vFabric Hyperic Web Services API

VMware vFabric Hyperic 5.0

This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see http://www.vmware.com/support/pubs.

EN-000962-00

You can find the most up-to-date technical documentation on the VMware Web site at:

http://www.vmware.com/support/

The VMware Web site also provides the latest product updates.

If you have comments about this documentation, submit your feedback to:

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Table of Contents

About vFabric Hyperic Web Services API

1

vFabric Hyperic Web Services API documents HQApi, a set of APIs for automating VMware® vFabric™ Hyperic® and Hyperic HQ configuration and administration processes. It has instructions for running each API from the command-line and introductory information for calling the APIs from Java.

Intended Audience

vFabric Hyperic Web Services API is intended for Hyperic administrators who to automate Hyperic configuration and administration processes, and for developers who want to access Hyperic backend functionality from Java programs.

Introduction to HQApi 2

*Topics marked with * relate to features available only in vFabric Hyperic.*

Overview of HQApi

HQApi is a set of Java APIs for accessing and updating HQ inventory and related configuration data. You can use the Hthe HQApi to access Hyperic data and resources from the command line, using scripts, or from Java. You can use the API to access platforms, servers, services, groups, escalations, users, and roles. For applicable inventory types, you can use the API to manage default metric collection settings and alert definitions.

Version Compatibility

Hyperic 5.0.0 ships with HQApi 5.0.0, and is not compatible with prior versions of HQApi.

What HQApi is Good For

HQApi allows you to circumvent the Hyperic user interface and directly access Hyperic Server functionality. This is useful for streamlining common Hyperic implementation and configuration, for instance, you can use command line tools to perform bulk updates to inventory and configuration data. The Hyperic API also allows you to implement interfaces between Hyperic and other management systems, for instance, you could write a utility that calls the APIs to extract inventory data for import into an asset tracking system.

With HQApi you can:

- create, update, and extract data about platforms, servers, services, groups,
- create, update, and extract metric collection settings for resource types and individual
- resources
- define alerts for resource types and individual resources
- create and update users, and roles

User Permissions and HQApi

A user's permissions and access to Hyperic resources, as defined in Hyperic, govern what that user can do with the HQApi. For example, the results returned to a user running a command-line tool to list resource data will only include only those resources to which the user is authorized to access in Hyperic. Similarly, users that do not have Hyperic "create" permissions for resources will not be able to use the HQApi to create resources.

How to Access HQApi

You can use HQApi:

- Programmatically - Each API can be called programmatically. This usage pattern is suitable for external systems or programs that need to expose or manipulate data in Hyperic. You can call the Java API methods directly; they are also exposed as RESTful web services on the Hyperic Server.
- Command-line tools - The HQApi package includes a command-line interface to the APIs. You can use the command line tools interactively from a shell, or in scripts. Command line access is targeted towards Hyperic administrators and users. The command line scripts allow for APIs to be invoked directly using arguments passed on the command line. Many commands return XML objects that you can edit and apply as updates to Hyperic. Common use cases are doing mass operations that would be tedious if done through the UI.

The command-line tools expose a subset of full APIfunctionality. The tools call the APIs, so any functionality available in the command-line tools is supported by the Java APIs themselves. The Java APIs provide additional functionality, for instance, richer functions for extracting metric data.

How to Install HQApi

HQApi is bundled with Hyperic. To download it:

1. Click the **Administration** tab in the Hyperic user interface.
2. In the "Plugins" section of the Hyperic **Administration** page, click **HQ Web Services API** to display a download page.
3. On the download page, click **hqapi1-client-4.0.0.ext.ext** to download the API archive.
4. Move the archive to the desired folder location and expand it.

API Package Contents

The HQApi archive expands to a folder named `hqapix.y.z`. The folder structure is shown below, including the contents of notable folders.

```
hqapi1-client-5.0.0
     bin
          hqapi.bat
          hqapi.sh
          hq-encrypt-password.sh
          hq-encrypt-password.sh
      lib
      conf
           log4j.properties
      hqapi1-4.0.0.jar
      javadoc
      lib
      logs
           commandline.log
      wadl
           HQApi1.wadl
      xsd
           HQApi1.xsd
```

Key API Files

- `hqapi1.jar` — Client JAR you run to interact with Hyperic backend.
- `hqapi.sh` or `hqapi.bat` — The script for starting and stopping the client.
- `log4j.properties`
- `HQApi1.wadl`
- `HQApi1.xsd`

HQApi Command-Line Tools 3

*Topics marked with * relate to features available only in vFabric Hyperic.*

Command-Line Tools Overview

This table lists the command-line tools:

Tool	Functionality
agent (see page 18)	List the agents registered with the HQ Server.
alertdefinition (see page 24)	List, update, create, and delete resource and resource type alert definitions. An alert definition specifies the alert details and the resource or resource type to which it is assigned.
alert (see page 40)	List information for fired alerts, fix an alert,acknowledge an alert in escalation, or delete an alert.
applications (see page 45)	Create, list, and update applications configured in HQ.
autodiscovery (see page 50)	List the platforms in the autodiscovery queue and approve them for import to inventory.
control (see page 53)	List supported control actions for a resource, list control action history for a resource, and perform issue a control action to a resource.
dependency (see page 57) *	List and manage dependency relationships between network or virtual hosts and platforms.
escalation (see page 66)	List and update Escalations.
event (see page 70)	List information about events – alerts, resource control actions, log events, or configuration events – that have been logged for a resource, or for all resources.
groovyshell (see page 75)	Run Groovy scripts from the command shell.
group (see page 77)	List resource group definitions, create or update groups, and delete groups.
maintenance (see page 83) *	Specify downtime periods for a resource or group, during which alerts on the resource(s) will not fire. Supported in vFabric Hyperic only.
metric (see page 90)	List and update the metric collection configuration for individual resources.
metricData (see page 94)	List the measurements for a particular metric for a particular resource.
metricTemplate (see page 97)	Output metric collection configuration for a specified resource type.
resource (see page 101)	Output resource data, create or update resources, and delete resources.
resourceprototype (see page 112)	Return the internal HQ ID and the name for resource types.

role (see page 115) *	List, update, and create non-system roles.
serverConfig (see page 120)	List and update selected HQ Server configuration properties,
user (see page 138)	List, update, and create users.

Invoking Command-Line Tools

To invoke the command-line tools from shell, you run the `hqapi.sh` script in the /bin directory of the client download, supplying the name of the tool you want to run. All commands have the following syntax:

```
./bin/hqapi.sh <top-level-command> <sub-command> <options>
```

where:

- `<top-level-command>` is the command-line tool name, for instance **agent** or **resource**.
- `<sub-command>` is a supported command option, for instance **list**, **sync**, or **delete**.
- `<options>` are one or more supported command qualifiers Supported options vary by command and subcommand, for example **resource list** supports the **--prototype** qualifier. See documentation for each command-line tool for supported command qualifiers.

This command returns a list of resources whose type is "MacOSX".

```
./bin/hqapi.sh resource list --prototype="MacOSX"
```

In the case of an error, the reason for the error will be printed to stderr and the process will exit with a negative return code.

A full listing of supported options can be obtained by passing the **-h** flag to any command.

Common Subcommands

Each command-line tool has one or more command options. The most common are described below.

list

list returns an XML object that describes one or more instances of the type of object you're working with - for instance, agents, resources, or roles. You can write the XML output to a file, and pipe it to the sync command to update the corresponding data in the HQ database. For example:

```
./bin/hqapi.sh resource list --prototype="HTTP" > http-resources.xml
```

sync

sync takes a valid XML object that describes items in HQ - for instance, platforms, servers, or services - and updates the HQ database accordingly.

For some command-line tools, **sync** can create new items in HQ; for others, **sync** only updates items that already exist in HQ.

Note: When you use **sync** to create a new item, do not specify the `id` attribute for the new item.

To update an item, typically you write the XML object to a file using the **list** option, edit the contents as desired, and then use **sync** to update attributes for an existing item in HQ.

By default, **sync** reads the XML over stdin. For example:

```
cat http-resources.xml | ./bin/hqapi.sh resource sync
```

As desired, you can specify the location of the file with the **--file** command qualifier.

```
./bin/hqapi.sh resource sync --file=http-resources.xml
```

delete

delete removes an item from the HQ database. The **delete** option supports only "one-at-a-time" deletions. You supply the name or ID of the item to delete.

Required Command Qualifiers and How to Supply Them

For each command line tool there is a set of required and optional command qualifiers.

The optional qualifiers usually set the scope of a command, for example, tell it to list a single resource instead of all the resources on a platform. Scoping qualifiers vary by command and are documented in the each command's reference documentation.

The qualifiers that every command requires are the parameters that specify how to connect to the HQ Server:

--host	HQ server hostname
--port	HQ server port
--user	user to connect as

--password	password for the given user
--secure	Flag to indicate if SSL should be used.

The optional qualifiers that every command supports are:

--properties	Supplies the path to a properties file that sets the values of the server connection settings listed above. Not necessary if you define the properties in the default location — `~/.hq/client.properties` — as described below in Define a Connection Properties File (see page 16)

Define a Connection Properties File

To avoid specifying the connection parameters each time you run a command, you can specify them in:

- `hqapi1-client-n.n.n/conf/client.properties` — The APIs honor this directory first.
- `~/.hq/client.properties` — Create the directory if it does not exist. Under Windows, you can create the `.hq` directory from a DOS shell. Use the **mkdir** command to create .hq under your home home directory.

For example:

```
host=localhost
port=7443
secure=true
user=hqadmin
password=hqadmin
```

or if you have encrypted the Hyperic password, as described below in Encrypt Hyperic Password (see page 16).

```
host=localhost
port=7443
secure=true
user=hqadmin
encryptedPassword=ENC(5EJNmq7JLKcIkTVlcahLrg==)
encryptionKey=popcorns
```

Encrypt Hyperic Password

`client.properties` supports encrypted passwords.

Use the `hqapi1-client-n.n.n/hq-encrypt-password.sh`, or under Windows `hqapi1-client-n.n.n/hq-encrypt-password.bat` to encrypt your password. You enter your password twice, and they supply an encryption key. For example:

```
$ sh bin/hq-encrypt-password.sh
Enter password:********
Re-Enter password:********
Encryption Key: popcorns
The encrypted password is ENC(5EJNmq7JLKcIkTVlcahLrg==)
```

Define the encrypted password and encryption key in `client.properties`, as described in the previous section.

