DELPHIX

Delphix Masking Engine API Cookbook

February, 2018

Delphix Masking Engine API Cookbook
You can find the most up-to-date technical documentation at:
docs.delphix.com The Delphix Web site also provides the latest product updates.
If you have comments about this documentation, submit your feedback to: infodev@delphix.com

Delphix Corp.
1400 Seaport Blvd, Suite 200
Redwood City, CA 94063

Masking API Cookbook

The following pages will describe how to use the Masking API client to automate your masking activities using API calls.

API Calls for Masking Administration

The Delphix Masking Engine supports the following two types of APIs:

- **Analytics APIs**
  These APIs are for including Masking performance information in the support bundle, and do not need to be used unless that information is requested.
- **Application Setting APIs**
  Application Setting APIs allow an administrator to change the Delphix Masking Engine settings. Presently there are two categories of settings: general settings and analytics settings. Over time, more settings will be added to give users direct control over the product's various settings.

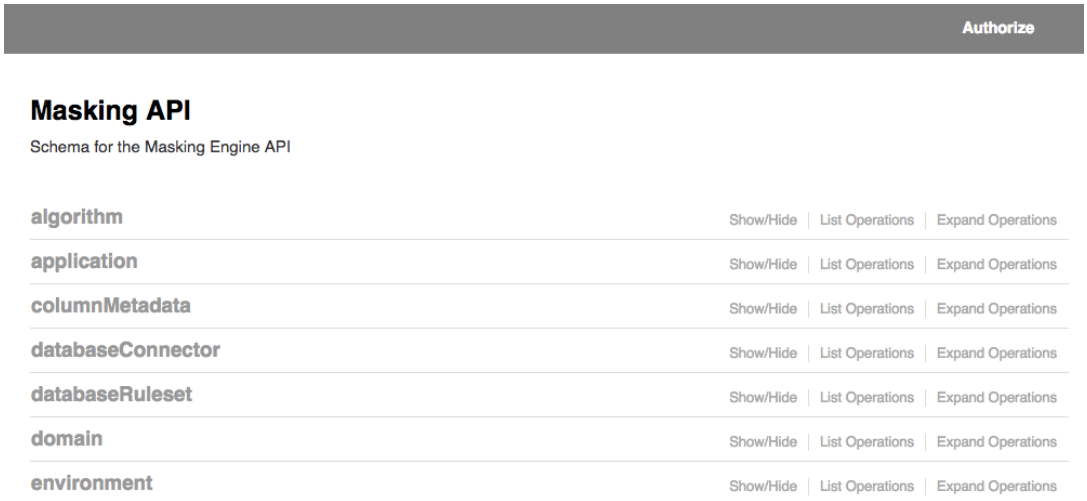
The Masking API Client

The Delphix Masking Engine now features an interactive API client that can generate commands specific to your masking engine. With those commands, you can:

- make changes to your engine
- copy and paste the commands to write code that can automate your masking activities

> **The API client will make real changes to your virtual machine. Any operations you run using the API Client will persist on the machine!**

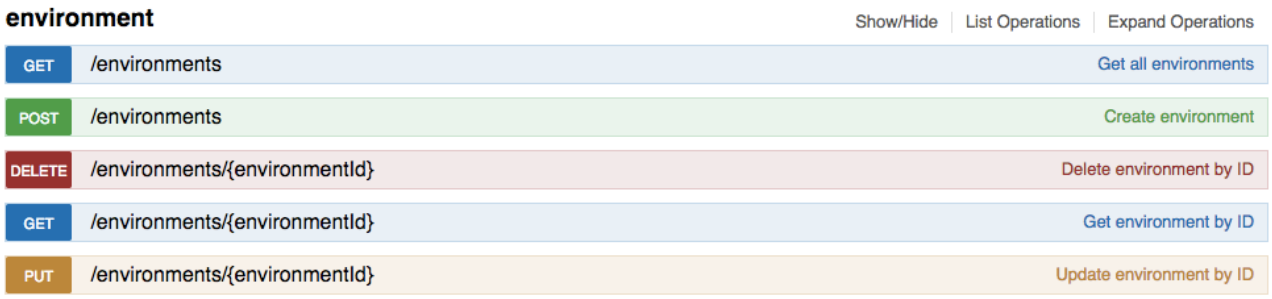
To access the Masking API client, use the following URL: http://myMaskingEngine.com:8282/masking/api-client/, replacing "myMaskingEngine.com" with the hostname or IP address of your virtual machine.



Example screen for the Masking API Client

The masking API client displays a list of masking components, such as algorithm, domain, and environment. You can expand each component for more information about operations relating to it, which you can use APIs to perform.

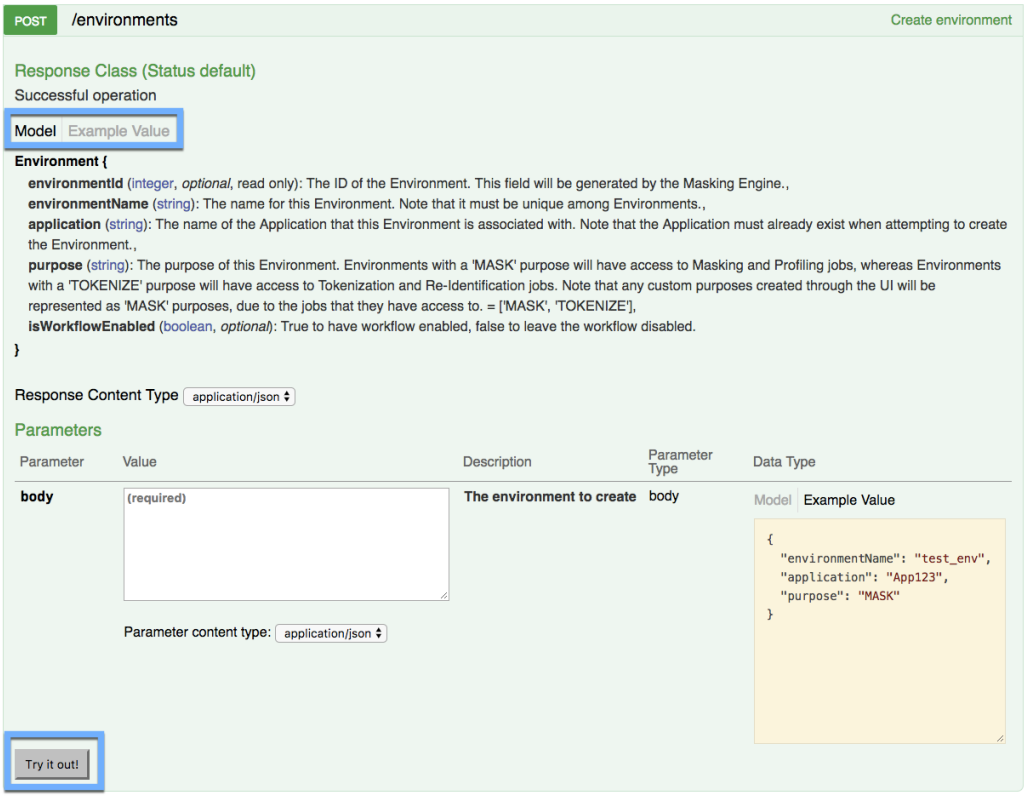
- **Show/Hide** – Expands or collapses detailed information about the masking component
- **List Operations** – Displays a simple list of operations related to the masking component, or hides that list if already displayed
- **Expand Operations** – Opens interactive panels for all operations relating to the masking component



