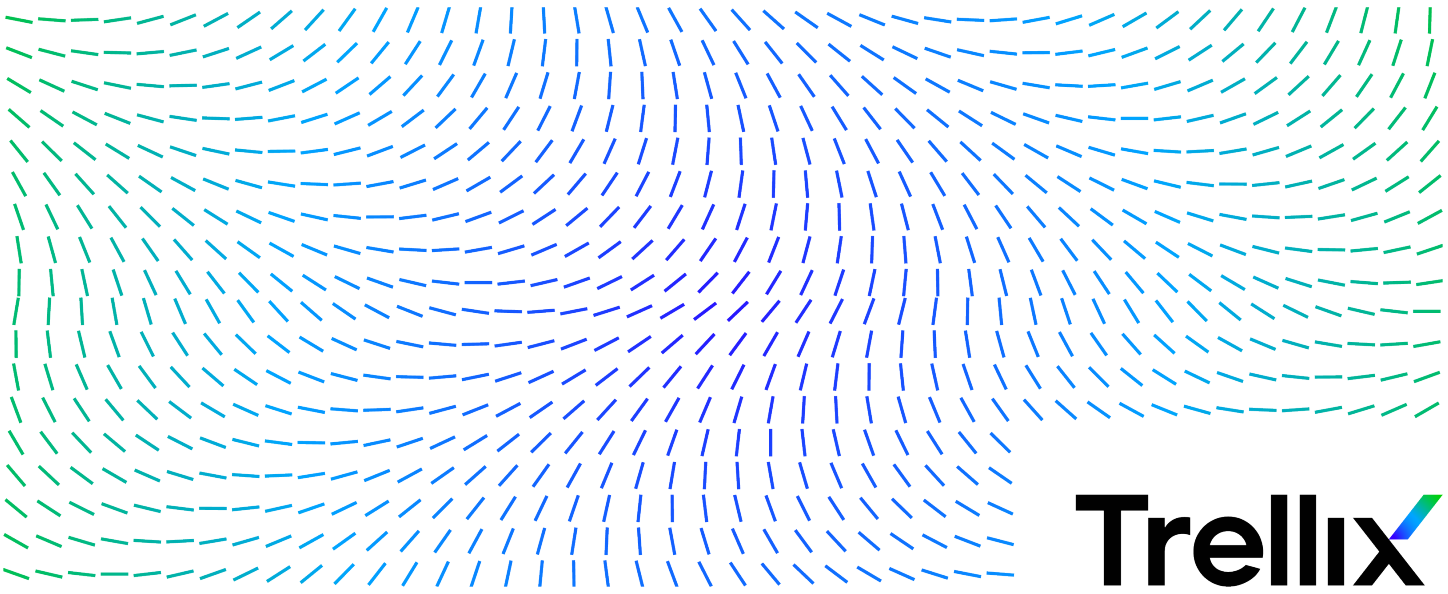
Trellix ePolicy Orchestrator - On-prem Web API Scripting Reference Guide



Trellix

Contents

Overview

Trellix® ePolicy Orchestrator® provides a web application programming interface (API) that allows you to script and automate common management activities. For example, you can automate user and System Tree maintenance and data import and export.

This guide explains what the ePolicy Orchestrator web API is, how to use it, and walks you through a few examples using a Python client. It also includes a detailed look at some key commands and an extensive description of the query system.

Web API basics

You can use the ePolicy Orchestrator web API commands, with the command-line, to automate ePolicy Orchestrator configuration using scripts instead of using the user interface.

This section uses examples of the cURL command line tool for transferring data with URL syntax. Each command is designed to work without user interaction. The cURL executable is free and open software that runs under a wide variety of operating systems.

Each cURL example includes standard parameters followed by the actual ePolicy Orchestrator web API URL that executes a command on your Trellix ePO server. These parameters should help you to understand what the command does, although in practice you should implement tighter security when it comes to trusting the site's certificate.

Note

The ePolicy Orchestrator web API supports using other command-line tools, for example **wget** (part of the GNU Project, © 2009, Free Software Foundation, Inc.), to retrieve data from your Trellix ePO server.

This command, for example, shows the cURL syntax and the URL to illustrate the core capabilities of the web API.

```
> curl -k -u ga:ga https://localhost:8443/remote/core.help
```

This table shows the parameters used with the **curl** command example.

Parameter	Description
-k	Allows cURL to perform "insecure" SSL connections and transfers.
-u	Specifies the user name and password to use for server authentication. If you enter just the user

Parameter	Description
	name (without a colon) cURL prompts you for a password.
ga	User name, "ga", global administrator used in this document's examples. **Note:** You can use special characters in your user names, but make sure you follow your shell's quoting and escaping rules.
ga	Password, "ga", used in this document's examples. **Note:** You can use special characters in your passwords, but make sure you follow your shell's quoting and escaping rules.
localhost :	ePolicy Orchestrator server, identified as "localhost", in this document's examples.
8443	Destination port, identified as "8443" (the default), in this document's examples.

In the examples in this document, the Trellix ePO server and destination port are identified as "localhost" and "8443" (the default). You need to replace these entries with the server name and port number of your own installation.

Note

Web API commands follow all role-based permissions as enforced through the Trellix ePO server graphical interface.

The web API is used primarily for two purposes:

- Performing simple tasks without using the user interface
- Scripting sequences of tasks

Scripts using the web API can be run from any computer that can connect to the Trellix ePO server. For security reasons, these commands should not be run on the same computer as the Trellix ePO server itself.

General syntax

The general syntax for a command sent via HTTPS is:

```
https://<server>:<port>/remote/<command>?<arg1>=<val1>&<arg2>=<val2>
```

Additional arguments can be specified as needed.
Some commands require input in other formats, such as importing a file. For example, importing an XML file containing permission sets looks like this:

```
> curl -k -u ga:ga "https://localhost:8443/remote/core.importPermissionSets" -F file=@permissionSets.xml
```

Output options

By default, commands return output in a human-readable format. When scripting, however, you usually want commands to return data in a more machine-readable format. This format is controlled with the **:output** parameter.

```
https://localhost:8443/remote/core.help?:output=json
```

This example returns data in JavaScript Object Notation (JSON) format. Other options include **verbose** (default), **terse**, and **xml**. These arguments must be supplied as all lowercase text. In addition, the parameters shown in this table are available.

Output format parameters

Parameter	Description	Values
:output	Specifies the output format.	**verbose** (default) **terse**, **xml**, **json**
:locale	Specifies the output locale.	Defaults to server's locale. Example values include **en**, **de**, **cn-zh**.
:validation	Specifies the validation level on the command. Strict validation throws errors when an argument is missing, loose validation ignores missing arguments.	**strict** (default), **loose**

Discover available commands through URLs

Newly installed **Trellix ePO - On-prem** extensions provide more web API commands. Learn which commands are available to you.

Use the **core.help** command to learn which commands you can access and the details of specific commands. When used without any arguments, **core.help** provides a list of available commands.

```
> curl -k -u ga:ga https://localhost:8443/remote/core.help
```

Note

The exact list of commands displayed depends on your permissions and the extensions installed.