Supply Properties File Path on Command Line

The APIs look for server connection properties in `hqapi1-client-n.n.n/conf`, and if not found there, in `~/.hq/client.properties`.

If you want to put `client.properties` in a different location, you can do so, and then supply the path to the file on the command line, using the `--properties` command qualifier.

Tips and Reminders for CLI Tools

Sync with Caution

There is no un-do command for updates you make with the API command line tools. Carefully review the XML you supply to a command line tool's **sync** method.

How to Find a Resource ID

When you run an HQ API command that operates on a particular resource you must identify the resource by its internal HQ ID. To obtain the ID for a resource, you can run the **resource list** command to find it. The **resource list** command returns inventory and configuration properties for one or more resources. For example, this command:

```
sh bin/hqapi.sh resource list --platform=vion.intranet.hc.net
```

returns the results shown below; the third line defines the internal HQ ID for the resource: `Resource id="27054"`

```
<ResourcesResponse>
    <Status>Success</Status>
    <Resource id="27054" name="vion.intranet.hc.net" description="CentOS 5.2" location="">
        <ResourcePrototype id="10120" name="Linux"/>
        <Agent id="12423" address="10.0.0.161" port="-1" version="4.3.0-EE" unidirectional="true"/>
        <Ip address="127.0.0.1" mac="00:00:00:00:00:00" netmask="255.0.0.0"/>
        <Ip address="10.0.0.131" mac="00:50:56:8F:19:24" netmask="255.255.255.0"/>
        <ResourceInfo key="fqdn" value="vion.intranet.hc.net"/>
    </Resource>
</ResourcesResponse>
```

You can use **resource list** to query by:

- resource name - regex for resource name
- platform name - to get properties for it and optionally, its child resources.
- resource type - by query by platform name and get properties for it, and optionally, the resources running on it.

HQApi agent command 4

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

agent Functionality

You can use the **agent** command to obtain information about a Hyperic Agent, and to push a new bundle or plugin to an agent.

agent Command Options

agent list

The **agent list** command returns an `AgentsResponse` element, which contains an `Agent` element for each agent registered with the Hyperic Server.

Structure of an AgentsResponse Element

```
<AgentsResponse>
     <Agent>
</AgentsResponse>
```

where there is one `<Agent>` element for each Hyperic Agent reporting to the Hyperic Server.

Content of an Agent Element

Each `Agent` element includes these attributes:

- `unidirectional` — whether agent uses unidirectional communications (for vFabric Hyperic only)
- `version` — agent software version
- `port` — port on which the agent listens for Hyperic Server communications
- `address` — agent's IP address
- `id` — Hyperic internal ID for the agent

agent list Command Qualifiers

none

agent list Example

```
$ ./bin/hqapi.sh agent list
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AgentsResponse>
<Status>Success</Status>
    <Agent id="10901" address="10.17.184.239" port="4749" version="4.5.1.2.BUILD-SNAPSHOT"
unidirectional="false"/>
</AgentsResponse>
```

agent ping

agent ping Command Qualifiers

You must supply use command qualifier(s) to identify the target agent. You can supply:

- the agent's IP address and port,
- the FQDN of the platform where the agent runs, or
- the agent's internal Hyperic ID

Qualifier	Description	Required
--agentAddress	The address of the agent to ping.	
--agentPort <Integer>	The port of the agent to ping. Required if target agent is identified by --agentAddress	
--fqdn	The platform FQDN of the agent to ping	
--id <Integer>	The id of the agent to ping.	

agent ping examples

```
$ ./bin/hqapi.sh agent ping --id=10001
Successfully pinged agent 127.0.0.1:2144
```

```
$ ./bin/hqapi.sh agent ping --fqdn=My-Machine-MacBook-Air.local
Successfully pinged agent 127.0.0.1:2144
```

```
$ ./bin/hqapi.sh agent ping --agentAddress=127.0.0.1 --agentPort=2144
```

agent transferPlugin

The **agent transferPlugin** command copies a plugin from the Hyperic Server to a target agent.

agent transferPlugin Command Qualifiers

You must supply use command qualifier(s) to identify the target agent. You can supply:

- the agent's IP address and port
- the FQDN of the platform where the agent runs
- the agent's internal Hyperic ID

Qualifier	Description	Required
--agentAddress	The address of the agent to ping.	

--agentPort <Integer>	The port of the agent to ping. Required if target agent is identified by --agentAddress
--fqdn	The platform FQDN of the agent to ping
--id <Integer>	The id of the agent to ping.
--plugin <PluginName>	The name of the plugin to transfer.

agent transferPlugin Examples

```
bin/hqapi.sh agent transferPlugin --plugin=jboss-plugin.jar --agentAddress=127.0.0.1
--agentPort=2144

Successfully transferred plugin jboss-plugin.jar to 127.0.0.1:2144
```

agent bundle-list

The **agent bundle-list** command returns an `AgentBundleFilesResponse` element, which contains an `AgentBundleFile` element for each agent bundle in the Hyperic Server's `ServerHome/hq-engine/hq-server/webapps/ROOT/WEB-INF/hq-agent-bundles` directory.

Structure of an AgentBundleFilesResponse Element

```
<AgentBundleFilesResponse>
      <AgentBundleFile>
</AgentBundleFilesResponse>
```

where there is one `<AgentBundleFile>` element for each agent bundle.

Content of an AgentBundleFile Element

Each `AgentBundleFile` element contains a single attribute:

- `name` — Name of the bundle file.

agent bundle-list Command Qualifiers

none

agent bundle-list Example

```
$ ./hqapi1-client-4.1.0.BUILD-SNAPSHOT/bin/hqapi.sh agent bundle-list
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AgentBundleFilesResponse>
<Status>Success</Status>
     <AgentBundleFile name="agent-4.5.1.2-custom.tar.gz"/>
     <AgentBundleFile name="agent-4.5.1.2.tar.gz"/>
</AgentBundleFilesResponse>
```

agent bundle-status

The **agent bundle-status** command returns the name of the bundle an target agent is running.

agent bundle-status Command Qualifiers

You must supply use command qualifier(s) to identify the target agent. You can supply:

- the agent's IP address and port
- the FQDN of the platform where the agent runs
- the agent's internal Hyperic ID

Qualifier	Description
--agentAddress	The address of the target agent.
--agentPort <Integer>	The listen port of the target agent. Required if target agent is identified by --agentAddress
--fqdn	The platform FQDN of the agent.
--id <Integer>	The id of the target agent

agent bundle-status example

```
$ ./hqapi1-client-4.1.0.BUILD-SNAPSHOT/bin/hqapi.sh agent bundle-status --id 10403
Current Bundle: agent-4.5.1.2-custom-20110317154325
```

bundle-push

The **agent bundle-push** command copies a selected bundle from the Hyperic Server to a target Hyperic Agent.

agent bundle-push Command Qualifiers

You must supply use command qualifier(s) to identify the target agent. You can supply:

- the agent's IP address and port
- the FQDN of the platform where the agent runs
- the agent's internal Hyperic ID

Qualifier	Description

--agentAddress	The address of the target agent.
--agentPort	The listen port of the target agent. Required if target agent is identified by --agentAddress
--fqdn	The platform FQDN of the agent.
--id	The id of the target agent
--bundle	The filename of the bundle to push to the target agent.

agent bundle-push example

```
$ ./hqapi1-client-4.1.0.BUILD-SNAPSHOT/bin/hqapi.sh agent bundle-push --id 10403 --bundle
"agent-4.5.1.2-custom.tar.gz"
```

HQApi alertdefinition command 5

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

This page documents v5.0 of HQApi.

alertdefinition Functionality

The **alertdefinition** command has options to output resource and (in vFabric Hyperic only) resource type alert definitions, create them, update them, and delete them.

An alert definition defines the target resource or (vFabric Hyperic only) resource type, the alert conditions, and as available, (in vFabric Hyperic) the escalation associated with the alert definition. This is useful for:

- Initial setup of alerts across a deployment.
- Implement changes in alerting strategy across a deployment.
- Exporting all alert definitions for import to another system.

Understanding Alert Definition Options

For information about alert definitions options, including condition types and action filtering, refer to Define an Alert for a Resource.

alertdefinition Command Options

alertdefinition list

The **alertdefinition list** command returns an `AlertDefinitionsResponse` object, which contains one or more alert definitions.

alertdefinition list Command Qualifiers

If you do not supply a qualifier, the **alertdefinition list** command returns all resource alert definitions, including those that were:

- configured for a specific resource instance
- (in vFabric Hyperic only) automatically created for a resource instance as a result of a resource type alert definition

When you run the **alertdefinition list** command you can supply one or more of the following qualifiers to limit what alert definitions are returned.

Qualifier	Description
`--alertName regex`	Where `regex` is a regular expression.Returns alert definitions with names that match the given regular expression.
`--alertPriority integer`	If specified, only include alerts with the given priority. 3 = high, 2 = medium, 1 = low
`--children`	If used with `resourceId`, includes children of resource.
`--conditionCount integer`	Where `integer` is an integer value. Returns alert definitions with the number of conditions specified.
`--conditionTypeExclude integer`	If specified, exclude alert definitions which have at least one condition of the given type.

`--conditionTypeInclude integer`	If specified, only show alert definitions which have at least one condition of the given type.
`--escalation EscName`	Where `EscName` is the name of an escalation. Return alert definitions to which the specified escalation is assigned.
`--excludeTypeAlerts`	Returns resource alerts that were configured for a specific resource instance. Resource alert definitions that (in vFabric Hyperic only) are automatically created for resource instances as a result of a resource type alert definition are *not* returned. This qualifier cannot be used in conjunction with the `--typeAlerts` qualifier.
`--excludeTypeIds`	Excludes the HQ-internal ID for alert definitions from the XML output. This qualifier is only valid used in conjunction with the `--typeAlerts`qualifer. Excluding the internal ID for alert definitions is useful if you want to export a list of definitions for import into another system.
`--id`	If specified, return the alert definition with the given ID.
`--group GroupName`	Where `GroupName` is the name of a group. Limits results to alert definitions for resources in a specified group.
`--platform`	Return all alerts on the given platform and all descendant children
`--resourceDescription`	If specified, only show alert definitions belonging to a resource with a description matching in whole or part the given description.
`--resourceId integer`	If specified, only show alert definitions belonging to a resource
`--resourceName regex`	Where `regex` is a regular expression. Limits results to alert definitions for resources whose name matches a specified regular expression.
`--typeAlerts`	Returns all resource type alert definitions. The "child" resource alert definitions that HQ automatically creates for resources of the type specified by a resource type alert definition are *not* returned.