*List of operations available for **environment***

Expanded operation panels

When you expand one or more operations, a panel will display detailed information about creating API calls for that operation. You can view either a **Model** of how to construct your code, as in the screenshot below, or an **Example Value**. An example value also appears in the lower right-hand corner of the panel. Clicking the **Example Value** box will populate that example into the **Value** field.



Expanded panel for creating an environment, showing model

To perform this operation on your virtual machine, click **Try it out!** at the bottom of the panel.

> Clicking **Try it out!** will make real changes to your virtual machine. Any operations you run using the API Client will persist on the machine!

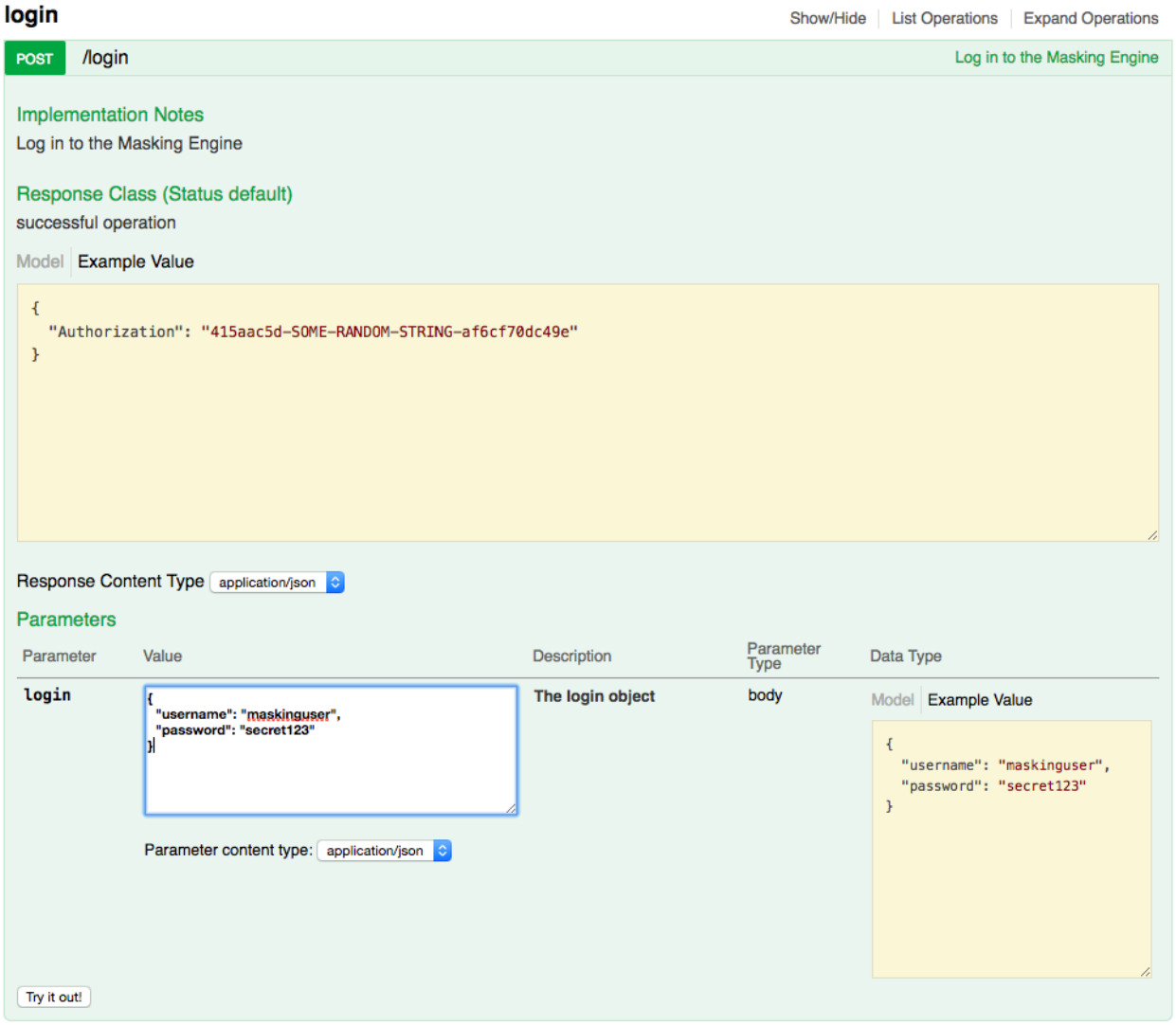
Logging into the API Client

This section will walk you through authorizing the API Client.

> All subsequent operations assume that you are authorized. In the examples that follow, you can see the presence of the “Authorization” string in the headers.