This command returns a list that looks similar to this example.

```
OK:
ComputerMgmt.createCustomInstallPackageCmd windowsPackage deployPath [ahId] [fallBackAhId] [useCred]
[domain] [username] [password] [rememberDomainCredentials] -
ComputerMgmt.create.Custom.Install.Package.Cmd.short-desc
agentmgmt.listAgentHandlers - List all Agent Handlers
clienttask.export [productId] [fileName] - Exports client tasks
clienttask.find [searchText] - Finds client tasks
clienttask.importClientTask importFileName - Imports client tasks from an XML file.
clienttask.run names productId taskId [retryAttempts] [retryIntervalInSeconds] [abortAfterMinutes]
[useAllAgentHandlers] [stopAfterMinutes] [randomizationInterval] - Runs the client task on a supplied list
of systems
clienttask.syncShared - Shares client tasks with participating registered servers
commonevent.purgeEvents queryId [unit] [purgeType] - Deletes threat events based on age or a queryId. The
query must be table based.
commonevent.purgeProductEvents queryId [unit] [purgeType] - Purge Client Events by Query ID or age.
console.cert.updatecrl console.updateCRL crlFile - cert.update.crl.help.oneline
core.addPermSetsForUser userName permSetName - Adds permission set(s) to specified user
core.addUser userName password [fullName=<>] [email=<>] [phoneNumber=<>] [notes=<>] [disabled=<>]
[admin=<>] - Adds a user to the system
core.executeQuery queryId [database=<>] - Executes a SQUID query and returns the results
.
[information deleted]
.
system.report names - Reports the systems in the System Tree
system.runTagCriteria tagID [resetTaggedSystems] - The Run Tag Criteria action evaluates every
managed system against the tag's criteria. system.setUserProperties names [description] [customField1]
[customField2] [customField3] [customField4] - Sets user properties on the given system
system.transfer names epoServer - Transfers systems to a different ePO server
system.wakeupAgent names [fullProps] [superAgent] [randomMinutes] [forceFullPolicyUpdate] [useAllHandlers]
[retryIntervalSeconds] [attempts] [abortAfterMinutes] [includeSubgroups] - Wakes up the agent on a
supplied list of systems
tasklog.listMessages taskLogId - Lists the messages for the specified task log entry
tasklog.listSubtasks taskLogId - Lists subtasks of a specified task log entry
tasklog.listTaskHistory [taskName] [taskSource] [maxRows] [age] [unit] - Lists
task log entries, optionally filtered by task name, task ID or task source
tasklog.listTaskSources - Lists the task sources
tasklog.purge [age] [unit] - Purges the Server Task Log beyond a given age and time unit
```

The Help output displays the:

- Prefix, for example "core".
- Command name, for example "help".
- Required arguments and optional arguments. Optional arguments are enclosed in square brackets ("[" and "]").

Note

Arguments followed by `=<>` require the specific argument name and a value. For example, if the command help shows [email=<>] you must provide the argument name and the value, as in **email=joe_test@trellix.com**.

- Brief description of the command.

This extended command example is used to request more detailed information about a specific command.

```
> curl -k -u ga:ga  https://localhost:8443/remote/core.help?command=core.listQueries
```

This command displays:

```
OK:
core.listQueries
Displays all queries that the user is permitted to see. Returns the list of
queries or throws on error.
Requires permission to use queries.
```

Example task using the web API

You can accomplish many simple tasks using the **Trellix ePO - On-prem** web API. This complete example demonstrates the suggested steps to complete a task.

This example allows you to use the web API to assign a policy to a group.

Task

1. **Use this Help request to find out what the policy.assignToGroup command requires.**

   ```
   > curl -k -u ga:ga "https://localhost:8443/remote/core.help?command=policy.assignToGroup"
   ```

   This command displays:

   ```
   OK:
   policy.assignToGroup groupId productId objectId [resetInheritance]
   Assigns policy to the specified group or resets group's inheritance for the specified policy
   Requires permission to at least one group in the System Tree and edit permission for at least one
   product
   Parameters:
    groupId (param 1) - Group ID as returned by system.findGroups
    productId (param 2) - Product ID as returned by policy.find
    objectId (param 3) - Object ID as returned by policy.find
    resetInheritance (param 4) - If true resets the inheritance for the specified policy on the given
   group. Defaults to false.
   ```

   This Help request shows that there are three required arguments:

   - Group ID
   - ID for the product you want to assign
   - Object ID of the policy to assign.

Note

You could also reset the inheritance, but that argument is not required, and is not used it in this example.

2. **Use the system.findGroups command to find the group ID.**

```
> curl -k -u ga:ga "https://localhost:8443/remote/system.findGroups?searchText=My%20Group"
```

That command returns a result similar to this:

```
OK:
groupId: 2
groupPath: My Organization

groupId: 4
groupPath: My Organization\My Group
```

3. **Use this policy.find command to find the product ID and policy object ID:**

```
> curl -k -u ga:ga "https://localhost:8443/remote/policy.find?searchText=quarantine"
```

That command returns two results:

```
OK:
featureId: VIRUSCAN8800
featureName: VirusScan Enterprise
objectId: 131
objectName: McAfee Default
objectNotes:
productId: VIRUSCAN8800
productName: VirusScan Enterprise 8.8.0
typeId: 67
typeName: Quarantine Manager Policies

featureId: VIRUSCAN8800
featureName: VirusScan Enterprise
objectId: 142
objectName: My Default
objectNotes:
productId: VIRUSCAN8800
productName: VirusScan Enterprise 8.8.0
typeId: 67
typeName: Quarantine Manager Policies
```

4. **Choose one of policy.find results. This example uses the My Default policy and you have the necessary information required to assign a policy to a group with this information:**
    - Group ID — 4
    - Product ID — VIRUSCAN8800
    - Policy object ID — 142
5. **Use the previous information and this policy.assignToGroup command to assign a policy to a group:**

```
> curl -k -u ga:ga https://localhost:8443/remote/policy.assignToGroup -d
"groupId=4&productId=VIRUSCAN8800&objectId=142"
```

This returns:

```
OK:
True
```

The policy is assigned.

Results

In general, the web API commands return three kinds of results:

1. **True** or **False** results indicating the command's success or failure
2. Data returned from the command
3. Multiple data items and which items succeeded or failed

Python client basics

Trellix provides a software download that includes Python version 2.7 and the Python Remote Client software, also known as pyclient. The supplied Python Remote Client software simplifies communication with, and exploration of, your **Trellix ePO - On-prem** web API.

Using **Trellix ePO - On-prem** web API commands with a scripting language, like Python, provides flexibility. The scripting language allows you to use:

- the output of one command as the input to another
- conditions to perform actions based on script input or command output.

Note

If you connect to **Trellix ePO - On-prem** 5.x using the **Trellix ePO - On-prem** 4.6.x Python client, you might experience intermittent connection errors. **Trellix ePO - On-prem** 5.x users should download the **Trellix ePO - On-prem** 5.x Python client available from the Software Manager.

Python client software requirements

The Python client requires Python version 2.6 or 2.7. The source code is included in a file named **mcafee.py**. For your Python client scripts to function properly, this **mcafee.py** file must be placed in the same folder as your scripts, or in the Python module search path.

Using the client

The Python client converts data returned from all commands into Python objects for easy processing. The Python client can be used in two ways: imported into a **.py** script run at the command line, or in Python interactive mode. In either case, follow the script requirements. See *Web API Python script requirements* for details.

The source code to the client, which is included, can be used for educational purposes, porting to other languages, or expanding the Python client's capabilities.

Notes on parameters

Some commands, such as **system.clearTag**, can take a comma-delimited list of values as a parameter. If you want to embed spaces in this list, enclose the entire list in quotes.

```
mc.system.clearTag("System1, System2, System3","oldTag")
```

Any parameter requiring a file name uses the **file:///c:/path/to/file** format used in URLs.

Web API Python script requirements

Before a Python script can execute any Python Remote Client commands, it must import the **Trellix** Python client library and authenticate with the **Trellix ePO - On-prem** server.

Python client script requirements

Every Python script you create must have these two lines of code at the beginning:

```
# McAfee Python script requirements
import mcafee
mc = mcafee.client("localhost","8443","username","password")
```

In this example:

- **import mcafee** — Imports the **Trellix** Python client library into your script. See *Import the McAfee Python client library*.
- **mc = mcafee.client("localhost","8443","username","password")** — Authenticates with the server. See *Script ePO server authentication*

After you complete these two tasks, the remainder of the script functions as expected.

Import the Trellix Python client library

Import the **Trellix** Python client library into any script you create to communicate with your **Trellix** server.

Task

1. **Make sure the mcafee.py file is stored in either the same folder as your script, or in the Python module search path.**
2. **Import the client code into your script with this command:**

   ```
   import mcafee
   ```

Results

This command imports the **Trellix** server client code, which includes a method that takes connection information, establishes a session with the indicated server, and returns a session object.

Script Trellix ePO server authentication

Before any commands can be sent to a Trellix ePO server, authenticate with the server and store the returned session object.