Structure of an AlertDefinitionsResponse

The `AlertDefinitionsResponse` object returned by the alertdefinition list command has this element structure:

```
AlertDefinitionsResponse
     AlertDefinition
          Resource
          ResourcePrototype
          AlertCondition
          AlertAction
          AlertActionConfig
          Escalation
```

where there are:

- 1 or more `alertdefinition` elements, each with
  - either a `Resource` or a `ResourcePrototype` element
  - 1 or more `AlertCondition` elements and,
  - optionally, 1 `escalation` element
  - optionally, 1 or more `AlertAction` elements, each with
    - 1 or more `AlertActionConfig` elements

The element structure of an `alertdefinition` for a resource type alert is the same as for a resource alert, except it contains a `ResourcePrototype` element instead of a `Resource` element.

Attributes in an AlertDefinitionsResponse

This table defines the attributes in a `AlertDefinitionsResponse`.

Note that there are eight different types of `AlertCondition` elements, each with a different "type" value.

Element	Attribute	Description	Values
`alertdefinition`			
	`id`	Internal HQ ID for the alert definition. This value is assigned by HQ Server when the alert definition is created. When you create an alert definition using the API, you do not supply the value of `id`. When you update an existing alert definition using `sync`, to identify the definition to update by its `id`.	Numeric value, for example, "10002"
	`name`	Name assigned to the alert definition by the person who created it.	"Linux Platform Availability"
	`description`	Description assigned to the alert definition by the person who created it.	
	`priority`	Numeric value for the priority assigned to the alert definition.	"1", "2", or "3" correspond to Low, Medium, and High.
	`active`	Whether or not the alert definition is currently active.	"true" or "false"
	`enabled`		"true" or "false"

	`frequency`	Use to configure the the alert definition's **Enable Actions(s**) behavior, which governs whether an alert will fire each time the condition(s) are met. For information on these options, see Enable Actions. ▪ "0" - Each time the conditions are met. ▪ "2" - Once every __ times conditions are met within a time period of... **Note**: Use Use `count` and `range` to specify the timing values.	▪ "0" ▪ "2"
	`count`	Use `count` and `range` to specify the timing values for an alert definition that is configured to fire "Once every __ times conditions are met within a time period of __ minutes". First make sure that `frequency` is set to "2", and then use: ▪ `count` to specify the "once every __ times" value ▪ `range` to specify the "time period of __..." value in seconds.	Non-zero integer.
	`range`	Use `range` and `count` to configure the alert definition to fire " Once every __ times conditions are met within a time period of __ minutes". First make sure that `frequency` is set to "2", and then use: ▪ `count` to specify the "once every __ times" value ▪ `range` to specify the "time period of __ ..." value in seconds.	Non-zero integer.
	`willRecover`	Used to configure the "Generate one alert and then disable alert definition until fixed" behavior. For information, see Enable Actions.	"true" or "false"
	`controlFiltered`	For a description of the behavior this attribute controls, please see Disregard Control Actions for Related Alerts.	"true" or "false"
	`notifyFiltered`	For a description of the behavior this attribute controls, please see Disregard Control Actions for Related Alerts.	"true" or "false"

	`parent`	In vFabric Hyperic, the internal ID of the parent resource type alert definition, if there is one.	Valid ID of a resource type alert definition in HQ. `"0"` indicates no parent.
	`ctime`		
	`mtime`		
`ResourcePrototype`			
	`name`	The name of the resource type.	Example values: ■ "Linux" ■ "Tomcat 6.0 Connector"
	`id`	The internal HQ ID for a resource type.	Example value:"10329"
`Resource`			
	`name`	The name of a resource as it appears in the HQ user interface.	Example value: "melba HQ Tomcat 6.0 9009 Tomcat 6.0 Connector"
	`id`	The internal HQ ID for a resource.	Example value: "10653"
`AlertCondition`			
	required	Whether condition is required to trigger the alert	"true" or "false"
	`type`	Specifies the type of an alert condition: ■ "1" - metric condition (compare to absolute value) ■ "2" - metric condition (compare to baseline) ■ "3" - control action condition ■ "4" - metric condition (change in value) ■ "5" - recovery alert condition ■ "6" - inventory property condition ■ "7" - event/log condition ■ "8" - config changed condition For more information about each type of alert condition, see condition types.	An integer value of 1 though 8
`AlertAction`			
	`id`	The internal HQ ID for the action.	

	`className`	The name of the Hyperic class that performs the action type.	
`AlertActionConfig`			
	`key`	The key of a parameter that the action type requires type of action.	For example, `key="notifyType"` for an email action.
	`value`	The value of a parameter required to run the action.	For example, `value="users"` for an email action.
	`id`	The internal HQ ID for an escalation.	
	`name`	The name assigned to the escalation by the user that created it.	
	`description`	The text description for the escalation supplied by the user that created it.	

AlertCondition Attributes for Different Condition Types

This table defines the attributes in `AlertCondition` element for each of the eight different condition types. For information about each type of condition, see Alert Conditions.

Condition Type	Attribute	Description	Value(s)
"1" - metric condition (compare to absolute value)			
	`thresholdValue`	The absolute value to which the actual metric value is compared.	"100.0"

	`thresholdComparator`	The operator used to compare the the actual metric value to `thresholdValue` In the UI, the operators are shown as:
		■ "greater than"
		■ "less than"
		■ "equal to"
		■ "not equal to"
			Allowable values (which correspond to the operators listed in cell to the left):
			`>`
			`<`
			`=`
			`!`

	`thresholdMetric`	The metric the condition is based upon. Allowable values depend on the type of resource to which the alert definition is assigned.	Example value: ■ "Availability"

"2" - metric condition (compare to baseline)			
	`baselineType`	Specifies whether the metric value is compared to ■ "max - maximum value ■ "mean" - baseline value ■ "min" - minimum value	■ "max ■ "mean" ■ "min"
	`baselinePercentage`	Specifies a threshold value, in terms of percentage of the selected baseline type (max, baseline, or min value), as specified by `baselineType`.	"60.0"
	`baselineMetric`	The metric the condition is based upon. Allowable values depend on the type of resource to which the alert definition is assigned.	"Free Memory"

	`baselineComparator`	The operator used to compare the the actual metric value to `baselinePercentage`. In the UI, the operators are shown as:
		■ "greater than"
		■ "less than"
		■ "equal to"
		■ "not equal to"
			Allowable values (which correspond to the operators listed in cell to the left):
			`>`
			`<`
			`=`
			`!`

"3" - control action condition			
	`controlAction`	Specifies the control action the condition is based upon. The condition is met when the specified action has been run with the completion status specified by `controlStatus`. Use this condition type only for resources with supported control actions.The values you can specify for `controlAction` depend on the type of resource type to which the alert is assigned.	Example values include: ■ "runGarbageCollector" ■ "vacuum" ■ "restart"

	`controlStatus`	Specifies the completion status for the control action specified by `controlAction` that makes the condition true. Use this condition type only for resources with supported control actions.	▪ "In Progress" ▪ "Completed" ▪ "Failed"
"4" - metric condition (change in value)			
	`metricChange`	The metric the condition is based upon. Allowable values depend on the type of resource to which the alert definition is assigned.	"Active Thread Count"
"5" - recovery alert target			
	`recover`	In an alert definition for a recovery alert, use this condition type to identify the target alert definition, for which the current alert is a recovery alert. **Note**: This is sort of a "pseudo" alert condition. From the UI perspective it is *not* a condition. In the alert definition UI, you identify the recovery alert's target by choosing it from the "Recover for..." pull-down, which displays a list of alert definitions. In addition to a condition of type="5", a recovery alert must also have an true alert condition - one that can be evaluated as "true" or "false", and is "true" when the target alert's condition is "false".	"Free Memory > 60.0% of Baseline"
"6" - inventory property condition			

	`property`	Specifies the internal key for the inventory property the condition is based upon. The inventory properties you can use in an alert vary by resource type. When you define an alert condition in the HQ user interface, the inventory properties you can use are presented in a selector list. Note however that the property names displayed in the list are friendly names, as opposed to the internal key you must use when defining an alert condition via the API. You can determine the properties for a resource type, and the internal key for each, by running the `resource list` command with the `--verbose` option for a resource of that type. The command will return a `ResourceProperty` element for each inventory property that you can use in alert conditions for the current type. A `ResourceProperty` element is a key=value pair. For more information, see List Resources of the Same Type Verbosely (see page 108)	Example values: ▪ "arch" ▪ "cpuSpeed" ▪ "defaultGateway" ▪ "ip" ▪ "primaryDNS"
"7" - event/log condition			
	`logMatches`	Specifies a string to look for in log events. The condition is met when when an event of the severity level specified by `logLevel` that contains the string specified by `logMatches` (if a value was supplied) is logged. The target resource must have log tracking enabled.	"any text you want"
	`logLevel`	Specifies a log message severity level. The condition is met when an event of this severity level that contains the string specified by `logMatches` (if a value was supplied) is logged. The target resource must have log tracking enabled.	▪ "ANY" ▪ "ERR" ▪ "WRN" ▪ "INF" ▪ "DBG"
"8" - config changed condition			
	`configMatch`		"testfile"

AlertActionConfig Attributes for Different Action Types

This table defines the attributes in an `AlertActionConfig` element for each of the different action types. For information about each type of action, see Alert Actions.