To access the Masking API client, use the following
URL: http://<myMaskingEngine.com>:8282/masking/api-client/

1. In the list of operations, click **login**.
2. If necessary, click again to expand.
3. For **Value**, enter the username and password. You can also pre-populate the JSON structure by clicking the **Example Value** in the lower right-hand corner.
4. Once you have filled in the username and password, click **Try it out!**



5. Upon a successful login, an Authorization string will appear. Copy the authorization string, not including the quotation marks.

6. Scroll to the very top of the API Client screen and click **Authorize** in the upper right-hand corner of the page.
7. When the **Available authorizations** dialog opens, paste your Authorization string from step 5 into the **V alue** field.

8. Click **Authorize**. The page will reload and all subsequent API calls for this session will automatically add your Authorization string to the header.

Automating Login

If you want to automate the authorization process, the following are examples of commands you might enter and receive.

REQUEST

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json'-d '{ "username": "myUsername", "password": "myPassword" }' 'http://<myMaskingEngine>:8282/masking/api/login'

RESPONSE

{"Authorization":"d32abf35-1097-419b-bfa2-d75f23eab602"}

More info

For more information about commands that will help you automate logging in, use the API client on your virtual machine:

http://<myMaskingEngine>:8282/masking/api-client/#!/login/login

> In all code examples, replace <myMaskingEngine> with the hostname or IP address of your virtual machine.

Related Links

- **Masking API Cookbook**
- **Sample Cookbook Scripts**

API Calls for Preparing to Mask

Below are examples of requests you might enter and responses you might receive from the Masking API client. For commands specific to your masking engine, work with your interactive client at http://<**myMaskingEngine>**:8282/masking/api-client/.

> In all code examples, replace **<myMaskingEngine>** with the hostname or IP address of your virtual machine.

> Using HTTP (as shown in these examples) is insecure. HTTPS (SSL/TLS) is recommend but for explanatory purposes the following examples use insecure HTTP.

Creating an Application

REQUEST

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization: d1313dd8-2ed9-4699-8e88-2b6a089ae2a6'
-d '{ "applicationName": "App123" }' 'http://<myMaskingEngine>:8282/masking/api/a
```

```
pplications'
```

RESPONSE

```
{ "applicationName": "App123" }
```

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/application/createApplication

Example

See how to use this in the context of a script **here**.

Creating an Environment

Object references you will need:

- The name of the application where you would like to create the environment

REQUEST

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization: d1313dd8-2ed9-4699-8e88-2b6a089ae2a6'
-d '{ "environmentName": "test_env", "application": "App123", "purpose": "MASK"
}' 'http://<myMaskingEngine>:8282/masking/api/environments'
```

RESPONSE

```
{ "environmentId": 2, "environmentName": "test_env", "application": "App123",
"purpose": "MASK", "isWorkflowEnabled": false }
```

Note the inclusion of the ID (“environmentId”) of the newly created object. This ID is used to reference this environment elsewhere.

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/environment/createEnvironment

Example

See how to use this in the context of a script **here**.

Creating a Database Connector

Object references you will need:

- The ID of the environment under which you want to create the connector

REQUEST

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization: e23bad24-8760-4091-a131-34f235d9b2d6'
-d '{ "connectorName": "connector", "databaseType": "ORACLE", "environmentId":
72, "host": "myHost", "password": "myPassword", "port": 1234, "schemaName":
"mySchema", "sid": "mySID", "username": "myUsername" }' 'http://<myMaskingEngine>
:8282/masking/api/database-connectors'
```

The included parameters in this example are specific to creating an Oracle connector. For more details of the various parameters for different types of database connectors, please consult the documentation at http://<myMaskingEngine>:8282/masking/api-client/ and navigating to the “POST /database-connectors” section.

RESPONSE

```
{ "databaseConnectorId": 1, "connectorName": "connector", "databaseType":
"ORACLE", "environmentId": 72, "databaseName": "mySchema", "host": "myHost",
"port": 1234, "schemaName": "mySchema", "sid": "mySID", "username": "myUsername",
"useKerberosAuth": false }
```

> Note the inclusion of the ID (“databaseConnectorId”) in the response for the newly created object. This ID is used to reference this database connector elsewhere.

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/connector/createDatabaseConnector

Example

See how to use this in the context of a script **here**.

Creating a Rule Set

Object references you will need:

- The ID of the database connector that the ruleset corresponds to

REQUEST

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization: 7c856e3d-5b20-4261-b5fe-cc2ffcee5ae0'
-d '{ "rulesetName": "myRuleset", "connectorId": 1 }' 'http://<myMaskingEngine>:8
```

```
282/masking/api/database-rulesets'
```

RESPONSE

```
{ "rulesetId": 2, "rulesetName": "myRuleset", "connectorId": 1, "environmentId":
1 }
```

Note that the response includes the ID of the newly created ruleset (“rulesetId”).

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/databaseRuleset/createDatabaseRuleset

Example

See how to use this in the context of a script **here**.

Related Links

- **Masking API Cookbook**
- **Sample Cookbook Scripts**

API Calls for Creating an Inventory

Below are examples of requests you might enter and responses you might receive from the Masking API client. For commands specific to your masking engine, work with your interactive client at http://**<myMaskingEngine >:8282**/masking/api-client/

> HTTPS (SSL/TLS) is recommended, but for explanatory purposes these examples use insecure HTTP

> In all code examples, replace **<myMaskingEngine>** with the hostname or IP address of your virtual machine.

Fetch Table Names from Database Connector

Object references you will need:

- The ID of the database connector to fetch tables for

> Note that this database connector ID (1, in this example) is included in the PATH for this operation, NOT the payload

REQUEST

```
curl -X GET --header 'Accept: application/json' --header 'Authorization:
7c856e3d-5b20-4261-b5fe-cc2ffcee5ae0'  'http://<myMaskingEngine>:8282/masking/api
/database-connectors/1/fetch'
```

RESPONSE

```
[ "ALL_COLUMNS", "DBVERIFICATION_TABLE" ]
```

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/databaseConnector/fetchTableMetadata

Example

See how to use this in the context of a script **here**.

