j conversion pattern, for example, # %d{ABSOLUTE} [%p,%c{1}]
%m%n

# Leave blank for default configuration

log4j.conversion.pattern=

# Set path to the custom Log4j configuration file, for example, # user-log4j-
config.xml

# This property overrides properties: maestro.debug.mode,

# http.client.verbose.mode, log4j.conversion.pattern.

# Leave blank for default configuration

custom.log4j.config.file=
```

6 MAESTRO CLI COMMANDS

Detailed Maestro-CLI command reference is contained in Maestro CLI User Guide

6.1 EXIT CODES

Each execution of a Maestro-CLI command returns an exit code. Exit codes help you understand what exactly went wrong during command execution. You can see the exit code for the command by entering the following line in your command line immediately after running a Maestro-CLI command:

```
echo %ERRORLEVEL%
```

The list of all exit codes is provided in the table below:

Error code	Unix error code description	Server response	Possible causes, CLIException
0	Success	200 OK (HTTP/1.0 - RFC 1945)	Successful command execution
1	Operation not permitted	Any not described below	Not specified execution fail, scripts exit code 1
13	Permission denied	N/A	Cannot retrieve user credentials
		401 Unauthorized (HTTP/1.0 - RFC 1945)	UnauthorizedException
22	Invalid argument	NA	CLI arguments parsing error: invalid or missing JCommander CLI arguments, environment variables or properties values
		400 Bad Request (HTTP/1.1 - RFC 2616)	BadRequestException
			CliValidationException
			InvalidRequestException
41	Unknown error 41	403 Forbidden (HTTP/1.0 - RFC 1945)	ForbiddenException
58	Unknown error 58	405 Method Not Allowed (HTTP/1.1 - RFC 2616)	NotAllowedException
61	No data available	204 No Content (HTTP/1.0 - RFC 1945)	No data for the specified request is available
70	Communication error on send	Any	HTTPResponse from HTTPClient is null
111	Connection refused	NA	java.net.UnknownHostException
		NA	org.apache.http.conn.HttpHostConnectException
133	Unknown error 133	404 Not Found (HTTP/1.0 - RFC 1945)	NotFoundException
134	Unknown error 134	500 Server Error (HTTP/1.0 - RFC 1945)	CliException
135	Unknown error 135	NA	Cannot get maestro-cli client version
		NA	Invalid response content type (expecting application/xml)
		406 Not Acceptable (HTTP/1.1 - RFC 2616)	InvalidCliVersionException

Error code	Unix error code description	Server response	Possible causes, CLIException
136	Unknown error 136	417 Expectation Failed (HTTP/1.1 - RFC 2616)	CliVirtualizationException
137	Unknown error 137	412 Precondition Failed	Action confirmation required
138	Unknown error 138	302 Moved Temporarily (HTTP/1.0 - RFC 1945)	EPAM Orchestrator maintenance is in progress

7 MAESTRO CLI OUTPUT CUSTOMIZATION

7.1 FREEMARKER ENGINE

Maestro CLI uses a template engine based on FreeMarker template engine (http://freemarker.org/) to display command execution results.

FreeMarker template engine is a generic tool to generate text output (anything from HTML to auto generated source code) based on templates. It's a Java package, a class library for Java programmers. It's not an application for end-users in itself, but something that programmers can embed into their products.

FreeMarker is designed to be practical for the generation of HTML Web pages, particularly by servlet-based applications following the MVC (Model View Controller) pattern. The idea behind using the MVC pattern for dynamic Web pages is that you separate the designers (HTML authors) from the programmers. Everybody works on what they are good at. Designers can change the appearance of a page without programmers having to change or recompile code, because the application logic (Java programs) and page design (FreeMarker templates) are separated. Templates do not become polluted with complex program fragments. This separation is useful even for projects where the programmer and the HTML page author is the same person, since it helps to keep the application clear and easily maintainable

FreeMarker allows defining output format by a template, being a text file with a certain layout. For instance, displaying results of Maestro CLI request execution is done using the following default template:

```
Response:
<#if Response.attributes??>
  <#list Response.attributes as attribute>
    <#if attribute.first == "PCDATA">
        ${attribute.second}
    <#else>
          ${attribute.first} = ${attribute.second}
    </#if>
  </#list>
</#if>
<#if Response.items??>
  <#list Response.items as item>

      ${item.name}:
      <#if item.attributes??>
        <#list item.attributes as attribute>
        <#if attribute.first == "PCDATA">
            ${attribute.second}
        <#else>
            ${attribute.first} = ${attribute.second}
        </#if>
        </#list>
      </#if>
  </#list>
</#if>
```

If given the ability to specify the name of a user template in the configuration file, users will be able to get responses in different formats, e.g. HTML reports.

In order to use Freemarker user template, place it to the **Lib** folder, already containing the **maestro-cli-full.jar** file. There is also a sample template (**example-template.ftl**), identical to the default template (see above).

Once you create a valid user template, specify its name in the '**cli.properties**' configuration file, as a value for the **'custom.freemarker.template.file**' property. If the system is unable to read the file, it will return display a respective warning in the console. In this case a default template will be used as a Freemarker template.

TABLE OF FIGURES

VERSION HISTORY

Version	Date	Summary
4.2	April 17, 2021	− Updated link to EPAM Cloud Terms and Condition guide
4.1	December 16, 2016	− Classification changed from Confidential to Public, approved by Dzmitry Pliushch
4.0	December 19, 2015	− Updated to new formatting. − Removed Admin Suite section as outdated
3.22	November 28, 2013	− Added the Preface − Documentation links updated
3.21	March 16, 2013	− Restructured − A link to a new document is added
2.21	January 18, 2013	− API entry point link is updated
3.3	November 1, 2014	− Updated documentation references
3.2	February 26, 2013	− Authorization token description is updated − A request header omission is fixed
2.2	January 16, 2013	− Authorization info is revised − API request example is updated
3.1	February 18, 2013	− New exit codes are added
2.1	January 10, 2013	− Design is altered − Response customization is revised
3.0	February 9, 2013	− Major revision for Orchestration 2.0.23
2.0	December 11, 2012	− Admin Suite description is added
1.0	December 03, 2012	− Segregated from Maestro CLI Reference Guide − Revised and re-structured − Default properties functionality is described