Action Type	Class	Key	Value(s)
Email Notification	com.hyperic.hq.bizapp.server.action.email.EmailAction	`notifyType`	▪ "users" — Equivalent to selecting **Notify HQ Users** in the alert definition UI. ▪ "roles" — Equivalent to selecting **Notify Roles** in the alert definition UI. ▪ "email" — Equivalent to selecting **Notify Other Recipients** in the alert definition UI.
Script	com.hyperic.hq.bizapp.server.action.control.ScriptAction	`script`	path to script
OpenNMS action	org.hyperic.hq.bizapp.server.action.integrate.OpenNMSAction	▪ `port` ▪ `server`	▪ OpenNMS listen port ▪ OpenNMS listen address

control action	com.hyperic.hq.bizapp.server.action.control.ControlAction	▪ `resourceId` ▪ `action`	▪ Internal Hyperic ID for the resource upon which the control action will be performed ▪ Action to performed. Allowable values vary by resource type.

alertdefinition sync

The **alertdefinition sync** command takes an `AlertDefinitionsResponse` and syncs any changes back into the HQ inventory. All attributes of the `alertdefinition` element, new or changed `AlertCondition` elements, and the optional `Escalation` element will be updated.

You can use the command qualifiers described in the following section to assign and remove alert actions or escalations to alert definitions in the sync.

Note: When you use **sync** to create a new item, do not specify the `id` attribute for the new item. See sync (see page 15) for more information.

alertdefinition sync Command Qualifiers

Qualifier	Description
`--assignControlAction`	If specified, assign the given Control Action to all alert definitions in this sync.
`--assignEscalation`	If specified, assign the given Escalation to all alert definitions in this sync.
`--assignOtherNotification`	If specified, assign notification to the given comma separated list of email addresses.
`--assignRoleNotification`	If specified, assign notification to the given comma separated list of roles.
`--assignScriptAction`	If specified, assign the given script action to all alert definitions in this sync.
`--assignUserNotification`	If specified, assign notification to the given comma separated list of users.
`--batchSize <Integer>`	Process the sync in batches of the given size.
`--clearActions`	If specified, clear alert actions from all alert definitions in this sync.
`--clearEscalation`	If specified, clear the assigned escalation from all alert definitions in this sync.

alertdefinition delete

The **alertdefinition delete** command removes the specified alert definition from HQ. If you delete a resource type alert definition, its child alert definitions (which are associated with each resource of the type specified in the resource type alert defintion) are also deleted.

The delete command requires a single **--id** argument that specifies the `AlertDefinition` to delete.

alertdefinition create

alertdefinition create Command Qualifiers

When you run the **alertdefinition list** command you can supply one or more of the following qualifiers to limit what alert definitions are returned.

Qualifier	Description
`--assignControlAction`	If specified, assign the given Control Action to all alert definitions in this create.
`--assignEscalation`	If specified, assign the given Escalation to all alert definitions in this create.
`--assignScriptAction`	If specified, assign the given script action to all alert definitions in this create.
`--batchSize <Integer>`	Process the create in batches of the given size.
`--equals <Double>`	Sets condition to if metric value = threshold.
`--file`	If specified, use the given file for commands that take XML input. If not specified, stdin will be used.
`--greaterthan <Double>`	Sets condition to if metric value > threshold.
`--group`	Create only for resources in the specified group
`--lessthan <Double>`	Sets condition to if metric value < threshold.
`--metric`	The metric name to use. Not valid with --templateDefinition.
`--name`	The name of the alert definition to create. Not valid with --templateDefinition.
`--notequalto <Double>`	Sets condition to if metric value is not equal to threshold.
`--priority <Integer>`	Sets priority. 1 = low, 2 = medium (default), 3 = high
`--prototype`	Specifies the prototype to find.
`--recoveryequals <Double>`	Sets the recovery alert to if metric value = threshold.
`--recoverygreaterthan <Double>`	Sets the recovery alert to if metric value > threshold.
`--recoverylessthan <Double>`	Sets the recovery alert to if metric value < threshold.

`--recoveryname`	Optionally specify name of recovery alert. If not specified then "fixed" is added to alert definition name.
`--recoverynotequalto <Double>`	Sets the recovery alert to if metric value != threshold.
`--regex`	Use pattern to find matching resources
`--templateDefinition <Integer>`	The id of the alert definition to use as a template. If specified only resource type alerts will be created.
`--typeAlerts`	If specified only resource type alerts will be created.
`--willrecover`	If specified sets the willRecover flag. Equivalent to "Generate one alert and then disable alert definition until fixed".

alertdefinition Command Examples

alertdefinition list

In this example, no qualifiers were supplied. All resource alert definitions are returned.

```
<AlertDefinitionsResponse>
    <Status>Success</Status>
    <AlertDefinition mtime="1313779425236" ctime="1313775515992" id="10016" name="alert1"
description="" priority="2" enabled="true" active="true" frequency="2" count="6" range="1"
willRecover="true" notifyFiltered="false" controlFiltered="false">
        <Resource id="10938" name="Marie-McGarrys-MacBook-Air.local"/>
        <Escalation id="100" name="Default Escalation" description="This is an Escalation Scheme
created by HQ that performs no actions" pauseAllowed="false" maxPauseTime="300000"
notifyAll="false" repeat="false"/>
        <AlertCondition required="true" type="4" metricChange="Availability"/>
        <AlertAction id="10006" className="com.hyperic.hq.bizapp.server.action.email.EmailAction">
            <AlertActionConfig key="notifyType" value="users"/>
            <AlertActionConfig key="names" value="hqadmin"/>
        </AlertAction>
        <AlertAction id="10007" className="com.hyperic.hq.bizapp.server.action.email.EmailAction">
            <AlertActionConfig key="notifyType" value="email"/>
            <AlertActionConfig key="names" value="marie@linsdall.com"/>
        </AlertAction>
        <AlertAction id="10008"
className="com.hyperic.hq.bizapp.server.action.control.ScriptAction">
            <AlertActionConfig key="script" value="/scripts/action.sh"/>
        </AlertAction>
        <AlertAction id="10009"
className="org.hyperic.hq.bizapp.server.action.integrate.OpenNMSAction">
            <AlertActionConfig key="port" value="5817"/>
            <AlertActionConfig key="server" value="333.444.444.4444"/>
        </AlertAction>
        <AlertAction id="10010"
className="com.hyperic.hq.bizapp.server.action.control.ControlAction">
            <AlertActionConfig key="resourceId" value="10945"/>
            <AlertActionConfig key="action" value="Vacuum"/>
        </AlertAction>
    </AlertDefinition>
</AlertDefinitionsResponse>
```

alertdefinition list --typeAlerts

This example returns all resource type alert definitions in HQ. In this case, there is just one resource type alert definition in HQ. The resource type the alert definition applies to - "JBoss 4.2 Stateless Session EJB" - is specified in the `ResourcePrototype` element Y

When you use the `--typeAlerts` qualifier, "child" resource alert definitions that HQ automatically creates for resources of the type specified by a resource type alert definition are *not* returned.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AlertDefinitionsResponse>
<Status>Success</Status>
<AlertDefinition controlFiltered="false" notifyFiltered="false" willRecover="true" range="0"
count="0" frequency="0" enabled="true" active="true" priority="2" parent="0" description=""
name="JbossStatelessResourceType" id="10001">
<ResourcePrototype name="JBoss 4.2 Stateless Session EJB" id="10182"/>
<AlertCondition metricChange="Availability" type="4" required="true"/>
</AlertDefintionsResponse>
```

AlertCondition Examples

The fragment below has an example of each type of `AlertCondition`

```
<AlertCondition thresholdValue="2.0" thresholdComparator=">" thresholdMetric=
"Load Average 5 Minutes" type="1" required="true"/>
<AlertCondition baselineType="mean" baselinePercentage="0.0" baselineComparator=">"
baselineMetric="Zombie Processes" type="2" required="true"/>
<AlertCondition controlStatus="Completed" controlAction="start" type="3" required="true"/>
<AlertCondition metricChange="Zombie Processes" type="4" required="true"/>
<AlertCondition recover="Availability Down" type="5" required="true"/>
<AlertCondition property="arch" type="6" required="true"/>
<AlertCondition logMatches="login" logLevel="INF" type="7" required="true"/>
<AlertCondition configMatch="/var/log/messages" type="8" required="true"/>
```

alertdefinition sync

In this example, the contents of the typedefs.xml file are written to HQ.

```
bash-3.2$ ./bin/hqapi.sh alertdefinition list \--typeAlerts > typedefs.xml
<EDIT typedefs.xml>
bash-3.2$ cat typedefs.xml \| ./bin/hqapi.sh alertdefinition sync
Successfully synced 14 alert definitions.
bash-3.2$
```

alertdefinition delete

In this example, the `alertdefintion` whose internal ID is 10045 is deleted.

```
bash-3.2$ ./bin/hqapi.sh alertdefinition delete \--id=10048
Successfully deleted alert definition id 10048
bash-3.2$
```

alertdefinition create - based on existing alert definition

In this example, a new alert definition is copied from an existing definition and assigned to members of a group that have the same resource type.

```
hqapi.sh alertdefinition create --templateDefinition 10100 --group "Linux" --prototype "Linux"
```

Where `--templateDefinition 10100` is the alert definition that already exists that you want to clone to all resources of type "Linux" in the group "Linux".

You can find the alert definition id with the `alertdefinition list` command. The template definition must be of the same type as the target members. The group does NOT need to be a compatible group. The command selects resources that match the prototype from the group.

alertdefinition create - new alert definition

In this example, an new alert definition is created and assigned to members of a group that have the same resource type.