Create Table Metadata

Object references you will need:

- The name of the table to create the metadata for
- The ruleset ID

REQUEST

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization: 7c856e3d-5b20-4261-b5fe-cc2ffcee5ae0'
-d '{ "tableName": "ALL_COLUMNS", "rulesetId": 2 }'  'http://<myMaskingEngine>:82
82/masking/api/table-metadata'
```

RESPONSE

```
{ "tableMetadataId": 2, "tableName": "ALL_COLUMNS", "rulesetId": 2 }
```

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/tableMetadata/createTableMetadata

Example

See how to use this in the context of a script **here**.

Get All Column Metadata Belonging to Table Metadata

Object references you will need:

- The table metadata ID to get the columns for

> Note that this table metadata ID (2, in this example) is included in the QUERY STRING for this operation, NOT the payload

REQUEST

```
curl -X GET --header 'Accept: application/json' --header 'Authorization:
7c856e3d-5b20-4261-b5fe-cc2ffcee5ae0' 'http://<myMaskingEngine>:8282/masking/api/
column-metadata?table_metadata_id=2'
```

RESPONSE

```
[ { "columnMetadataId": 12,  "columnName": "schoolnme",  "tableMetadataId": 2,
"columnLength": 50, "isMasked": false, "isPrimaryKey": false, "isIndex": false,
"isForeignKey": false }, … ]
```

Note that the above response has been truncated due to its length for the purposes of this documentation.

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/columnMetadata/getAllColumnMetadata

Example

See how to use this in the context of a script **here**.

Update Column Metadata with Algorithm Assignment

Object references you will need:

- Column metadata ID for the column you wish to update

  > Note that this column metadata ID (20, in this example) is included in the PATH for this operation, NOT the payload

  Since the names can vary in the API and UI, you should use the names obtained through the API (these may not align with the UI).
- Algorithm name
- Domain name

REQUEST

```
curl -X PUT --header 'Content-Type: application/json' --header 'Accept:
```

```
application/json' --header 'Authorization: 7c856e3d-5b20-4261-b5fe-cc2ffcee5ae0'
-d '{ "algorithmName": "AddrLine2Lookup",  "domainName": "ADDRESS_LINE2" }'  'htt
p://<myMaskingEngine>:8282/masking/api/column-metadata/20'
```

RESPONSE

```
{ "columnMetadataId": 20, "columnName": "l2_address", "tableMetadataId": 2,
"algorithmName": "AddrLine2Lookup", "domainName": "ADDRESS_LINE2",
"columnLength": 512, "isMasked": true, "isPrimaryKey": false, "isIndex": false,
"isForeignKey": false }
```

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/columnMetadata/updateColumnMetadata

Example

See how to use this in the context of a script **here**.

Related Links

- **Masking API Cookbook**
- **Sample Cookbook Scripts**

API Calls for Creating and Running Masking Jobs

Below are examples of requests you might enter and responses you might receive from the Masking API client. For commands specific to your masking engine, work with your interactive client at http://**<myMaskingEngine>**:8282/masking/api-client/

> In all code examples, replace **<myMaskingEngine>** with the hostname or IP address of your virtual machine.

> HTTPS (SSL/TLS) is recommended, but for explanatory purposes these examples use insecure HTTP.

Creating a Masking Job

Object references you will need:

- The ID of the ruleset for which you wish to create the masking job

REQUEST

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --header 'Authorization: e23bad24-8760-4091-a131-34f235d9b2d6' -d '{ "jobName": "some_masking_job", "rulesetId": 7, "jobDescription": "This example illustrates a MaskingJob with just a handful of the possible fields set. It is meant to exemplify a simple JSON body that can be passed to the endpoint to create a MaskingJob.", "feedbackSize": 100000, "onTheFlyMasking": false }'  'http://<myMaskingEngine>:8282/masking/api/masking-jobs'
```

RESPONSE

```
{ "jobId": 1, "jobName": "some_masking_job", "rulesetId": 7, "createdBy": "Axistech", "createdTime": "2017-07-04T00:31:00.952+0000", "environmentId": 2,
```

```
"feedbackSize": 100000, "jobDescription": "This example illustrates a MaskingJob
with just a handful of the possible fields set. It is meant to exemplify a simple
JSON body that can be passed to the endpoint to create a MaskingJob.",
"maxMemory": 1024, "minMemory": 1024, "multiTenant": false, "numInputStreams": 1,
"onTheFlyMasking": false }
```

> Note that the response includes the ID of the newly created job (“jobId”).

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/job/createMaskingJob

Running a Masking Job

Create a new execution of a masking job.

Object references you will need:

- The ID of the job you want to run

REQUEST

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization: e23bad24-8760-4091-a131-34f235d9b2d6'
-d '{ "jobId": 1 }' 'http://<myMaskingEngine>:8282/masking/api/executions'
```

RESPONSE

```
{ "executionId": 1, "jobId": 1, "status": "RUNNING" }
```

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/execution/createExecution

Checking the Status of a Masking Job

Object references you will need:

- The ID of the execution you want to check (IN THE PATH)

> Note that this execution id (1, in this example) is included in the PATH for this operation, NOT the payload.
>
> The executions endpoint only returns status for the most recent job run, this is the expected behavior. Although the Masking Service does not currently retain historical execution results, the API has been designed to allow for historical results to be returned in the future.

REQUEST

```
curl -X GET --header 'Accept: application/json' --header 'Authorization:
8935f7f7-6de6-40ba-80d8-d8956b71248b' 'http://<myMaskingEngine>:8282/masking/api/
executions/1'
```

RESPONSE

```
{ "executionId": 1, "jobId": 1, "status": "SUCCEEDED" }
```

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/execution/getExecutionById

Related Links

- **Masking API Cookbook**
- **Sample Cookbook Scripts**

API Calls involving File Upload

API calls involving file upload are noteworthy because the generated curl from the Masking API client will be **m issing the parameter referencing the file**; as such, those commands from the Masking API client **will not work**.

Instead, below are examples of working requests and responses for API calls involving file upload.

For commands specific to your masking engine, work with your interactive client at http://**<myMaskingEngine >**:8282/masking/api-client/

> HTTPS (SSL/TLS) is recommended, but for explanatory purposes these examples use insecure HTTP

> In all code examples, replace **<myMaskingEngine>** with the hostname or IP address of your virtual machine.

Creating a File Format

REQUEST