To allow your script to communicate with the server, use the **mcafee.client** command which takes between four and six of these string parameters.

mcafee.client parameters

Parameter	Description
server	Contains the name of the server. Do not include the **https://** prefix. If your server name is **https://myserver**, type **myserver** in this parameter.
port	Contains the port the server uses. This is typically **8443** for HTTPS connections, but could be different.
username	Contains the user name for authentication.
password	Contains the password for authentication. **Important:** The password used in your script is stored in plain text. Secure your scripts appropriately. Alternatively, either prompt your user for a password, or store the password encrypted. The web API does not support certificate authentication.
protocol	[optional] Contains the protocol. The default is **HTTPS**.
outputtype	[optional] Contains the output type returned from commands. This value defaults to **json** but **verbose**, **terse**, and **xml** are other valid values.

Authentication examples

If you normally log on to your **ePOserver** on port 8443 with the user name **adminfred** and password **mydOgsname37**, your Python command to log on is:

```
mc = mcafee.client("ePOserver","8443","adminfred","mydOgsname37")
```

The **mc** variable stores the session information used for all later commands in that script. For example, if the next thing you want to do in the script is list all currently running server tasks, the command is:

```
mc.scheduler.listRunningServerTasks()
```

Discover available commands in Python

Trellix extensions can add commands to the **Trellix** Python client library. The Python client provides a way to determine which commands are available on your server.

Before you begin

For these commands to work, the script must meet the script conditions described in *Web API Python script requirements*.

Task

1. **Use the Python dir() command to find the list of modules available.**

   ```
   >>> dir(mc)
   ```

   This command returns a Python list similar to this:

   ```
   ['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattr__', '__getattribute__',
    '__hash__', '__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
   '__setattr__',
   '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_invoker', 'core', 'help', 'scheduler',
   'tasklog']
   ```

   There are a number of attributes in the example, but what you are looking for are the names at the end of the list without leading underscores. In this example, these are **core**, **help**, **scheduler**, and **tasklog**. These objects contain commands you can execute.

   Note

   The other attribute names with leading underscores are internal to either the client or Python itself.

2. **Use the dir() command, passing the module as a parameter, to find the list of commands in a module.**

   ```
   >>> dir(mc.scheduler)
   ```

   This command returns a list of attributes and commands in the scheduler module similar to this:

   ```
   ['__doc__', '__getattr__', '__init__', '__module__', '_invoker', '_module',
   'cancelServerTask', 'getServerTask', 'listAllServerTasks', 'listRunningServerTasks',
   'runServerTask', 'setServerTaskStatus']
   ```

As in the previous step, look for names without leading underscores. In this case, there are six commands: **cancelServerTask**, **getServerTask**, **listAllServerTasks**, **listRunningServerTasks**, **runServerTask**, and **setServerTaskStatus**.

3. **Once you have found a command you want to use, type help() and pass the command as a parameter.**

```
>>> mc.help('scheduler.listRunningServerTasks')
```

This command returns a description like this:

```
scheduler.listRunningServerTasks
Get the list of all running server tasks. Returns the list of tasks or throws an error.
Requires permission to view server tasks.
```

This description lists the command name and parameters (in this case there aren't any), a description of what it does, and the permission required to execute it.

Results

Perform similar steps for any command you want to run. These steps help determine all capabilities available for your scripts.

Additional documentation included with the web API

Several HTML files are included with the web API package that provide more information.

All files are included in the **pyclient/Python26/mcafee-docs** folder contained in the package.

Documentation files in the web API package

File	Topic
FAQ.html	Contains common questions asked when using the Python client.
getting_started.html	Contains a brief tutorial for writing Python scripts.
implementation.html	Contains Python client implementation details including response formats.
setup.html	Contains instructions for installing the Python distribution, or for customizing an existing installation.

Key commands

Some commands are more commonly used than others. To create scripts quickly, we recommend that you familiarize yourself with the syntax for these common commands.

These tables list commonly used commands with their syntax and description. Each table covers a different functional area.

Note

Specify arguments followed by **=<>** by name. For example, the argument **fullName=** must be included in this command, **core.addUser("ga", "ga", fullName="Joe Tester")**

Commands for searching, querying, and listing

Command	Syntax	Description

core.executeQuery	`core.executeQuery queryId [database=<>]`
	`core.executeQuery target=<> [select=<>] [where=<>] [order=<>] [group=<>]`
	`[database=<>] [depth=<>] [joinTables=<>]`
		Executes a query and returns the results as a list of objects.

core.help	`core.help [command] [prefix=<>]`	Lists all registered commands, and displays help strings.
core.listDatabases	`core.listDatabases`	Returns all databases the user is permitted to see as a list of objects.
core.listQueries	`core.listQueries`	Returns all queries the user is permitted to see as a list of objects.
core.listTables	`core.listTables [table=<>]`	Returns all database tables the user is permitted to see as a list of objects.

Command	Syntax	Description
policy.find	`policy.find searchText`	Finds all policies that the user is permitted to see that match the given search text.
repository.find	`repository.find searchText`	Finds all repositories that the user is permitted to see that match the given search text.
system.find	`system.find searchText`	Finds systems in the System Tree.

Commands for creating, importing, and exporting

Command	Syntax	Description

core.addUser	`core.addUser userName password [fullName=<>] [email=<>] [phoneNumber=<>] [notes=<>] [disabled=<>] [admin=<>]`
	`core.addUser userName=<> windowsUserName=<> windowsDomain=<> [fullName=<>] [email=<>] [phoneNumber=<>] [notes=<>] [disabled=<>] [admin=<>]`
	`core.addUser userName=<> subjectDN=<> [fullName=<>] [email=<>] [phoneNumber=<>] [notes=<>] [disabled=<>] [admin=<>]`
		Adds a user to the system. Authentication parameters are mutually exclusive: either password, windowsUserName/ windowsDomain, or subjectDN can be specified.

core.importPermissionSets	`core.importPermissionSets file [overwrite]`	Imports permission sets from a file.
core.exportPermissionSets	`core.exportPermissionSets`	Exports all permission sets as an XML string.

Command	Syntax	Description
system.importSystem	system.importSystem fileName branchNodeID [allowDuplicates] [uninstallRemoved] [pushAgent] [pushAgentForceInstall] [pushAgentSkipIfInstalled] [pushAgentSuppressUI] [pushAgentInstallPath] [pushAgentPackagePath] [pushAgentDomainName] [pushAgentUserName] [pushAgentPassword] [deleteIfRemoved] [createNewInLostAndFound] [flattenTreeStructure	Imports systems from a text file or a supplied comma-separated list.
repository.checkInPackage	repository.checkInPackage packageLocation branch [option] [moveToPrevious] [allowUnsignedPackages]	Checks package into the master repository.

Commands for editing, assigning, and moving

Command	Syntax	Description
core.updateUser	core.updateUser userName [password=<>] [windowsUserName=<>] [windowsDomain=<>] [subjectDN=<>] [newUserName=<>] [fullName=<>] [email=<>] [phoneNumber=<>] [notes=<>] [disabled=<>] [admin=<>]	Updates an existing user. Authentication parameters are mutually exclusive: either password, windowsUserName/windowsDomain, or subjectDN can be specified.
core.addPermSetsForUser	core.addPermSetsForUser userName permSetName	Adds the given permission set to the specified user.
system.applyTag	system.applyTag names tagName	Assigns the given tag to a supplied list of systems.

Command	Syntax	Description
system.setUserProperties	`system.setUserProperties name [description] [customField1] [customField2] [customField3] [customField4]`	Sets user properties on the given system.
system.deployAgent	`system.deployAgent names username [password] [agentPackage] [skipIfInstalled] [suppressUI] [forceInstall] [installPath] [domain] [useAllHandlers] [primaryAgentHandler] [retryIntervalSeconds] [attempts] [abortAfterMinutes] [includeSubgroups] [useSsh] [inputSource]`	Deploys an agent to the given list of systems.
policy.assignToSystem	`policy.assignToSystem names productId typeId objectId [resetInheritance]`	Assigns the policy to a supplied list of systems.