```
hqapi.sh alertdefinition create --group "Linux" --prototype "Linux" --name "My Alert" --metric
Availability --lessthan 1 --recoveryequals 1
```

This creates an alert definition for all Linux type resources in the group "Linux". The alert definition is named "My Alert" and the condition is set to "If Availability < 100" and also a matching recovery alert for when Availability = 100%.

HQApi alert command 6

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

The `alert` command has options you can use to list information for fired alerts, to fix an alert, to acknowledge an alert in escalation, and to delete an alert.

alert Command Options

alert list

The **alert list** command returns an `<AlertsResponse>` element, which contains alert data for fired alerts.
Command qualifiers:

Qualifier*	Description	Required
`--begin`	Use the `--begin` and `--end` qualifiers to specify the beginning and end of a time interval for which you wish to list fired alerts. Specify in epoch-millis.	no
`--count`	The number of alerts to return.	yes
`--end`	Use the `--begin` and `--end` qualifiers to specify the beginning and end of a time interval for which you wish to list fired alerts. Specify in epoch-millis.	no
`--groupId`	Use to limit the alerts returned to alerts in the specified groiup.	no
`--inEsc`	Use to limit the alerts returned to alerts in escalation.	no
`--notFixed`	Use to limit the alerts returned to alerts that have not been marked "fixed".	no
`--resourceId`	Use to limit the alerts returned to alerts for the specified resource	no
`--severity`	Use to restrict the alerts returned to those with a specified priority (as defined in the alert definition) or higher. Allowable values are: ■ 1 — returns alerts with priority level "Low" or higher, in effect, returns alerts regardless of priority. This is the default value. ■ 3 — returns only alerts whose priority is "High"	no

Structure of an AlertsResponse Element

The `AlertsResponse` object returned by the **alert list** command has this element structure:

```
<AlertsResponse>
<Alert>
<AlertActionLog>
```

where there is

- One `<Alert>` element for each fired alert.
- One `<AlertActionLog>` element for each action.

Attributes in an AlertsResponse

This table defines the attributes in a `AlertsResponse`.

Note that there are eight different types of `AlertCondition` elements, each with a different "type" value.

Element	Attribute	Description	Values
Alert	id	HQ internal ID for the alert.	
	name	Name of the alert definition.	
	alertDefinitionId	HQ internal ID for the alert definition.	
	resourceID	HQ internal ID for the resource to which the alert definition was assigned.	
	ctime	When the alert fired, in milliseconds since epoch.	
	fixed	Whether or not the alert has been "fixed".	true or false
	reason	Alert condition that fired the alert and the triggering value.	
AlertActionLog	timestamp		
	user	HQ user account under which the alert status event was performed.	
	detail	Short description of what was logged for the alert.	

List Examples

List Last n Alerts

This command requests the last 5 alerts that have fired regardless of acknowledgement of fix status.

```
hqapi.sh alert list --count=5
```

The following results are returned:

```
<AlertsResponse>
<Alert id="386023" name="RTA-up alert" alertDefinitionId="13337" resourceId="19023"
ctime="1278609900000" fixed="true" reason="If Availability &gt; 0.0% (actual value = 100.0%)"/>
<Alert id="386022" name="RTA-Down Alert" alertDefinitionId="13336" resourceId="19023"
ctime="1278609660000" fixed="true" reason="If Availability &lt; 100.0% (actual value = 0.0%)">
<AlertActionLog timestamp="1278609905762" user="admin" detail="Recovery Alert"/>
</Alert>
<Alert id="385922" name="RTA-up alert" alertDefinitionId="12020" resourceId="16595"
ctime="1278461340000" fixed="true" reason="If Availability &gt; 0.0% (actual value = 100.0%)"/>
<Alert id="385921" name="RTA-Down Alert" alertDefinitionId="12019" resourceId="16595"
ctime="1278461160000" fixed="true" reason="If Availability &lt; 100.0% (actual value = 0.0%)">
<AlertActionLog timestamp="1278461426260" user="admin" detail="Recovery Alert"/>
</Alert>
<Alert id="385824" name="RTA-up alert" alertDefinitionId="12020" resourceId="16595"
ctime="1278334680000" fixed="true" reason="If Availability &gt; 0.0% (actual value = 100.0%)"/>
</Alert>
```

List Last n Alerts in Escalation

This command requests the last 10 fired alerts that are in escalation.

```
hqapi.sh alert list --inEsc --count=10
```

List Last n Unfixed Alerts

This command requests the last 10 alerts that are not fixed.

```
hqapi.sh alert list --notFixed --count=10
```

List Alerts that Fired During an Interval

This command requests the last 10 alerts that fired between Tue, 03 Aug 2010 09:00:00 GMT and Tue, 03 Aug 2010 17:00:00 GMT.

```
hqapi.sh alert list --count=10 --begin=1280826000 end=1283207083
```

List Unfixed Alerts with Severity Level

This command requests the last 20 unfixed alerts with priority level "high".

```
sh bin/hqapi.sh alert list --count=10 --notFixed --severity=3
```

alert ack

The **alert ack** command acknowledge an alert in escalation.

Command qualifiers:

Qualifier	Description	Required
`--id`	The internal HQ ID for the alert.	yes
`--pause`	If specified, pause the Escalation for the specified number of milliseconds.	no
`--reason`	The reason for acknowledging the alert.	no

This command acknowledges the alert whose internal ID is 10254.

```
sh bin/hqapi.sh alert ack --id=10254
```

If the command was successful, a response like the following is returned:

```
Successfully acknowledged alert id 10254
```

alert fix

The **alert fix** command marks an alert "fixed".

Command qualifiers:

Qualifier	Description	Required
`--id`	The internal HQ ID for the alert.	yes
`--reason`	The reason for fixing the alert.	no

This command fixes the alert whose internal ID is 10254.

```
sh bin/hqapi.sh alert fix --id=10254
```

If the command was successful, a response like the following is returned:

```
Successfully fixed alert id 10254
```

alert delete

The **alert delete** command deletes a fired alert from history.

Command qualifiers:

Qualifier	Description	Required
`--id`	The internal HQ ID for the alert.	yes

This command deletes the alert whose internal ID is 10254.

```
sh bin/hqapi.sh alert delete --id=10254
```

If the command was successful, a response like the following is returned:

```
Successfully deleted alert id 10254
```

HQApi application command 7

View Source

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

An authorized user can use the **application** command to create, update, and delete HQ application definitions.

For information about applications in Hyperic, see "About Applications in Hyperic" in *vFabric Hyperic Administration Guide*.

application Command Options

application list

The **application list** command returns a list of applications in the HQ deployment and the services assigned to each.

Structure of an ApplicationResponse

The `ApplicationResponse` object returned by the **application list** command has this element structure:

```
<ApplicationResponse>
    <Application>
        <Resource.../>
    </Application>
</ApplicationResponse>
```

where there is:

- one `<Application>` element for each application.
- one `<Resource>` for each service assigned to the application.

Attributes in an ApplicationResponse

This table defines the attributes in a `ApplicationResponse`.

Element	Attribute	Description and Values	Required*
Application			
	bizContact	The name of the business user concerned with the application.	no
	opsContact	A contact name on operations.	no
	engContact	A contact name in engineering.	no
	location	Where the application runs, as applicable.	no
	description	Short description of the application.	no
	name	The name assigned to the application when it was configured in HQ.	yes
	id	The internal HQ ID for the application. This value is assigned by HQ when the application is first created. After an application has been created, you can use the **application list** command to determine its ID.	no
Resource	id	Internal HQ ID for the service.	no
	name	Name of the service.	no

description	Description of the service	no

Example - application list

In an HQ deployment with two applications, this command:

```
application list
```

returns results similar to the following following:

```
<ApplicationsResponse>
    <Application id="10002" name="JBaway" description="" location="" engContact="" opsContact=""
bizContact="">
        <Resource id="10825" name="Ms-MacBook-Pro-15.local HQ PostgreSQL 8.2 eam_action"
description=""/>
        <Resource id="10889" name="Ms-MacBook-Pro-15.local HQ PostgreSQL 8.2 eam_agent"
description=""/>
        <Resource id="10908" name="Ms-MacBook-Pro-15.local HQ PostgreSQL 8.2 eam_agent_type"
description=""/>
        <Resource id="10857" name="Ms-MacBook-Pro-15.local HQ PostgreSQL 8.2 eam_aiq_ip"
description=""/>
        <Resource id="10920" name="Ms-MacBook-Pro-15.local HQ PostgreSQL 8.2 eam_aiq_platform"
description=""/>
        <Resource id="10868" name="Ms-MacBook-Pro-15.local HQ PostgreSQL 8.2 eam_aiq_server"
description=""/>
        <Resource id="10860" name="Ms-MacBook-Pro-15.local HQ PostgreSQL 8.2 eam_aiq_service"
description=""/>
    </Application>
    <Application id="10003" name="Pops" description="SF" location="Mezz" engContact="B. Phelp"
opsContact="S. Townsend" bizContact="Jack Nok">
        <Resource id="10930" name="Marie-McGarrys-MacBook-Pro-15.local HQ Tomcat 6.0
/jbossws-context Tomcat 6.0 Webapp" description=""/>
        <Resource id="11023" name="Marie-McGarrys-MacBook-Pro-15.local JBoss 4.2 default HQ
Internals" description=""/>
        <Resource id="10988" name="Marie-McGarrys-MacBook-Pro-15.local JBoss 4.2 default
ActionManager Stateless Session EJB" description=""/>
        <Resource id="10942" name="Marie-McGarrys-MacBook-Pro-15.local JBoss 4.2 default
AgentManager Stateless Session EJB" description=""/>
        <Resource id="11028" name="Marie-McGarrys-MacBook-Pro-15.local JBoss 4.2 default
agentScheduleQueue JMS Destination" description=""/>
    </Application>
</ApplicationsResponse>
```

application sync

The **application sync** command can create new applications or update existing applications.

Command qualifiers:

Qualifier	Description	Required
`--batchSize`	The number of applications in each batch of applications to be committed to the database. By default, the HQ server writes application updates in batches of 10.	no

Example - Create Applications with application sync

1. Create an XML file that contains an `<ApplicationsResponse>`, which in turn contains an `<Application>` for each application you want to create, as shown above in Example - application list\ (see page 47). (You may find it useful to use **application list** to list an existing application, and copy the XML to an editor as a starting point.)
   - **Do not** specify an `id` for the applications to be created.
   - Save the file with an .xml extension.
2. Pipe the file to the **application sync** command, for example:

```
cat app.xml | bin/hqapi.sh application sync
```

3. A message like this indicates success:

```
Successfully synced 1 applications.
```

Example - Update Applications with application sync

1. Create an XML file that contains an `<ApplicationsResponse>`, which in turn contains an `<Application>` for each application you want to update. (You may find it useful to use **application list** to list existing applications, and copy the `<Application>` element for the applications you with to update to an editor as a starting point.)
2. Update the contents of each `<Application>` element as desired:
   - To add a service to the application, add a `<Resource>` element that defines it to the each `<Application>` element, making sure not to delete any existing `<Resource>` elements. (The **application sync** command updates the list of member services to match the sync file.)
   - To remove a service from the application, delete the corresponding `<resource>` element from the file.
   - When editing the attributes of a `<Resource>` element, be sure not to edit or delete the `id` attribute.
   - When editing the attributes of an `<Application>` element be sure not to edit or delete the `id` attribute. If `id` is not specified, a new application is created.
3. Save the file with an .xml extension.
4. Pipe the file to the **application sync** command, for example:

```
cat app.xml | bin/hqapi.sh application sync
```

5. A message like this indications the command execution was successful:

```
Successfully synced 1 applications.
```

application delete

The **application delete** command deletes an application from HQ inventory.

Note: Deleting an application does not delete the services that are assigned to the application.

Qualifier	Description
--id	Internal HQ ID for the application

Supply the ID of the application you wish to delete, for example:

```
hqapi.sh application delete --id=10002
```

A message like this indicates success:

```
Successfully deleted application id 10002
```

HQApi autodiscovery command 8

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

You can use the **autodiscovery** command to list platforms in the auto-discovery queue and import them to inventory. This is useful for approving a large number of auto-discovered items.

For information about auto-discovery in Hyperic, see "Discover and Import Resources to Inventory" in *vFabric Hyperic Administration Guide*.

autodiscovery Command Options

autodiscovery list

The **autodiscoverylist** command returns a `QueueResponse` object, which lists all platforms currently in the HQ auto-discover queue.

autodiscovery list Command Qualifiers

None.

Structure of a QueueResponse

The `QueueResponse` object returned by the **autodiscoverylist** command, contains one or more `AIPlatform` elements.

```
QueueResponse
     AIPlatform
```

Attributes in a QueueResponse

The `AIPlatform` element contains these attributes:

- fqdn - the fully qualified domain name of the platform
- name - the name of the platform
- id - the internal ID of the platform

autodiscovery approve

The **autodiscovery approve** command imports the some or all of the platforms in the auto-discovery queue to inventory. The servers and services discovered on approved platforms are also imported.

autodiscovery approve Command Qualifiers

When you run the **autodiscovery approve** command, you can optionally use the `--regex` qualifier to specify a regular expression to approve only the platforms in the queue whose name attribute matches the regular expression. Otherwise, all platforms in the queue will be added to inventory.

The format of the regex follows Java's "http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html" conventions.

Examples

autodiscovery list

In this example, there is a single platform in the queue.

```
bash-3.2$ ./bin/hqapi.sh autodiscovery list
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<QueueResponse>
    <Status>Success</Status>
    <AIPlatform fqdn="localhost" name="localhost" id="10008"/>
</QueueResponse>
```

autodiscovery approve

This command approves all platforms in the queue.

```
bash-3.2$ ./bin/hqapi.sh autodiscovery approve
Approving localhost
Approved 1 platforms.
bash-3.2$
```

autodiscovery approve with Regex Qualifier

This command approves platforms whose name matches the regular expression "local.*"

```
bash-3.2$ ./bin/hqapi.sh autodiscovery approve --regex="local.*"
Approving localhost
Approved 1 platforms.
```

HQApi control command 9

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

The **control** command can be used to list the control actions a resource instance supports, run a supported control action, and get information on the control actions that have been performed on a resource.

For an introduction to control actions see Resource Control in Hyperic.

control Command Options

This section defines command options for the **control** command.

control actions

The **control actions** command returns a list of supported control actions for a resource instance.

Command qualifiers:

Qualifier	Description	Required
`--resourceId`	Internal HQ ID for a resource. (See How to Find a Resource ID (see page 17).)	yes

The list of supported control actions is returned in plain text. The resource type and name is included in the output.

For an example, see control actions --resourceId (see page 55).

control history

The **control history** command returns the most recent control action that has been performed on a resource instance. Supply the following command qualifier:

Qualifier	Description	Required
`--resourceId`	Internal HQ ID for a resource. (See How to Find a Resource ID (see page 17).)	yes

The following information is returned, in plain text, for each control action performed on the resource:

- date and time
- `action` - the action that was performed.
- `dur` - how long the action took to execute, in milliseconds.
- `status` - the completion status of the control action, one of:
  - In Progress
  - Completed
  - Failed

For an example, see control history --resourceId (see page 56).

control execute

The **control execute** command runs a supported control action on a resource instance.

Command qualifiers:

Qualifier	Description	Required
`--resourceId`	Internal HQ ID for a resource. (See How to Find a Resource ID (see page 17).)	yes
`--action`	Control action to perform.	yes
`-- option`	For a custom control action only, can be used to supply values for command qualifiers required or supported by the control action. Common usage is to specify the `prefix` and `timeout` command qualifiers, which allow you to run the action as root, and specify a timeout for action execution, respectively. For information about custom control actions, see Configure a Custom Control Action. Note that the syntax for this qualifier is `-- Option1Name=Option1Value Option2Name=Option2Value` where: ▪ there is a space between the "`--`" and the the name=value pairs ▪ `OptionName` is the name of a supported qualifier ▪ each name=value pair supplied is separated by a space.	no

For examples, see:

- #control execute --action --resourceId (see page 56)
- #control execute --action --resourceId – option (see page 56)

control Command Examples

The sections below have examples of command-line execution of the **control** command.

control actions --resourceId

This command returns a list of the control actions supported by the resource whose internal HQ ID is "10786":

```
sh bin/hqapi.sh control actions --resourceId=10786
```

The results are:

```
Control actions for Dii-Xrr-MacBook-Pro-15.local HQ PostgreSQL 8.2
- Analyze
- ResetStatistics
- Vacuum
- VacuumAnalyze
```

The results include the resource type and FQDN for the resource, "HQ PostgreSQL 8.2" and "Dii-Xrr-MacBook-Pro-15.local", respectively.

control history --resourceId

This command returns a list of the control actions that have been performed on the resource whose internal HQ ID is "10786":

```
sh hqapi.sh control history --resourceId=10786
```

The command results are:

```
Control history for Dii-Xrr-MacBook-Pro-15.local HQ PostgreSQL 8.2
5/25/10 10:18 AM action=start dur=60004 status=Failed
5/24/10 12:59 PM action=VacuumAnalyze dur=1101 status=Completed
```

The results include the resource type and FQDN for the resource, "HQ PostgreSQL 8.2" and "Dii-Xrr-MacBook-Pro-15.local", respectively.

control execute --action --resourceId

This command executes the **start** command on the resource whose internal HQ ID is "10786":

```
sh bin/hqapi.sh control execute --action=vacuum --resourceId=10786
```

If the control action was successful executed, results like this are returned.

```
Ran action 'vacuum' on Dii-Xrr-MacBook-Pro-15.local HQ PostgreSQL 8.2
```

control execute --action --resourceId – option

This command executes the custom control action **run** on the resource whose internal HQ ID is "10786". `path` and `prefix` are qualifiers. Note that there is a space between the "--" and the name=value pairs that follow.

```

bin/hqapi.sh control execute --action=run --resourceId=10786 -- path=/usr/sbin/httpd prefix=sudo
```

HQApi dependency command 10

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

dependency Functionality

The **dependency** command allows you to query Hyperic inventory in terms of dependency relationships. You can:

- view and update the relationships between top level platforms and operating system platforms, and
- view parent-child relationships between platforms, servers, and services

In the Hyperic inventory model, there are two sorts of platform resources:

- Platforms that correspond to a host machine and its operating system have a "platform type" that identifies the host's operating system: "AIX", "HPUX", "Linux", and so on. For clarity, this kind of platform is referred to below as an *operating system platform* or a *dependent platform*. An operating system platform is the top resource in the "platform-server-service" hierarchy of resources that a Hyperic Agent discovers on a host.
- Platform types that correspond to is virtual or network resource are: "Network Host", "Network Device", "Cisco IOS", "Cisco PIXOS", "VMware V13 Host", and "Xen Host". This kind of platform is referred to below as *top level platform*. Unlike operating system platforms, which are auto-discovered and cannot be created manually, you explicitly add top level platforms to inventory.

Once you have created and correctly configured a top level platform (for instance, a Cisco switch) in Hyperic inventory, you can associate it with the operating system host platforms that depend on it. Do so extends the reach of hierarchical alerting beyond the platform-server-service hierarchy to network devices and virtual hosts - so that the health of a parent resource can be considered during alert processing for dependent resources.

The **dependency** API should be used only by advanced Hyperic administrators in consultation with Hyperic support or professional services. Under most circumstances, Hyperic recommends that administrators define network dependencies using the **Network and Host Dependency Manager**, available in the "Plugins" section of the **Administration** page in the vFabric Hyperic user interface.

For more information about platform dependencies in Hyperic see "Define Host Dependencies for Hierarchical Alerting" in *vFabric Hyperic Administration Guide.*

dependency Command Options

dependency list

The **dependency list** command returns a list top level platforms to which dependent platforms have been assigned.

dependency list Command Qualifiers

the **dependency list** command returns top level platforms with dependent platforms.

Qualifier	Description
none	Returns all top level platforms that have dependents.
`--children`	Returns the dependent resources for parents that match criteria specified using one or more of --prototype, --id, or --name
`--file=FileName`	Where `FileName` is the name of a file. If used, results are written to specified file. Otherwise, stdin is used.

`--id=Integer`	Where `Integer` is an integer value. Results are returned for the top level platform with the specified internal Hyperic ID.
`--name=regex`	Where `regex` is a regular expression. Limits results to the top level platforms whose name matches the specified regular expression. Can use with --prototype.
`--prototype=ResourceType`	Where `ResourceType` is a Hyperic resource type. Limits the results to top level platforms of specified resource type. Can use with --name.
`--relation=containment`	Use with --children and --id to return the immediate children of the specified resource, which may be a platform, server, or service.

Structure of ResourcesResponse

The `ResourcesResponse` object returned by the **dependency list** command has this element structure:

```
ResourcesResponse
     Resource
          ResourcePrototype
          Agent
          Ip
          ResourceInfo
```

where there are:

- 1 or more `Resource` elements, each with one instance of each of the following elements
- `ResourcePrototype`
- `Agent`
- `Ip`
- `ResourceInfo`

Atributes of ResourcesResponse

Element	Attribute	Description	Example Value
Resource			
	description		"Cisco IOS Software, C2960 Software (C2960-LANBASE-M), Version 12.2(25)FX"
	name		"cisco ios platform"
	id		"13797"
ResourcePrototype			
	name		"Cisco IOS"
	id		"10004"
Agent			
	unidirectional		"false"
	version		"4.2.0-EE"
	port		"2144"

	address	"10.2.0.108"
	id	"10029"
Ip		
	mac	""
	netmask	""
	address	"10.2.0.108"
ResourceInfo		
	value	"cisco ios test"
	key	"fqdn"

Structure of ResourceEdgesResponse element

The `ResourceEdgesResponse` object returned by the **dependency list** and the --children qualifier has this element structure:

```
ResourceEdgesResponse
    ResourceEdge
        ResourceFrom
        ResourceTo
```

where there are:

- 1 or more `ResourcEdge` elements, each with
  - one `ResourceFrom` and
  - one or more `ResourceTo` elements

dependency list Examples

dependency list

List all top level resources in a network hierarchy:

```
hqapi.sh dependency list

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesResponse>
<Status>Success</Status>
<Resource description="Cisco IOS Software, C2960 Software (C2960-LANBASE-M), Version 12.2(25)FX"
name="cisco ios platform" id="13797">
<ResourcePrototype name="Cisco IOS" id="10004"/>
<Agent unidirectional="false" version="4.2.0-EE" port="2144" address="10.2.0.108" id="10029"/>
<Ip mac="" netmask="" address="10.2.0.108"/>
<ResourceInfo value="cisco ios test" key="fqdn"/>
</Resource>
<Resource description="" name="xen 2 platform" id="13918">
<ResourcePrototype name="Xen Host" id="10042"/>
<Agent unidirectional="false" version="4.2.0-EE" port="2144" address="10.2.0.108" id="10029"/>
<Ip mac="" netmask="" address="10.0.0.137"/>
<ResourceInfo value="xen-02" key="fqdn"/>
</Resource>
</ResourcesResponse>
```

dependency list --prototype

List top level resources in a network hierarchy by prototype:

```
hqapi.sh dependency list --prototype="Xen Host"

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesResponse>
<Status>Success</Status>
<Resource description="" name="xen 2 platform" id="13918">
<ResourcePrototype name="Xen Host" id="10042"/>
<Agent unidirectional="false" version="4.2.0-EE" port="2144" address="10.2.0.108" id="10029"/>
<Ip mac="" netmask="" address="10.0.0.137"/>
<ResourceInfo value="xen-02" key="fqdn"/>
</Resource>
</ResourcesResponse>
```

Dependency List --name

List top level resources in a network hierarchy by regex name:

```
hqapi.sh dependency list --name=cisco

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesResponse>
<Status>Success</Status>
<Resource description="Cisco IOS Software, C2960 Software (C2960-LANBASE-M), Version 12.2(25)FX"
name="cisco ios platform" id="13797">
<ResourcePrototype name="Cisco IOS" id="10004"/>
<Agent unidirectional="false" version="4.2.0-EE" port="2144" address="10.2.0.108" id="10029"/>
<Ip mac="" netmask="" address="10.2.0.108"/>
<ResourceInfo value="cisco ios test" key="fqdn"/>
</Resource>
</ResourcesResponse>
```

dependency list --prototype --name

List top level resources in a network hierarchy by prototype and regex name:

```
/hqapi.sh dependency list --prototype="Xen Host" --name="Xen 2"

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesResponse>
<Status>Success</Status>
<Resource description="" name="xen 2 platform" id="13918">
<ResourcePrototype name="Xen Host" id="10042"/>
<Agent unidirectional="false" version="4.2.0-EE" port="2144" address="10.2.0.108" id="10029"/>
<Ip mac="" netmask="" address="10.0.0.137"/>
<ResourceInfo value="xen-02" key="fqdn"/>
</Resource>
</ResourcesResponse>
```

dependency list --children --id

List dependent resources under a top level resource in a network hierarchy:

```
hqapi.sh dependency list --children --id=13797

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourceEdgesResponse>
<Status>Success</Status>
<ResourceEdge distance="1" relation="network">
<ResourceFrom>
<Resource name="cisco ios platform" id="13797"/>
</ResourceFrom>
<ResourceTo>
<Resource name="kparikh-T60-x" id="13852"/>
<Resource name="patrick-nguyens-macbook-pro.local" id="10611"/>
</ResourceTo>
</ResourceEdge>
</ResourceEdgesResponse>
```

dependency list --children --prototype

List dependent resources (and their top level resource) by prototype in a network hierarchy:

```
patrick-nguyens-macbook-pro:bin pnguyen$ ./hqapi.sh dependency list --children --prototype=Win32
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourceEdgesResponse>
<Status>Success</Status>
<ResourceEdge distance="1" relation="network">
<ResourceFrom>
<Resource name="cisco ios platform" id="13797"/>
</ResourceFrom>
<ResourceTo>
<Resource name="kparikh-T60-x" id="13852"/>
</ResourceTo>
</ResourceEdge>
<ResourceEdge distance="1" relation="network">
<ResourceFrom>
<Resource name="cisco ios platform" id="13797"/>
</ResourceFrom>
<ResourceTo>
<Resource name="patricktest" id="13925"/>
</ResourceTo>
</ResourceEdge>
</ResourceEdgesResponse>
```

dependency list --children --name

List dependent resources (and their top level resource) by regex name in a network hierarchy:

```
hqapi.sh dependency list --children --name=patrick

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourceEdgesResponse>
<Status>Success</Status>
<ResourceEdge distance="1" relation="network">
<ResourceFrom>
<Resource name="cisco ios platform" id="13797"/>
</ResourceFrom>
<ResourceTo>
<Resource name="patrick-nguyens-macbook-pro.local" id="10611"/>
</ResourceTo>
</ResourceEdge>
<ResourceEdge distance="1" relation="network">
<ResourceFrom>
<Resource name="xen 2 platform" id="13918"/>
</ResourceFrom>
<ResourceTo>
<Resource name="patricktest" id="13925"/>
</ResourceTo>
</ResourceEdge>
</ResourceEdgesResponse>
```

dependency list --children --prototype --name

```
To List dependent resources (and their top level resource) by prototype and regex name in a network
hierarchy:

hqapi.sh dependency list --children --prototype=Win32 --name=pat
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourceEdgesResponse>
<Status>Success</Status>
<ResourceEdge distance="1" relation="network">
<ResourceFrom>
<Resource name="cisco ios platform" id="13797"/>
</ResourceFrom>
<ResourceTo>
<Resource name="patricktest" id="13925"/>
</ResourceTo>
</ResourceEdge>
</ResourceEdgesResponse>
```

Dependency List --children --id --relation=containment

```
List the immediate child resources under a resource based on the Hyperic inventory model
(platform/server/services):

hqapi.sh dependency list --children --id=10611 --relation=containment
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourceEdgesResponse>
<Status>Success</Status>
<ResourceEdge distance="1" relation="containment">
<ResourceFrom>
<Resource name="patrick-nguyens-macbook-pro.local" id="10611"/>
</ResourceFrom>
<ResourceTo>
<Resource name="patrick-nguyens-macbook-pro.local HQ Agent 4.2.0-EE" id="10613"/>
<Resource name="patrick-nguyens-macbook-pro.local HQ JBoss 4.x" id="10617"/>
<Resource name="patrick-nguyens-macbook-pro.local MacOSX FileServer" id="10616"/>
<Resource name="patrick-nguyens-macbook-pro.local MacOSX NetworkServer" id="10612"/>
<Resource name="patrick-nguyens-macbook-pro.local MacOSX ProcessServer" id="10615"/>
<Resource name="patrick-nguyens-macbook-pro.local Net Services" id="10614"/>
<Resource name="patrick-nguyens-macbook-pro.local Tomcat 5.5" id="10618"/>
<Resource name="patrick-nguyens-macbook-pro.local Tomcat 6.0" id="10872"/>
</ResourceTo>
</ResourceEdge>
</ResourceEdgesResponse>
```

dependency sync

The **dependency sync** command creates or updates the dependency relationships between a specified top level platform and the operating system host platforms the depend on it, from the `ResourceEdge` elements in an XML file.

dependency sync Command Qualifiers

Qualifier*	Description
`--all`	Defines a new network dependence hierarchy - the dependency relationships between a top level platform and the operating system host platforms that depend on it. If a network dependence hierarchy is already defined for a top level platform, `dependency sync -all` will delete the previous relationships and create new ones.
`--add`	Adds one or more additional dependencies to the dependence hierarchy for a top level platform. Use this command qualifier to add dependent platforms, without over-writing or deleting existing dependence relationships the top level platform has with other children.
`--remove`	Removes one or more dependencies from the dependence hierarchy for a top level platform.

dependency delete

The **dependency delete** command deletes all dependency relationships between a specified top level platform and the operating system host platforms that depend on it, from the `ResourceEdge` elements in an XML file.

dependency delete Command Qualifiers

Qualifier	Description

--id=IntegerValue	This required qualifier identifies, by its internal Hyperic ID, the top level platform whose dependency relationships to children will be deleted.
--relation	The resource relationship.

dependency select

The **dependency select** command lists top level platforms that are not associated with a network hierarchy. (Top level platforms without relationships defined to dependent platforms.)

dependency select Command Qualifiers

The **dependency select** command returns platforms that don't have network dependency relationships defined. You can use **select** to return top level platforms without dependent children, or operating system platforms without parents.

You can supply one or more of the following qualifiers to limit what top level platforms are returned.

Qualifier	Description
none	Returns all top level platforms without dependents.
`--children`	Returns operating system host platforms, rather than top level platforms. Returns the platforms without a parent.
`--file FileName`	Where `FileName` is the name of a file.Results are output to the file specified, instead of stdin.
`--name` *regex*	Limits the platforms returned to those whose name matches the specified regular expression.
`--prototype`	Limits the platforms returned to those of the specified resource type.
`--relation`	The resource relationship.

HQApi escalation command 11

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

About Escalations

- Introduction to Escalations describes escalations and how they work.
- Configure and Manage Escalations has instructions for setting up and managing escalations from the Hyperic user interface; it includes useful information about the components of an escalation.

Functionality

The **escalation** command can be used to list and update Escalations. It is useful for:

- Reviewing all escalations defined in Hyperic.
- Updating multiple escalations in a single step.

escalation Command Options

escalation list

The **escalation list** returns an `EscalationsResponse` element that defines all escalations defined in Hyperic.

escalation list Command Qualifiers

None.

Structure of an EscalationsResponse

The `EscalationsResponse` returned by the **escalation list** command has this element structure.