```
curl -X POST --header 'Content-Type: multipart/form-data' --header 'Accept:
application/json' --header 'Authorization: d1313dd8-2ed9-4699-8e88-2b6a089ae2a6'
-F fileFormat=@/path/to/file_format/delimited_format.txt -F
fileFormatType=DELIMITED 'http://<myMaskingEngine>:8282/masking/api/file-formats'
```

RESPONSE

```
{ "fileFormatId": 123, "fileFormatName": "delimited_format.txt",
"fileFormatType": "DELIMITED" }
```

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/fileFormat/createFileFormat

Creating an SSH Key

REQUEST

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
application/json' --header 'Authorization: d1313dd8-2ed9-4699-8e88-2b6a089ae2a6'
-F sshKey=@/path/to/ssh_key/this_file_name_is_your_ssh_key_name.txt 'http://<myMa
skingEngine>:8282/masking/api/ssh-keys'
```

RESPONSE

```
{ "sshKeyName": "this_file_name_is_your_ssh_key_name.txt" }
```

More info

http://<myMaskingEngine>:8282/masking/api-client/#!/sshKey/createSshKey

Related Links

- **Masking API Cookbook**
- **Sample Cookbook Scripts**

Backwards-Compatible API Usage

> In all examples, replace **<myMaskingEngine>** with the hostname or IP address of your virtual machine.

API Versioning Context

The Masking API being shipped with the 5.2 series of releases of the Delphix Masking Engine is version **v5.0.0** in accordance with the Semantic Versioning format: http://semver.org/. In subsequent maintenance and major releases of the Masking product, the Masking API may be updated and a new API version will be released *(e.g. **v5.0.1**, **v5.1.0**, etc).* As scripts using the new Masking API are being written, they must reference an explicit API version or else there are no guarantees that the scripts will work on future releases of the Masking product.

Pinning Down a Version Number To Guarantee Backwards-Compatibility

`'http://<myMaskingEngine>:8282/masking/api/v5.0.0/environments'`

This is the format for specifying a version in the URL of an API request targeting the **environments** endpoints. The only possible version value for the Masking API in the first 5.2 release is **v5.0.0**. As more releases of the Masking product are shipped in the future, the set of possible versions will expand.

Scripts that specifically pin down the version of the Masking API in the URL will continue to work upon future upgrades of the Masking product--even if a newer version of the API is available in the future Masking product–with the exception that **Incubating API Endpoints** are never guaranteed to be backwards-compatible.

For example, consider the scenario where a script is being developed today with a pinned down version **v5.0.0** in the URL of the API requests. Upon upgrade to a future release of the Masking product that has the API **v5.1.0** available, the same, untouched script that was developed with the pinned down version **v5.0.0** in the URL of the API requests is expected to continue working. That said, in order to leverage any new features of the API **v5.1.0**, the original script will need to be updated to specify the new API version in the URL, and the requests may need to be updated to conform to the new API specification.

Omitted Version Numbers

```
'http://<myMaskingEngine>:8282/masking/api/environments'
```

This is the format for not specifying a version in the URL of an API request targeting the **environments** endpoi

nts. When the API version number is omitted, the latest API version is taken as a default. In the first 5.2 release, an API request with an omitted version number will be interpreted as a request against the **v5.0.0** ver sion of the API. In a future release that hypothetically has the API **v5.3.0** available, an API request with an omitted version number will be interpreted as a request against the **v5.3.0** version of the API.

Scripts that omit the version of the Masking API in the URL are not guaranteed to work upon future upgrades of the Masking product because the API specification may change between versions, and requests that conform to the old API specification may not work on the new API specification.

Related Links

- **Masking API Cookbook**
- **Sample Cookbook Scripts**

Incubating API Endpoints

Context

APIs that are released across the industry are expected to have a stable specification that consumers can depend on when writing scripts and automation. This notion of a stable API specification is at odds with the natural process of iteration and refinement that a newly released feature is expected to undergo. As such, in order to accommodate the anticipated iteration and refinement of this newly released Masking API, Delphix is introducing the notion of Incubating API endpoints.

Definition

An Incubating API endpoint is available for immediate use, but the specification of an Incubating API endpoint is subject to change in the future *(i.e. the specification is not stable)*.

Backwards-Compatibility of Incubating API Endpoints

There are no backwards-compatibility guarantees when using Incubating API endpoints, even when **pinning down the API version number**.

That said, it is not the case that an Incubating API will *always* change in a future release, but rather that it *mighty* change in a future release such that any scripts that were developed to use an Incubating API would need to be updated to work against a future release of the API.

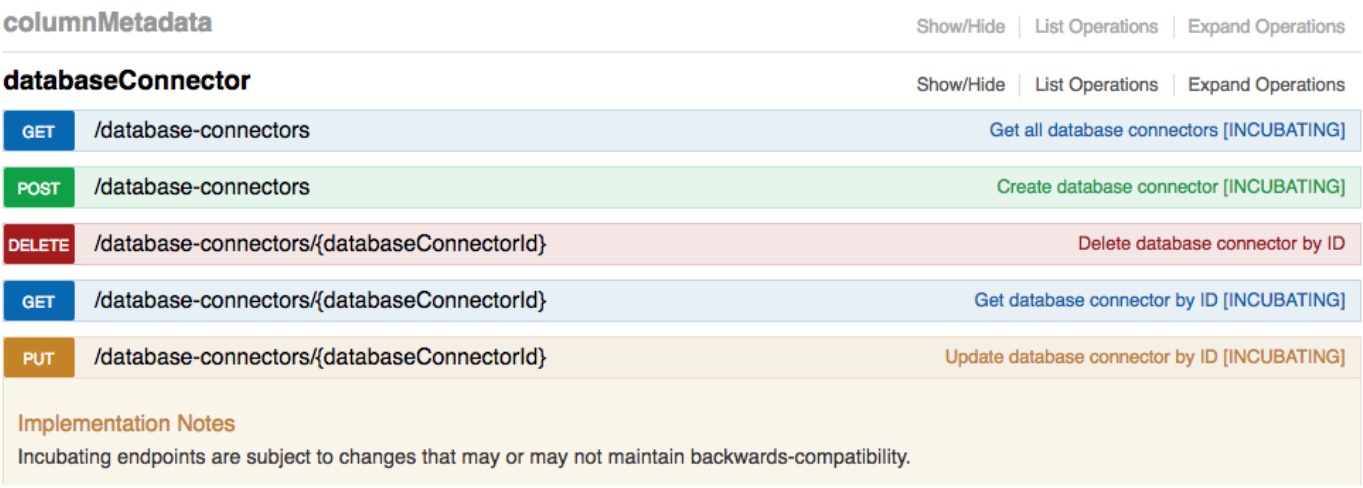
> Note that all changes to the API *(not just backwards-incompatible changes)* will be documented and distributed with future releases of the API.