Commands for running and aborting

Command	Syntax	Description
clienttask.run	`clienttask.run names productId taskID [retryAttempts] [retryIntervalInSeconds] [abortAfterMinutes] [useAllAgentHandlers] [stopAfterMinutes] [randomizationInterval]`	Runs the client task on a supplied list of systems.
scheduler.cancelServerTask	`scheduler.cancelServerTask taskLogId`	Ends a currently running task.
scheduler.runServerTask	`scheduler.runServerTask taskName`	Runs the specified server task.

Command	Syntax	Description
system.wakeupAgent	`system.wakeupAgent names [fullProps] [superAgent] [randomMinutes] [forceFullPolicyUpdate] [useAllHandlers] [retryIntervalSeconds] [attempts] [abortAfterMinutes] [includeSubgroups]`	Wakes up the agent on a supplied list of systems.
repository.pull	`repository.pull sourceRepository targetBranch [moveToPrevious] [productList]`	Pulls packages from the source repository.

Commands for deleting and purging

Command	Syntax	Description
commonevent.purgeEvents	`commonevent.purgeEvents queryId [unit]`	Deletes threat events based on age or a queryId. The query must be table-based.
core.purgeAuditLog	`core.purgeAuditLog [age] [unit]`	Purges the audit log by age.
system.delete	`system.delete names [uninstall]`	Deletes systems from a Trellix ePO server by name or ID.
tasklog.purge	`tasklog.purge [age] [unit]`	Purges the task log by age. Defaults to purging all entries.

Example Python scripts

Creating some example Python scripts can show you the various ways scripting can help keep your **Trellix ePO - On-prem** servers maintained and up-to-date.

Go through these scripts, taken from differing categories of tasks, in this order.

1. Tag systems from a list.
2. Take a tag as input and send a **Trellix Agent** wake-up call to all systems with that tag.
3. Apply automation to cleaning up disabled **Trellix ePO - On-prem** users.
4. Import computers from an external file and adding them to the System Tree.
5. Export policies and client tasks from one **Trellix ePO - On-prem** server and import them into another.
6. Organize your System Tree.

These scripts slowly build on each other and the concepts explained.

Note

Later scripts don't repeat concepts explained in earlier scripts.

Example 1: Tag systems from a list

You can learn the basics of writing Python scripts using this example and the **Trellix ePO - On-prem** web API. After showing you the entire script, the individual parts of the script are described in detail.

This script assumes:

- You have a text file, `myfile.txt`, with a list of managed systems with one system per line and the systems are listed by name or IP address
- You want to apply a specific tag, in this example `myTag`, to every system in that list
- The tag named `myTag` has already been created.

Note

A more robust script would manage those assumptions, but that would complicate the example.

```
#Example 1
import mcafee
mc = mcafee.client('localhost','8443','ga','ga')

file = open('C:/path/to/myfile.txt', 'r')
for line in file:
        mc.system.applyTag(line.rstrip(‘\n’), 'myTag')
```

Examine the various script sections in detail

This line in the script imports the provided **Trellix** Python module (**mcafee.py**), kept in the same directory as your script.

```
import mcafee
```

The next line creates the connection to the **Trellix ePO - On-prem** server by specifying the server name, connection port, user name, and password, in that order.

Note

This initialization function can take up to two more parameters that specify the protocol and the presentation of the output.

```
mc = mcafee.client('localhost','8443','ga','ga')
```

The full parameter list is:

mc = mcafee.client('yourservername','port','username','password', 'protocol', 'outputtype')

- The **protocol** defaults to **https** on your **Trellix ePO - On-prem** server.
- The **outputtype** determines the format of output from commands as described in *Web API Basics*.

With those two lines, you have established a connection to your server.

This line creates your file handle in read-only mode.

```
file = open('C:/path/to/myfile.txt', 'r')
```

These lines iterate through the file, run the command **system.applyTag** to each system in the file, while stripping out the newline (**'\n'**):

```
 for line in file:
       mc.system.applyTag(line.rstrip('\n'), 'myTag')
```

After finishing the loop, each system in the file has the tag **myTag** applied.

Example 2: Automate repetitive tasks on managed systems

You can take a tag name for input and send a **Trellix Agent** wake-up call to all systems having that tag with this example script.

```
#Example 2
import mcafee
mc = mcafee.client('localhost','8443','ga','ga')

input = 'myTag'
```

```
systems = mc.system.find(input)
for system in systems:
         id = system['EPOComputerProperties.ParentID']
         result = mc.system.wakeupAgent(id)
```

This script takes a single argument as input, in this example **myTag**. It uses the **system.find** command to search for all computers with that tag.

Note

Your input could be something other than a tag. For example, the **system.find** command description displays "Find Systems in the **Trellix ePO - On-prem** tree by name, IP address, MAC address, user name, AgentGUID, or tag."

This script uses the **EPOComputerProperties.ParentID** property to send to the **system.wakeupAgent** command, but since that command also takes a name, that line could have been written as:

```
id = system['EPOComputerProperties.ComputerName']
```

Note

You could build a comma-delimited string and send the list to the command directly since **system.wakeupAgent** also accepts a list of names or IDs as input.

Example 3: Automate user management

You use this script to search through all **Trellix ePO - On-prem** users and deletes any marked as "disabled" if they are not administrators.

Also, this script provides examples of more detailed handling of actions, plus an exception when the user deletion is unsuccessful.

```
#Example 3
import mcafee
mc = mcafee.client('localhost','8443','ga','ga')

users = mc.core.listUsers();
for user in users:
   if user['disabled'] == True and user['admin'] == False:
      name = user['UserName']
      try:
          mc.core.removeUser(name)
      except Exception, e:
         print 'Error ' + str(e)
```

To see the available properties, examine the output from the **core.listUsers** command. This script uses the **disabled**, **admin**, and **UserName** properties to find and remove specific users.

Note

The **core.removeUser** command requires administrator rights to execute. Required permissions for each command are listed in its detailed help.

The **core.listUsers** command returns different values for the **authType**. This table lists the human-readable formats, used in the user interface, and the machine-readable formats.

Human readable	Machine readable
MFS authentication	pwd
Certificate Based Authentication	cert
Windows authentication	ntlm

Example 4: Import computers from external sources

You can use this script to take a comma-delimited file containing system information, and import those systems into a specified **Trellix** server System Tree group.

This script assumes that there are these two source files:

- **myfile.txt** — Contains the System Tree group where the systems are added.
- **systemsToAdd.txt** — Contains one system per line with comma-delimited properties in this order: MAC address, IP address, system name, and domain name.

```
#Example 4
import mcafee, sys
mc = mcafee.client('localhost','8443','ga','ga','https','json')

file = open('C:/myfile.txt', 'r')
for line in file:
  #determine the ID of the group to add it to
  groups = mc.system.findGroups(line.rstrip('\n'))
  groupId = -1
  for group in groups:
    if group['groupPath'] == 'My Organization\\' + line.rstrip('\n'):
    groupId = group['groupId']

    if groupId == -1:
      error = 'Error finding the specified group.'
      sys.exit(error)

#now that we have the group id, pull in the systems from file
sourceId = "12"
sourceType = "CLI"
file = open('C:/systemsToAdd.txt', 'r')
for line in file:
  sysProps = line.rstrip('\n').split(',')
  # Contains line break at "\"
```

```
systemId = mc.detectedsystem.add(sourceId,sourceType,sysProps[0],sysProps[1],dnsName= \
sysProps[2],do main=sysProps[3])
mc.detectedsystem.addToTree(str(systemId),str(groupId))
```

This example first determines where to add the systems using the **system.findGroups()** command searches for the group by name and, using that name, obtains the group ID.