Backwards-incompatible changes to the API are known to be disruptive to automation built around the API, and therefore changes to Incubating APIs will be carefully considered and minimized.

List of Incubating API Endpoints

Refer to the **The Masking API Client** to see the list of Incubating API endpoints.

All Incubating API endpoints are labeled with **INCUBATING** in their description, and they are also accompanied by an **Implementation Note** explaining the implications of an Incubating endpoint with respect to backwards-compatibility.



Related Links

- The Masking API Client
- Masking API Cookbook
- Sample Cookbook Scripts

Sample Cookbook Scripts

The following set of sample bash scripts can be used to perform some basic operations using the Masking APIs. We use the curl command to call the APIs and jq () to process the resulting JSON. jq is a 3rd party tool that is used to process JSON on the command line. More information can be found here - **https://stedolan.github.io/jq/**.

- **apiHostInfo**
- **createApplication**
- **createDatabaseConnector**
- **createDatabaseRuleset**
- **createEnvironment**
- **createInventory**
- **getSyncableObjects**
- **getSyncableObjectsExport**
- **helpers**
- **loginCredentials**

Each script is designed to do a specific task and to work out of the box once the connection info to the masking engine and the credentials are provided (see **apiHostInfo** and **loginCredentials**).

A helper library is provided to fetch objects when there are any dependencies for the current operation. When we fetch objects we simply get all objects and return the first one we find. In practice, the client will want to filter out an object by a unique characteristic (e.g.: name or id). Some very basic error handling has been provided for ease of use, namely checking for any errors in the response (check_error) and checking for an empty response if we are expecting to use the result later in the script (check_empty).

> Almost all the provided scripts rely on **apiHostInfo**, **loginCredentials** and **helpers**. Care should be taken if any of these scripts are renamed.

apiHostInfo

apiHostInfo

```
#!/bin/bash

#
# This file contains all the host information for the masking engine.
Additionally,
# this file allows configuration of SSL if desired.
#


# update host name
HOST="myMaskingEngine.com"
API_PATH="masking/api"

# To connect via SSL, set $SSL to "on" and update the port if necessary
(default 8443).
# Additionally, you must update the path to the ssl certificate.
SSL="off"
SSL_PORT="8443"
# update cert name
SSL_CERT_PATH="self-signed.cer"

if [ "$SSL" = "on" ]
then
    MASKING_ENGINE="https://$HOST:$SSL_PORT/$API_PATH"
    SSL_CERT="--cacert $SSL_CERT_PATH"
else
    MASKING_ENGINE="http://$HOST:8282/$API_PATH"
    SSL_CERT=""
fi
```

createApplication

createApplication

```
#!/bin/bash

#
# This script will login and create an application. It depends on helpers in
the helpers script as well as host and login
# information found in apiHostInfo and loginCredentials, respectively.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

echo "* creating application 'App123'..."
curl $SSL_CERT -X POST -H ''"$AUTH_HEADER"'' -H 'Content-Type:
application/json' -H 'Accept: application/json' --data @-
$MASKING_ENGINE/applications <<EOF
{
    "applicationName": "App123"
}
EOF

echo
```

createDatabaseConnector

Note: this is the only creation file that requires user modification to function. The schema can be changed based on databaseType, for more details refer to **The Masking API Client**.

createDatabaseConnector

```
#!/bin/bash

#
# This script will login and create a database connector in an environment.
It depends on helpers in the helpers
# script as well as host and login information found in apiHostInfo and
loginCredentials, respectively.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

#
# When deciding which environment to place the connector in we simply choose
the first environment found. You are
# encouraged to modify this to suit your needs. Please see get_environment_id
in helpers for more information.
#
get_environment_id

echo "* creating database connector 'connector' in environment
'$ENVIRONMENT_ID'..."
curl $SSL_CERT -X POST -H ''"$AUTH_HEADER"'' -H 'Content-Type:
application/json' -H 'Accept: application/json' --data @-
$MASKING_ENGINE/database-connectors <<EOF
{
    "connectorName": "connector",
    "databaseType": "ORACLE",
    "environmentId": $ENVIRONMENT_ID,
    "host": "myHost",
    "password": "myPassword",
    "port": 1234,
    "schemaName": "MYSCHEMA",
    "sid": "mySID",
    "username": "MYUSERNAME"
}
EOF

echo
```

createDatabaseRuleset

createDatabaseRuleset

```
#!/bin/bash

#
# This script will login and create a database ruleset for a database
connector. It depends on helpers in the helpers
# script as well as host and login information found in apiHostInfo and
loginCredentials, respectively.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

#
# When deciding which database connector we will use, we simply choose the
first database connector found. You are
# encouraged to modify this to suit your needs. Please see get_connector_id
in helpers for more information.
#
get_connector_id

echo "* creating database ruleset 'myRuleset' in db connector
'$CONNECTOR_ID'..."
curl $SSL_CERT -X POST -H ''"$AUTH_HEADER"'' -H 'Content-Type:
application/json' -H 'Accept: application/json' --data @-
$MASKING_ENGINE/database-rulesets <<EOF
{
    "rulesetName": "myRuleset",
    "databaseConnectorId": $CONNECTOR_ID
}
EOF

echo
```