The systems are added to the group with the **detectedsystem.add** command. This command has these optional parameters:

```
detectedsystem.add sourceID sourceType MAC IPAddress [IPSubnet][IPSubnetMask][dnsName] [OSPlatform]
[OSFamily] [OSVersion]
[domain] [netbiosName][netbiosComment] [users] [agentGUID] [detectedTime] [externalID]
```

You can add optional parameters in order (using any values for parameters you aren't providing). These optional parameters are, for example:

```
mc.detectedsystem.add(sourceId, sourceType,sysProps[0], sysProps[1], "0.0.0.0", "0.0.0.0", sysProps[2],
'', '', '', sysProps[3])
```

You can also assign parameters by name with the **dnsName** and **domainName** parameters in this script.

Note

The parameters **sourceID** and **sourceType** are arbitrary values defined when you add the systems. These parameters are stored in the database to record what source detected, or added, any given system.

The return value of this command is the ID of the newly added detected system. You use this ID as input to this command () which adds detected systems to the System Tree.

```
detectedsystem.addToTree UIDs branchNodeID [allowDuplicates] [dirSort]
```

By default, the system is not added if it's a duplicate and it is not automatically sorted, but you can override these defaults if you want. This script accepts the defaults and, using the newly obtained detected system ID and the group ID, moves this system into the System Tree.

Example 5: Import and export data

You can export data from an **Trellix ePO - On-prem** server and import that data to another **Trellix ePO - On-prem** server with a script.

Importing and exporting common settings for permissions, policies, or other **Trellix ePO - On-prem** objects is useful when performing:

- Server migrations

- Setting up a secondary **Trellix ePO - On-prem** server
- Creating a test environment.

This script exports client tasks and policies for a given product, then imports them to a second server.

```
#Example 5
import mcafee
mc = mcafee.client('localhost','8443','ga','ga','https','json')

# Find the product id
productId = None
policies = mc.policy.find('McAfee Agent')
for policy in policies:
        productId = policy['productId']

if productId == None:
        error = 'Error finding the product id.'
        sys.exit(error)

tasks = mc.clienttask.export(productId=productId)
file = open('tasks.xml', 'w')
print >>file, tasks
file.close()

policies = mc.policy.export(productId=productId)
file = open('policies.xml', 'w')
print >>file, policies
file.close()

# Import these into another server:
mc2 = mcafee.client('anotherEpoServer','8443','ga','ga','https','json')
mc2.clienttask.importClientTask(uploadFile='file:///tasks.xml')
mc2.policy.importPolicy(file='file:///policies.xml')
```

The script retrieves the product ID by searching for a policy containing the string '**McAfee Agent**'. Using that product ID, you can export all client tasks and policies for that product.

To import the tasks and policies, create a connection to a second **Trellix ePO - On-prem** server (**mc2**) and run the corresponding import commands.

Example 6: Automate maintenance of the System Tree

You can use this script to reapply System Tree sorting rules to any systems found in the **Lost&Found** System Tree group.

```
#Example 6
import mcafee
mc = mcafee.client('localhost','8443','ga','ga','https','json')

#first, as before, get the id of the Lost&Found group
groups = mc.system.findGroups('Lost&Found')
groupId = -1
for group in groups:
  if group['groupPath'] == 'My Organization\\Lost&Found':
    groupId = group['groupId']

  if groupId == -1:
     error = 'Error finding the specified group.'
```

```
      sys.exit(error)

#find all systems for this group
systems = mc.epogroup.findSystems(str(groupId),'true')
for system in systems:
   id = system['EPOComputerProperties.ParentID']
    mc.system.resort(str(id))
```

This example is fairly straightforward given the previous example scripts. This example introduces a new command, **epogroup.findSystems**, which finds all systems in a given group. The last parameter, **'true'** determines whether to search all subgroups. In this example, this parameter is set to **true**. The command iterates through all systems found under **Lost&Found** and any of its subgroups, then reapplies the sorting rules to those systems.

Remote queries

ePolicy Orchestrator remote commands allow you to query your database remotely using the web API. These commands allow you to execute persistent queries, which exist in the ePolicy Orchestrator database, as well as dynamic user-defined ad-hoc queries.

Persistent queries

A persistent query is accessible using the **Queries and Reports** page in the ePolicy Orchestrator user interface.

Persistent queries include both pre-installed queries and queries you create. You must know the query ID to remotely execute a persistent query.

Query examples

In all of these query examples, the first two lines of the example contain the URL and Python forms of the command. For example, the examples of the **core.listQueries** command appear as:

```
URL: https://servername:port/remote/core.listQueries
Python: mc.core.listQueries();
```

The URL example could be typed directly into your browser address bar as:

```
https://localhost:8443/remote/core.listQueries
```

Or, the URL example could be used with a cURL command as:

```
> curl -k -u ga:ga https://localhost:8443/remote/core.listQueries
```

The Python example could be used as:

```
import mcafee
mc = mcafee.client('localhost','8443','ga','ga')
mc.core.listQueries();
```

Find available queries

You must find the queries available and the query IDs before you can run any remote queries.

Use the **core.listQueries** command to find the ID of any persistent query that you can access.

```
URL: https://servername:port/remote/core.listQueries
Python: mc.core.listQueries();

OK:
```

```
Id: 1
Name: Effective permissions for users
Description: Shows all the permissions for each user
Criteria: ( where ( ne EntitlementView.RoleUri "%%NOEPOROLES%%" ) )
Group Name: Permission Queries
Owner: ga
Database Type:
Target: EntitlementView
Created by: ga
Created on: 10/25/10 8:40:33 AM PDT
Modified by: ga
Modified on: 10/25/10 8:40:33 AM PDT

Id: 2
Name: Permission set details
Description: Shows the permissions associated with each permission set
```

Remotely execute the query

Run the query using the **core.executeQuery** command with the **queryId** parameter once you know the query ID. This example uses "5" as the query ID

```
URL: https://servername:port/remote/core.executeQuery?queryId=5
Python: mc.core.executeQuery('5');

OK:
User Name: ga
Action: Create Response
Success: true
Start Time: 10/26/10 9:00:24 AM PDT

User Name: ga
Action: Create Response
Success: true
Start Time: 10/26/10 9:00:24 AM PDT
```

Ad-hoc queries

Ad-hoc queries are performed entirely remotely and do not rely on a query stored in an ePolicy Orchestrator database.

In an ad-hoc query, you specify the target of the query and up to four of the parameters in this table.

Ad-hoc query parameters

Clause	Description	Behavior if omitted
select	Controls which columns from the target table (and any joined tables) the query returns.	All columns in the target table are returned.
condition	The **condition** parameter filters results. Only records in the	All rows are returned.

Clause	Description	Behavior if omitted
	database that satisfy the filtering clause are returned.	
group	Controls grouping of the returned data.	The returned data is not grouped.
order	Controls the order of the returned data (either ascending or descending by columns).	The returned rows are ordered according to the natural order of the database.

You can use the **core.listTables**, **core.listDatabases**, and **core.listDatatypes** commands to determine the names and types of target table columns. This information makes it easy to determine which columns to select, and which operations are allowed in the condition.

Example ad-hoc queries

This query is a simple ad-hoc query against the **OrionAuditLog** table.

```
URL: https://servername:port/remote/core.executeQuery?target=OrionAuditLog&select=(select
OrionAuditLog.UserName OrionAuditLog.CmdName)

Python: mc.core.executeQuery(target="OrionAuditLog", select="(select OrionAuditLog.UserName
OrionAuditLog.CmdName)");

OK:
User Name:
Action: Server restart

User Name: ga
Action: Login attempt

User Name: ga
Action: Upload Extension
```

This query returns the **CmdName** and **EndTime** for all audit log entries. The results are grouped alphabetically by the **CmdName**, then by the **EndTime**.

```
URL: https://servername:port/remote/core.executeQuery?target=OrionAuditLog& select=(select
OrionAuditLog.CmdName.OrionAuditLog.EndTime)& group=(group.OrionAuditLog.CmdName OrionAuditLog.EndTime)

Python: mc.executeQuery(target="OrionAuditLog", select="(select OrionAuditLog.CmdName
OrionAuditLog.EndTime)",
group="(group OrionAuditLog.CmdName OrionAuditLog.EndTime)");

OK:
User Name: ga
Priority: 1
```

```
Action: Login attempt
Details: Failed logon for user "ga" from IP Address: 172.1.6.1
Success: false
Start Time: 10/11/12 4:41:18 PM PDT
Completion Time: 10/11/12 4:41:18 PM PDT

User Name: system
Priority: 1
Action: Server restart
Details: Server was started.
Success: true
Start Time: 10/11/12 4:41:42 PM PDT
Completion Time: 10/11/12 4:41:42 PM PDT
```

This query returns all OrionTaskLog entries the "ga" user created. The results are listed in ascending order of the task **StartDate**.

```
https://servername:port/remote/core.executeQuery?target=OrionTaskLogTask& where=(where (eq
( OrionTaskLogTask.UserName "ga" ))) & order=(order (asc OrionTaskLogTask.StartDate) )

Python: mc.core.executeQuery(target="OrionTaskLogTask", where="(where ( eq ( OrionTaskLogTask .UserName
"ga" )))", order="(order (asc OrionTaskLogTask.StartDate) )");

OK:
Name: Deploy McAfee Agent
Start Date: 10/11/12 5:00:01 PM PDT
End Date: 10/11/12 5:00:38 PM PDT
User Name: ga
Status: 0
Source: scheduler
Duration: 36846

Name: Deploy McAfee Agent
Start Date: 10/11/12 5:04:19 PM PDT
End Date: null
User Name: ga
Status: 10
Source: scheduler
Duration: 1889951790
```

ePolicy Orchestrator uses symbolic expressions (S-expressions) internally as a portable and database-agnostic schema to define queries and operations. All values passed to each parameter must be valid S-expressions.

Create an ad-hoc query from a query definition

Queries stored in ePolicy Orchestrator can be exported and duplicated in a script.

If you have an existing persistent query, and you can define it as an ad-hoc query using **core.executeQuery**. Use the **Export Definitions** action in ePolicy Orchestrator to obtain the internal representation of the query. In almost all cases, the exported definition can be used to construct the **core.executeQuery** method call. For example, starting with an existing query as a model, then you modify the parameters, filtering, or sorting when executing the query from a script.

This is an example of using an exported persistent query to create an ad-hoc query.

Example

This is a typical exported query definition:

```
<query>
  <name language="en">My AuditLogQuery</name>
  <description language="en"></description>
  <property name="target">OrionAuditLog</property>
  <property name="tableURI">query:table?
orion.table.columns=OrionAuditLog.UserName%3AOrionAuditLog.CmdName%3A
OrionAuditLog.Success%3AOrionAuditLog.StartTime&amp;orion.table.order.by=OrionAuditLog.CmdName
&amp;orion.table.order=asc</property>
  <property name="conditionURI">query:condition?orion.condition.sexp=%28+where+%28+olderThan+
OrionAuditLog.EndTime+3600000++%29+%29</property>
  <property name="summaryURI">query:summary?orion.sum.query=false&amp;orion.query.type=table.table</
property>
</query>
```

Dissect this definition as:

- The **target** attribute is used directly as the **target** parameter of the ad-hoc query.
- The **conditionURI** attribute contains the S-Expression to use as the **where** parameter.

In an S-expression, the SELECT clause mirrors the limitations of a SELECT SQL clause. The SELECT clause operations include columns and unary operations on table columns. For example, Count, Max, Top, and others.

The unary operators work on only one expression of any one of the data types of the numeric data type category. For example, you cannot use SUM, or any other aggregate operations, with SELECT.

Note

The best way to become familiar with what SELECT clause arguments are supported, and their limitations in an ad-hoc query S-expression, is to export queries and examine their structure.

Remember that the exported form of the query contains strings that are URL-encoded. To form a valid query string, decode the URL-encoded characters. For example:

- `"+"` is used for a single space " "
- `%28` is an opening parenthesis `"("`
- `%29` is a closing parenthesis `")"`
- `%3A` is a colon `":"`

This is the equivalent ad-hoc URL query using the exported query definition:

```
https://servername:port/remote/core.executeQuery?target=OrionAuditLog&select=(select
OrionAuditLog.UserName OrionAuditLog.CmdName OrionAuditLog.Success
OrionAuditLog.StartTime)&where=(where(olderThan OrionAuditLog.EndTime 36000000))&order=(order(asc
OrionAuditLog.CmdName))
```

This is the equivalent ad-hoc Python query using the exported query definition:

```
mc.core.executeQuery(target="OrionAuditLog",
select="(select OrionAuditLog.UserName OrionAuditLog.CmdName OrionAuditLog.Success
OrionAuditLog.StartTime)",
where="(where(olderThan OrionAuditLog.EndTime 36000000))",
order="(order(asc OrionAuditLog.CmdName))");
```

This equivalent ad-hoc query returns this output:

```
OK:
User Name: ga
Priority: 1
Action: Login attempt
Details: Failed logon for user "ga" from IP Address: 172.1.5.1
Success: false
Start Time: 10/11/12 4:41:18 PM PDT
Completion Time: 10/11/12 4:41:18 PM PDT

User Name: system
Priority: 1
Action: Server restart
Details: Server was started.
Success: true
Start Time: 10/11/12 4:41:42 PM PDT
Completion Time: 10/11/12 4:41:42 PM PDT
.
.
.
```

Get information about registered databases and tables

You can use remote query commands to get information about registered databases and tables. This information is needed to create ad-hoc queries.

You can use the **core.listTables** command to get details about each table in the system, including the names and types of the columns.

Example output

This output is from the **core.listTables** command run on the **OrionAuditLog** table.

```
Name: Audit Log Entries
Target: OrionAuditLog
Type: target
Database Type:
Description: Retrieves information on changes and actions made by users of this server.
Columns:
   Name      Type          Select? Condition? GroupBy? Order? Number?
   --------- ------------- ------- ---------- -------- ------ -------
   AutoId    int           false   false      false    true   true
   UserId    int           false   false      false    true   true
   UserName  string_lookup true    true       true     true   false
   Priority  enum          true    true       true     true   false
   CmdName   string_lookup true    true       true     true   false
   Message   string        true    true       false    true   false
   Success   boolean       true    true       true     true   false
```

```
   StartTime timestamp     true    true       true     true   false
   EndTime   timestamp     true    true       true     true   false
Related Tables:
   Name
   ----
Foreign Keys: None
```

You can use the **core.listTables** command to list the columns, their types, whether the column can be used in the **select**, **condition**, **group** or **order** parameters, and whether the column is a number. The command also lists any registered related tables that can be joined with the **joinTables** parameter.

Get information on registered databases

You can use the **core.listDatabases** command to get details about each database in the system. This information can then be used to create ad-hoc queries.

In general, when issuing queries against targets that are not part of the default schema, prepend the database name to the target. For example, to reference an **"OutsidePolicy"** target that is part of an **"Outsider"** database, you would use the identifier **"target=Outsider.OutsidePolicy"**.

Queries with joins

You can use joins to display information from two or more tables. These joins are handled automatically.

To use the join functionality, specify the tables (**targets**) and the columns from those tables that you want in the **core.executeQuery select** parameter. The underlying query system computes the necessary join criteria transparently and displays the correct results.

Note

An error appears if there is no join information registered for the tables when the query runs.

You can use the **core.listTables** command to determine which tables are related and are capable of participating in joins. The **relatedTables** table property contains this join information.

Example query with a join

This example executes a simple join between the **OrionTaskLogTask** and **OrionTaskLogMessage** tables.

```
URL: https://servername:port/remote/core.executeQuery?target=OrionTaskLogTaskMessage&select=(select
OrionTaskLogTask.Name OrionTaskLogTaskMessage.Message )&joinTables=OrionTaskLogTask

Python: mc.core.executeQuery(target="OrionTaskLogTaskMessage", select="(select OrionTaskLogTask.Name
OrionTaskLogTaskMessage.Message )", joinTables="OrionTaskLogTask")

OK:
Name: New Task
Message: Purge audit log
```

```
Name: New Task
Message: Purge audit log (Purge log records older than: 1 days)
```

Retrieve hierarchical query results

Queries involving joined tables can return results in a hierarchy.