createEnvironment

createEnvironment

```
#!/bin/bash

#
# This script will login and create an environment with an application. It
depends on helpers in the helpers
# script as well as host and login information found in apiHostInfo and
loginCredentials, respectively.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

#
# When deciding which application to place the environment in we simply
choose the first application found. You are
# encouraged to modify this to suit your needs. Please see get_application_id
in helpers for more information.
#
get_application_id

echo "* creating environment 'newEnv' in application '$APPLICATION_ID'..."
curl $SSL_CERT -X POST -H '"$AUTH_HEADER"' -H 'Content-Type:
application/json' -H 'Accept: application/json' --data @-
$MASKING_ENGINE/environments <<EOF
{
    "environmentName": "newEnv",
    "application": "$APPLICATION_ID",
    "purpose": "MASK"
}
EOF

echo
```

createInventory

createInventory

```
#!/bin/bash

#
# This script will login, create table metadata for a given table name and
ruleset, and then update an
# inventory (i.e. assign an algorithm and domain to a specific column of the
table). It depends on helpers
# in the helpers script as well as host and login information found in
apiHostInfo and loginCredentials, respectively.
# This script uses jq to process JSON. More information can be found here -
https://stedolan.github.io/jq/.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

#
# When deciding which connector, ruleset, and table to use we simply use the
first ones found of each. You are
# encouraged to modify this to suit your needs. Please see the respective
functions in helpers for more information.
#
get_connector_id
get_ruleset_id
get_table

echo "* creating table metadata for ruleset id '$RULESET_ID' with table
'$TABLE_NAME'..."
TABLE_METADATA_RESPONSE=$(curl $SSL_CERT -s -X POST -H ''"$AUTH_HEADER"'' -H
'Content-Type: application/json' -H 'Accept: application/json' --data @-
$MASKING_ENGINE/table-metadata <<EOF
{
    "tableName": "$TABLE_NAME",
    "rulesetId": $RULESET_ID
}
EOF)
check_error "$TABLE_METADATA_RESPONSE"
TABLE_METADATA_ID=$(echo $TABLE_METADATA_RESPONSE | jq -r '.tableMetadataId')
echo "using table metadata '$TABLE_METADATA_ID'"

get_column_metadata_id

curl $SSL_CERT -X PUT -H ''"$AUTH_HEADER"'' -H 'Content-Type:
application/json' -H 'Accept: application/json' --data @-
$MASKING_ENGINE/column-metadata/$COLUMN_METADATA_ID  <<EOF
{
```

```
    "algorithmName": "AddrLine2Lookup",
    "domainName": "ADDRESS_LINE2"
}
```

```
EOF

echo
```

getSyncableObjects

getSyncableObjects

```
#!/bin/bash

#
# This script is an "out of the box" script that goes through
# Login and GET /syncable-objects with the authentication
# token from Login
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

echo "* GET /syncable-objects from $EXPORT_ENGINE"
EXPORT_RESPONSE=$(curl $SSL_CERT -X GET -H '"$AUTH_HEADER"' -H 'Accept:
application/json' $MASKING_ENGINE/syncable-objects)
echo $EXPORT_RESPONSE
```

getSyncableObjectsExport

getSyncableObjectsExport

```
#!/bin/bash

#
# This script will log in and get all syncable objects on
# the Masking Engine and then, given a grouping command, save the
# exported document in a file and export all syncable objects
# in the indicated group
#
# Grouping command:
# algoType: -t <LOOKUP | BINARYLOOKUP | SEGMENT | TOKENIZATION | MAPPLET |
KEY>
# algoCd: -n <RegexForAlgoName>
#
# Currently the response from GET /syncable-objects is saved
# to getobj_response.json, and the grouped input for /export
# in grouped_export_list.json, and the final export response
# into export_response.json. But of course, this can script
# can be modified to save to other specified places.
#

source apiHostInfo
eval $(cat loginCredentials)
source helpers

login

echo "* GET /syncable-objects"
GETOBJ_RESPONSE=$(curl $SSL_CERT -X GET -H '"$AUTH_HEADER"' -H
'Content-Type: application/json' $MASKING_ENGINE/syncable-objects)
echo $GETOBJ_RESPONSE > "./getobj_response.json"

# Create a temporary export list file
GROUPED_EXPORT_LIST="./grouped_export_list.json"
echo "[]" > $GROUPED_EXPORT_LIST

if [[ $1 == "-t" ]]; then
   ALGO_TYPE=$2
   echo "* Filter for all syncable objects of algorithm type $ALGO_TYPE"

   jq -c '.responseList[]' getobj_response.json | while read i; do
      if [[ $(echo $i | jq '.objectType') == \"$ALGO_TYPE\" ]]; then
         # The key to getting the correct json format here was to use
         # the --argjson instead of --arg. --arg will stringify everything
         # and escape all special characters like {, ", etc.
         echo $(cat $GROUPED_EXPORT_LIST | jq --argjson obj "$i" '. |= . +
[$obj]') > $GROUPED_EXPORT_LIST
      fi
   done
elif [[ $1 == "-n" ]]; then
```

```
   ALGO_NAME_REGEX=$2
   echo "* Filter for all syncable objects where algorithmCd matches the
regex $ALGO_NAME_REGEX"

   jq -c '.responseList[]' getobj_response.json | while read i; do
      if [[ "$(echo $i | jq '.objectIdentifier.algorithmName')" =~
\"$ALGO_NAME_REGEX\" ]]; then
          echo $(cat $GROUPED_EXPORT_LIST | jq --argjson obj "$i" '. |= . +
[$obj]') > $GROUPED_EXPORT_LIST
      fi
   done
fi

echo "* Export syncable objects from $GROUPED_EXPORT_LIST"
EXPORT_RESPONSE=$(curl $SSL_CERT -X POST -H ''"$AUTH_HEADER"'' -H
'Content-Type: application/json' -H 'Accept: application/json' -d
"$(<$GROUPED_EXPORT_LIST)" $MASKING_ENGINE/export)

# Save the grouped export response into a file
echo $EXPORT_RESPONSE > export_response.json
```

```
echo '* Completed exporting. Check "export_response.json" for the export
document. This export document json object will be what you literally put in
as the input for import'
```