The **core.executeQuery** command can also be used in **joinTables** mode. In this mode, you specify only the tables you want joined. The results of the query are used as keys to perform a subquery for all related results from the joined table. The joined table creates a hierarchical result set that could contain nested results. The nested results level depends on how many join tables are specified.

To join tables using **core.executeQuery**, you specify a comma-separated list of tables to join as the **joinTables** parameter. You do not specify a select parameter when joining tables. The results are returned as a result hierarchy with each record of the parent table becoming the parent node of the related records in each child table. The hierarchy continues until all joined tables are displayed.

Example

This example executes a simple join between the **OrionTaskLog** and the **OrionTaskLogMessage** tables.

```
OK:
<?xml version="1.0" encoding="UTF-8"?>
<result>
 <list>
  <row>
   <OrionTaskLogTask.Name>New Task</OrionTaskLogTask.Name>
   <OrionTaskLogTask.StartDate>2010-11-23T13:01:37-08:00</OrionTaskLogTask.StartDate>
   <OrionTaskLogTask.EndDate>2010-11-23T13:01:37-08:00</OrionTaskLogTask.EndDate>
   <OrionTaskLogTask.UserName>ga</OrionTaskLogTask.UserName>
   <OrionTaskLogTask.Status>0</OrionTaskLogTask.Status>
   <OrionTaskLogTask.TaskSource>scheduler</OrionTaskLogTask.TaskSource>
   <OrionTaskLogTask.Duration>493</OrionTaskLogTask.Duration>
   <OrionTaskLogTaskMessage>
    <list>
     <row>
      <OrionTaskLogTaskMessage.Message>Purge audit log</OrionTaskLogTaskMessage.Message>
     </row>
     <row>
      <OrionTaskLogTaskMessage.Message>Purge audit log (Purge log records older than: 1 days)</
OrionTaskLogTaskMessage.Message>
     </row>
    </list>
   </OrionTaskLogTaskMessage>
  </row>
 </list>
</result>
```

This output shows the **core.executeQuery** command returned each top-level object (the **OrionTaskLogTask** record) and its two associated message records. The output is shown as XML to highlight the hierarchical arrangement of the results.

Limit query result depth

Queries that join tables can return results with deep hierarchies. You can control this depth using parameters.

A query that doesn't use joins returns tabular results. Tabular results are defined as having a depth of one. Queries that use the **joinTable** parameter return hierarchical results. Specifying more than one table in a **joinTable** parameter can result in a multi-level result set. Each level of objects increases the result depth by one. To prevent result sets from becoming too deep, the **core.executeQuery** command defaults to a maximum depth of 5 levels. If this default limit is too restrictive for the query being run, you can change it with a new value for the **depth** parameter.

Caution

Ad-hoc queries that return deep result sets can take a long time to run and can consume many system resources to generate. You should consider this when joining more than two tables, or increasing the query result depth limit.

Remote query commands

You can use a few commands when executing remote queries.

Note

Specify arguments followed by "=<>" by name. For example, the argument "target=" must be included in the command, core. For example, **executeQuery?target=EntitlementView**

Remote query commands

Command	Syntax	Description
core.executeQuery	**core.executeQuery queryId [database=<>]** **core.executeQuery target=<> [select=<>] [where=<>] [order=<>] [group=<>] [database=<>] [depth=<>] [joinTables=<>]**	Executes a query and returns the results as a list of objects.
core.listDatabases	**core.listDatabases**	Returns all databases the user is permitted to see as a list of objects.

Command	Syntax	Description
core.listTables	**core.listTables [table]**	Returns all database tables the user is permitted to see as a list of objects.
core.listQueries	**core.listQueries**	Returns all queries the user is permitted to see as a list of objects.
core.listDatatypes	**core.listDatatypes**	Returns a list of all types and their supported operations.

Ad-hoc query reference

Ad-hoc queries require operators, data types, and other information that you can access in your scripts.

In general, the data type in the column dictates the operators that the column supports. For example:

- Data type **string** supports the **startsWith** and **endsWith** operations
- Number columns support the **gt** and **lt** operations

Note

The data types and types of columns in a target can be determined using the **core.listTables** command.

These tables provide these basic elements and examples of their use.

General query datatypes

You must store data in ePolicy Orchestrator databases with specific types in remote queries.

General query datatypes

Type	Description
Int	An integer
string_lookup	A lookup string field

Type	Description
Enum	An enumerated value, stored in the database as an integer
Mac	A MAC address
Long	A long value
Float	A float value
Timespan	An SQL timespan
boolean	A Boolean value
string_enum	An enumerated value stored as a string instead of an integer
ipv4	An IPv4 address
ipv6	An IPv6 address

General S-Expression operations

ePolicy Orchestrator uses S-Expressions internally to define queries and operations. These S-Expressions must be used correctly.

These tables show operators you can use on S-expressions to define queries.

General S-Expression Operations

Operator	Description	Example
and	Logical AND of two or more S-Expressions.	**(where (and (eq OrionAuditLog.UserName "ga") (eq OrionAuditLog.Priority 1)))**
or	Logical OR of two or more S-Expressions.	**(where (or (eq OrionAuditLog.UserName "ga")**

Operator	Description	Example
		(eq OrionAuditLog.UserName "admin")))
not	Logical negation of an S-Expression.	**(where (not (eq OrionAuditLog.UserName "ga")))**
eq	Logical comparison for equality.	**(where (eq OrionAuditLog.UserName "ga"))**
ne	Logical comparison for inequality.	**(where (ne OrionAuditLog.UserName "ga"))**
gt	Logical comparison for greater than.	**(where (gt OrionAuditLog.Priority 1))**
lt	Logical comparison for less than.	**(where (lt OrionAuditLog.Priority 3))**
ge	Logical comparison for greater than or equal.	**(where (ge OrionAuditLog.Priority 2))**
le	Logical comparison for less than or equal.	**(where (le OrionAuditLog.Priority 2))**
is_Blank	Returns the row if the column value is null or the trimmed value is empty.	**(where (isBlank OrionAuditLog.UserName))**
not_isBlank	Returns the row if the column value is not null or the trimmed value is not empty.	**(where (not_isBlank OrionAuditLog.UserName)))**
in	Returns the row if the following item (usually a property or value) is in a following list. This is similar to the SQL **IN** directive.	**(where (in OrionAuditLog.UserName "ga" "bob"))**

Operator	Description	Example
contains	Returns the row if the column value contains the stubstring argument value.	**(where (contains OrionAuditLog.UserName "ga"))**
notContains	Returns the row if the column value does not contain the stubstring argument value.	**(where (notContains OrionAuditLog.UserName "ga"))**
startsWith	Returns the row if the string starts with the supplied string. Similar to column **in s%** in SQL.	**(where (startsWith OrionAuditLog.UserName "g"))**
endsWith	Returns the row if the column ends with the argument value.	**(where (endsWith OrionAuditLog.UserName "a"))**
like	Returns the row if the column contains a value that matches the pattern. Similar to **like** in SQL.	**(where (like OrionAuditLog.UserName "ga"))**
newerThan	Returns the row if the first timestamp parameter is newer than the second timestamp parameter.	**(where (newerThan OrionAuditLog.EndTime 3600000))**
olderThan	Returns the row if the first timestamp parameter is older than the second timestamp parameter.	**(where (olderThan OrionAuditLog.EndTime 36000000))**
between	Returns the row if the timestamp or IP address argument (column or value) likes between the two values.	**(where (between OrionAuditLog.EndTime (timestamp 1288888320000) (timestamp 1288888360000)))**
beforeNow	Returns the row if the timestamp value (column or value) is before the current time.	**(where (beforeNow OrionAuditLog.EndTime))**

Operator	Description	Example
match_any	Returns the row if the IP address matches one in the following list.	**(where(match_any EPOComputerProperties.IPV4x (ipv4 "192.168.1.1")))**
notBetween	Returns the row if the IP address is not between the two argument addresses.	**(where (notBetween EPOComputerProperties.IPV4x (ipv4 "192.168.1.1")(ipv4 "192.168.255.255")))**
not_in_subnet	Returns the row if the address is not in the given subnet.	**(where (not_in_subnet EPOComputerProperties.IPV4x (ipv4 "255.255.255.0") 25))**
in_subnet	Returns the row if the address is in the given subnet.	**(where (in_subnet EPOComputerProperties.IPV4x (ipv4 "255.255.255.0") 25))**
in_ipv6_subnet	Returns the row if the address is in an IPv6 subnet.	**(where (not_in_ipv6_subnet EPOComputerProperties.IPV6 (ipv6 "0.0.0.0") 1))**
not_match_any	Returns the row if the argument IPv6 address does not match one of the argument addresses.	**(where(not_match_any EPOComputerProperties.IPV6 (ipv6 "fc00::/7")))**

Selection-only S-Expressions

Operator	Description	Example
Distinct	Selects records that contain a distinct output value in a given column. Similar to the SQL **distinct** directive.	**(select (distinct) OrionAuditLog.UserName)**

Operator	Description	Example
	Note: If you use an **order** clause with **select distinct**, include the **order** columns in the selection.	
Top N	Selects the first N records to display. N must be an integer. Similar to the Microsoft SQL **Top**, or the MySQL **Limit** clause.	**(select (top 5) OrionTaskLogTask.Name OrionTaskLogTask.StartDate)**

S-Expression operator and datatype combinations

You can only use certain datatypes with each S-Expression operator. Use this table to confirm you are using the data types and operators correctly.

Operator	int	long	float	string	enum	mac	timestamp	timespan	string_enum	boolean	string_lookup	issue_type	ipv4	ipv6
eq	x	x		x	x				x	x	x	x	x	x
ne	x	x		x	x				x	x	x	x	x	x
gt	x	x	x											
lt	x	x	x											
ge	x	x												
le	x	x												
startsWith				x							x			
endsWith				x							x			
like				x							x			
in														
isBlank				x			x				x		x	x
not_isBlank				x			x				x		x	x
contains				x							x			
not_contains				x							x			
olderThan							x							
newerThan							x							
olderThanAbsolute							x							
newerThanAbsolute							x							
between							x						x	x
beforeNow							x							
mac_match_any						x								
mac_not_match_any						x								
match_any													x	x
notBetween													x	x
in_ipv6_subnet														x
not_match_any														x
not_in_ipv6_subnet														x

Special ePolicy Orchestrator datatypes

ePolicy Orchestrator creates a number of special datatypes used for various objects.

Special ePolicy Orchestrator datatypes

Type	Description
applied_tags	Used for the **EPOLeafNode.AppliedTags** column to determine which tags are applied to a system.
byte	Used for memory storage units such as **EPOComputerProperties.TotalPhysicalMemory** and **EPOComputerProperties.FreeMemory**.
eventID	Used for identifying events such as Threat Events.
group	Used for groups in the System Tree. For example **Groups.L1ParentID**.
multiselect_group	Used for selecting multiple System Tree groups.
managedState	A Boolean value used for the managed status of a device in the System Tree.
Megabytes	Used for disk storage units such as **EPOComputerProperties.FreeDiskSpace** and **EPOComputerProperties.TotalDiskSpace**.
optionGroup_enum	An enumeration of grouped lists such as policy categories.
Percentagefromstring	Used for percentages and compliance columns.
Policycolumn	Used for policies.
productVersion	Used for version numbers. Behaves normally when using equality functions, but handles numeric strings with more than one decimal point.
Rsdmac	Used for MAC addresses. Most often associated with detected systems.

Type	Description
Rsdoui	Used for OUIs associated with detected systems.
string_lookupWithResolver	Used to allow the user to choose from a list of known string values.
Threatcategory	A grouped enumeration used for listing the available threat categories.
DATversion	Used for DAT version information.
engineVersion	Used for engine version information.
datVersion	Used for DAT version information. Identical to DATversion.

Special ePolicy Orchestrator operators

ePolicy Orchestrator defines a number of operators designed to operate on its own custom datatypes.

ePolicy Orchestrator special operators

Operator	Description	Example
hasTag	Evaluates to true if the specified tag is applied to the system.	**(hasTag EPOLeafNode.AppliedTags 'fullTagName')**
containsTag	Evaluates to true if any tags applied to the system contain the specified partial tag name.	**(containsTag EPOLeafNode.AppliedTags 'partialTagName')**
hasTagExcluded	Evaluates to true if the specified tag is excluded on the system.	**(hasTagExcluded EPOLeafNode.AppliedTags 'excludedTagName')**
doesNotHaveTag	Evaluates to true if the specified tag is not applied to the system.	**(doesNotHaveTag EPOLeafNode.AppliedTags 'fullTagName')**

Operator	Description	Example
doesNotHaveAnyTag	Evaluates to true if the system has no tags applied to it.	**(doesNotHaveAnyTag EPOLeafNode.AppliedTags)**
childOf	Returns rows that are direct descendants of the given nodes.	**(childOf EPOLeafNode.parentId <nodeId1> <nodeId2> ...)**
descendsFrom	Returns rows that are descended from the given nodes.	**(descendsFrom EPOLeafNode.parentId <nodeId1> <nodeId2> ...)**
stateEq	Evaluates to true if the managed state of the system matches either "managed" or "unmanaged".	**(stateEq EPOLeafNode.ManagedState ["managed"|"unmanaged"])**
version_eq	Evaluates to true if the versions are equal using dotted version notation.	**(version_eq EPOMasterCatalog.ProductVersion '3.1.4.1')**
version_neq	Evaluates to true if the versions are not equal using dotted version notation.	**(version_neq EPOMasterCatalog.ProductVersion '3.1.4.1')**
version_ge	Evaluates to true if the first version is greater or equal to the second using dotted version notation.	**(version_ge EPOMasterCatalog.ProductVersion '3.1.4.1')**
version_lt	Evaluates to true if the first version is less than the second using dotted version notation.	**(version_lt EPOMasterCatalog.ProductVersion '3.1.4.1')**
threatcategory_belongs	Returns the row if the threat category belongs to the given group.	**(threatcategory_belongs EPOEvents.ThreatCategory 'av')**

Operator	Description	Example
threatcategory_not_belongs	Returns the row if the threat category does not belong to the given group.	**(threatcategory_not_belongs EPOEvents.ThreatCategory 'av')**
withinRepositoryDatVersion	Evaluates to true if the DAT version is within a supplied number of versions to what is in the repository.	**(withinRepositoryDatVersion EPOProdPropsView_VIRUSCAN.datver 3)**
not_withinRepositoryDatVersion	Evaluates to true if the DAT version is not within a supplied number of versions to what is in the repository.	**(not_withinRepositoryDatVersion EPOProdPropsView_VIRUSCAN.datver 3)**

Special ePolicy Orchestrator operator and datatype combinations

You can only use the specific combinations of datatypes and operators defined by ePolicy Orchestrator and shown in this table.

	applied_tags	byte	eventid	group	multiselect_group	managedstate	megabytes	optionGroup_enum	Percentagefromstring	Policycolumn	productVersion	Rsdmac	Rsdoui	String_lookupWithResolver	Threatcategory	DATversion	engineVersion	datVersion
eq		x	x				x	x	x	x		x	x	x		x		
ne		x	x				x	x	x	x		x	x	x		x		
gt		x	x				x		x									
lt		x	x				x		x							x		
ge		x	x				x		x							x		
le		x	x				x		x									
in			x						x									
hasTag	x																	
containsTag	x																	
hasTagExcluded	x																	
doesNotHaveTag	x																	
doesNotHaveAnyTag	x																	
childOf				x	x													
descendsFrom				x	x													
stateEq						x												
version_eq											x						x	x
version_neq											x						x	x
version_ge											x						x	x
version_lt											x						x	x
isBlank												x		x				
not_isBlank												x		x				
startsWith												x	x	x				
notStartsWith												x	x	x				
contains														x				
notContains														x				
threatcategory_belongs															x			
threatcategory_not_belongs															x			
withinRepositoryDatVersion																x		
not_withinRepositoryDatVersion																x		

COPYRIGHT

Trellix