helpers

helpers

```
#!/bin/bash

#
# This file contains helpers for the various Masking API cookbook scripts.
# This script uses jq to process JSON. More information can be found here -
https://stedolan.github.io/jq/.
#

# Login and set the correct $AUTH_HEADER.
login() {
    echo "* logging in..."
    LOGIN_RESPONSE=$(curl -s $SSL_CERT -X POST -H 'Content-Type:
application/json' -H 'Accept: application/json' --data @-
$MASKING_ENGINE/login <<EOF
{
    "username": "$USERNAME",
    "password": "$PASSWORD"
}
EOF)
    check_error "$LOGIN_RESPONSE"
    TOKEN=$(echo $LOGIN_RESPONSE | jq -r '.Authorization')
    AUTH_HEADER="Authorization: $TOKEN"
}

# Get all applications and select the first one. Place the applicationName in
$APPLICATION_ID.
get_application_id() {
    echo "* getting all applications and selecting first one"
    APPLICATIONS_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H
'Content-Type: application/json' $MASKING_ENGINE/applications)
    check_error "$APPLICATIONS_RESPONSE"
    NUM_APPLICATIONS=$(echo $APPLICATIONS_RESPONSE | jq -r
'._pageInfo.total')
    check_empty $NUM_APPLICATIONS "found no applications to use"
    APPLICATION_ID=$(echo $APPLICATIONS_RESPONSE | jq -r
'.responseList[0].applicationName')
    echo "using application '$APPLICATION_ID'"
}

# Get all environments and select the first one. Place the environmentId in
$ENVIRONMENT_ID.
get_environment_id() {
    echo "* getting all environments and selecting first one"
    ENVIRONMENTS_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H
'Content-Type: application/json' $MASKING_ENGINE/environments)
    check_error "$ENVIRONMENTS_RESPONSE"
    NUM_ENVIRONMENTS=$(echo $ENVIRONMENTS_RESPONSE | jq -r
'._pageInfo.total')
    check_empty $NUM_ENVIRONMENTS "found no environments to use"
```

```
    ENVIRONMENT_ID=$(echo $ENVIRONMENTS_RESPONSE | jq -r
'.responseList[0].environmentId')
    echo "using environment '$ENVIRONMENT_ID'"
}

# Get all database connectors and select the first one. Place the
databaseConnectorId in $CONNECTOR_ID.
get_connector_id() {
    echo "* getting all database connectors and selecting first one"
    CONNECTORS_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H
'Content-Type: application/json' $MASKING_ENGINE/database-connectors)
    check_error "$CONNECTORS_RESPONSE"
    NUM_CONNECTORS=$(echo $CONNECTORS_RESPONSE | jq -r '._pageInfo.total')
    check_empty $NUM_CONNECTORS "found no db connectors to use"
    CONNECTOR_ID=$(echo $CONNECTORS_RESPONSE | jq -r
'.responseList[0].databaseConnectorId')
    echo "using database connector '$CONNECTOR_ID'"
}

# Get all database rulesets and select the first one. Place the
databaseRulesetId in $RULESET_ID.
get_ruleset_id() {
    echo "* getting all database rulesets and selecting first one"
    RULESETS_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H
'Content-Type: application/json' $MASKING_ENGINE/database-rulesets)
    check_error "$RULESETS_RESPONSE"
    NUM_RULESETS=$(echo $RULESETS_RESPONSE | jq -r '._pageInfo.total')
    check_empty $NUM_RULESETS "found no db rulesets to use"
    RULESET_ID=$(echo $RULESETS_RESPONSE | jq -r
'.responseList[0].databaseRulesetId')
    echo "using database ruleset '$RULESET_ID'"
}

# Get all database tables for a database connector specificed by
$CONNECTOR_ID. Select the first one and place in $TABLE_NAME.
get_table() {
    echo "* getting all tables for connector '$CONNECTOR_ID' and selecting
first one"
    TABLES_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H
'Content-Type: application/json'
$MASKING_ENGINE/database-connectors/$CONNECTOR_ID/fetch)
    check_error "$TABLES_RESPONSE"
    NUM_TABLES=$(echo $TABLES_RESPONSE | jq -r '. | length')
    check_empty $NUM_TABLES "found no tables to use"
    TABLE_NAME=$(echo $TABLES_RESPONSE | jq -r '.[0]')
    echo "using table '$TABLE_NAME'"
}

# Get all column metadata for table metadata specified by $TABLE_METADATA_ID.
Select the first one and place in $COLUMN_METADATA_ID.
get_column_metadata_id() {
    echo "* getting all column metadata belonging to table metadata
'$TABLE_METADATA_ID' and selecting the first one"
```

```
    COLUMNS_RESPONSE=$(curl -s $SSL_CERT -X GET -H ''"$AUTH_HEADER"'' -H
'Content-Type: application/json'
$MASKING_ENGINE/column-metadata?table_metadata_id=$TABLE_METADATA_ID)
    check_error "$COLUMNS_RESPONSE"
    NUM_COLUMNS=$(echo $COLUMNS_RESPONSE | jq -r '. | length')
    check_empty $NUM_COLUMNS "found no columns to use"
    COLUMN_METADATA=$(echo $COLUMNS_RESPONSE | jq -r '.responseList[0]')
    COLUMN_METADATA_ID=$(echo $COLUMN_METADATA | jq -r '.columnMetadataId')
    echo "using column '$COLUMN_METADATA_ID'"
}

# Check if $1 is equal to 0. If so print out message specified in $2 and
exit.
check_empty() {
    if [ $1 -eq 0 ]; then
        echo $2
        exit 1
    fi
}

# Check if $1 is an object and if it has an 'errorMessage' specified. If so,
print the object and exit.
check_error() {
    # jq returns a literal null so we have to check againt that...
    if [ "$(echo "$1" | jq -r 'if type=="object" then .errorMessage else
"null" end')" != 'null' ]; then
        echo $1
```

```
        exit 1
    fi
}
```

loginCredentials

Login credentials for the Masking Engine.

loginCredentials
USERNAME="myUsername" PASSWORD="myPassword"