```
EscalationsResponse
Escalation
Action
```

where there are:

- 1 or more `Escalation` elements, each with
- 1 or more `Action` elements

Attributes in an EscalationsResponse

This table defines the attributes in a `EscalationsResponse`.

Note that there are three different types of `Action` elements, one for email, one for syslog actions, one to suppress the alert. In the `Action` element, the different types are distinguished by the value of the actionType attribute; that remaining attributes in the element vary by action type.

Element	Attribute	Values
`Escalation`		
	repeat	true false
	notifyAll	true false
	maxPauseTime	
	pauseAllowed	true false
	description	
	name	
	id	
`Action`		

	sms	true false
	notifyType	users - corresponds to "Notify HQ Users" roles - corresponds to "Notify Roles" email - corresponds to "Notify Other Recipients"
	actionType	"EmailAction"
	wait	
	id	
`Action`		
	syslogVersion	
	syslogProduct	
	syslogMeta	
	actionType	"SyslogAction"
	wait	
	id	
`Action`		
	actionType	"NoOpAction" Corresponds to the "Suppress Alerts" action type.
	wait	
	id	
`Action`		
	actionType	"SnmpAction" Corresponds to "SNMP Notification" action type, supported if SNMP is enabled.
	wait	
	id	

escalation sync

The **escalation sync** command updates escalations defined in Hyperic with the attributes defined in the `EscalationsResponse` element. You cannot create new escalations with **escalation sync**.

When you add Actions to an escalation, the order in which you specify the actions is the order in which the actions will be performed.

Examples

escalation list

In this example, two escalations are returned.

The first escalation, whose name is "My Escalation", has three email actions, one to the Hyperic admin user, one to all Hyperic users with the Hyperic Super User Role role, and one to "ops@bar.com".

The second role is Hyperic's built-in escalation, which has no actions.

```
$ ./bin/hqapi.sh escalation list
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<EscalationsResponse>
<Status>Success</Status>
<Escalation repeat="true" notifyAll="false" maxPauseTime="172800000" pauseAllowed="true"
description="" name="My Escalation" id="10044">
<Action actionType="NoOpAction" wait="0" id="10495"/>
<Action sms="false" notifyType="users" actionType="EmailAction" wait="2700000" id="10496">
<Notify name="hqadmin"/>
</Action>
<Action sms="false" notifyType="roles" actionType="EmailAction" wait="300000" id="10497">
<Notify name="Super User Role"/>
</Action>
<Action sms="false" notifyType="email" actionType="EmailAction" wait="3600000" id="10498">
<Notify name="ops@bar.com"/>
</Action>
<Action syslogVersion="4.1" syslogProduct="Hyperic HQ" syslogMeta="Error Text"
actionType="SyslogAction" wait="0" id="10573"/>
</Escalation>

<Escalation repeat="false" notifyAll="false" maxPauseTime="300000" pauseAllowed="false"
description="This is an Escalation Scheme created by HQ that performs no actions" name="Default
Escalation" id="100"/>
<Escalation repeat="false" notifyAll="false" maxPauseTime="300000" pauseAllowed="false"
description="" name="Support Escalation - HIGH" id="10020"/>
<Escalation repeat="false" notifyAll="false" maxPauseTime="300000" pauseAllowed="false"
description="" name="Support Escalation - LOW" id="10022"/>
<Escalation repeat="false" notifyAll="false" maxPauseTime="300000" pauseAllowed="false"
description="" name="Support Escalation - MEDIUM" id="10021"/>
</EscalationsResponse>
$
```

escalation sync

In this example, the first command writes the escalations in Hyperic to a file called `escalations.xml`. The second command pipes `escalations.xml` to the sync method to update the escalations in Hyperic.

```
$ ./bin/hqapi.sh escalation list > escalations.xml
...
...
$ cat escalations.xml | ./bin/hqapi.sh escalation sync
Successfully synced 5 escalations.
bash-3.2$
```

HQApi event command 12

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

The **event** command returns data about events - for fired alerts, for resource control actions that were performed, and (if you have so configured) for log and configuration change events that meet the tracking criteria defined for the resource.

For information about Hyperic's event tracking functionality, see Log and Configuration Event Tracking.

You can list the events for either a particular resource instance, or for all resources.

event Command Options

This section defines command options.

event list

The **event list** command returns events for a resource instance, or for all resource instances.

Command qualifiers:

Qualifier	Description	Required
`--resourceId`	Internal HQ ID for a resource. If supplied, only events for that resource are returned. (See How to Find a Resource ID (see page 17).)	no

Structure of an EventsResponse

The `<EventsResponse>` object returned by the **event list** command has this element structure:

```
<EventsResponse>
    <Event.../>
</EventsResponse>
```

where there is one `<event>` element for each event returned.

Attributes in an EventResponse Element

This table defines the attributes in a `<EventsResponse>`.

Element	Attribute	Description and Values
`<EventsResponse>`		
	`resourceID`	Internal HQ ID for the resource. (See How to Find a Resource ID (see page 17).)
	`status`	Indicates the nature of the event. If the event is: ▪ alert event - `status` will be `ALR` ▪ control action event - `status` will be `Completed` or `Failed` ▪ log event - status will be one of: `"INF"`, `"WRN"`, `"ERR"`, or `"DBG"` ▪ configuration event - status will be `"INF"`

`user`	The `user` attribute provides different information, depending on event type: - alert event - `user` is the name of the associated alert definition. - control action event - `user` is the HQ user account under which the action was performed. - log event - `user` is the path/name of the tracked log file (or, on Windows, the name of Windows log) to which the tracked message was written. - configuration event - `user` is the file path/name of the tracked file.
`ctime`	When the event occurred, in timestamp in milliseconds from epoch.
`type`	Type of event object. - alert event - org.hyperic.hq.events.AlertFiredEvent - control action event - org.hyperic.hq.control.ControlEvent - log event - org.hyperic.hq.measurement.shared.ResourceLogEvent - configuration event - org.hyperic.hq.measurement.shared.ConfigChangedEvent

`detail`	The `detail` attribute provides different information, depending on event type.
	- alert event - value varies by the type of alert condition, it could be one of:
	  - the metric value that satisfied a metric-based condition.
	  - the new value of the inventory property that the alert condition was based on.
	  - the control action that was performed, for an alert condition based on a control action.
	- log event - `detail` will contain both:
	  - the full path to the file, or for windows, name of event log, and
	  - the message written to the log.
	- configuration event - `detail` will contain:
	  - when the tracked file was changed and when the Inode for the file was changed.
	  - system groups that have access to the file.
	  - size of the tracked file in bytes before and after modification.
	  - Inode for tracked file before and after the change.

event Command Examples

event list --resourceId

This command returns events for the resource whose internal HQ ID is 10781:

```
sh bin/hqapi.sh event list --resourceId=10781
```

The command results are:

```
<EventsResponse>
<Status>Success</Status>
<Event resourceId="10781" status="INF"
user="/Applications/hqEE43-1433/server-4.3.0-EE/hq-engine/server/default/../../../logs/server.log"
ctime="1275056690366" type="org.hyperic.hq.measurement.shared.ResourceLogEvent"
detail="/Applications/hqEE43-1433/server-4.3.0-EE/hq-engine/server/default/../../../logs/server.log:
2010-05-28 07:24:11,917 INFO [Scheduler-2] [org.hyperic.hq.auth.shared.SessionManager@51] done
cleaning up expired sessions (0 expired sessions)"/>
...
<Event resourceId="10781" status="WRN"
user="/Applications/hqEE43-1433/server-4.3.0-EE/hq-engine/server/default/../../../logs/server.log"
ctime="1275060530465" type="org.hyperic.hq.measurement.shared.ResourceLogEvent"
detail="/Applications/hqEE43-1433/server-4.3.0-EE/hq-engine/server/default/../../../logs/server.log:
2010-05-28 08:28:33,142 WARN [Thread-4046]
[org.hyperic.hq.bizapp.server.session.LatherDispatcher@217] Unauthorized agent from 192.168.0.7
denied"/>
</EventsResponse>
```

event list

This command lists all events for all resources:

```
sh bin/hqapi.sh event list
```

The results returned are that same as those in the example above, but include events for all resources.

Example Event Element for Different Event Types

Event Element for an Alert Event

This is the result returned for an alert event:

```
Event resourceId="10781" status="ALR" user="ThreadCount" ctime="1274811000000"
type="org.hyperic.hq.events.AlertFiredEvent" detail="130.0"/>
```

These results indicate that:

- An alert whose definition is named "ThreadCount" fired.
- The triggering metric value was "130.0".
- The event type is "org.hyperic.hq.events.AlertFiredEvent".

Event Element for an Control Action Event

This is the result returned for a control action event:

```
<Event resourceId="10786" status="Completed" user="hqadmin" ctime="1274900999202"
type="org.hyperic.hq.control.ControlEvent" detail="Vacuum"/>
```

These results indicate that:

- The "Vacuum" control action was initiated and completed.
- The HQ user that ran the control action was "hqadmin".
- The event type is "org.hyperic.hq.control.ControlEvent".

Event Element for an Log Event

This is the result returned for a log event:

```
<Event resourceId="10781" status="INF"
user="/Applications/hqEE43-1433/server-4.3.0-EE/hq-engine/server/default/../../../logs/server.log"
ctime="1274820000805" type="org.hyperic.hq.measurement.shared.ResourceLogEvent"
detail="/Applications/hqEE43-1433/server-4.3.0-EE/hq-engine/server/default/../../../logs/server.log:
2010-05-25 13:36:28,919 INFO \[http-0.0.0.0-7080-5|http-0.0.0.0-7080-5\] \[\] 192.168.0.15:2144 \->
track:trackAdd"/>
```

These results indicate that:

- A message of level "INF" was written to the `server.log` file.
- The message was "="/Applications/hqEE43-1433/server-4.3.0-EE/hq-engine/server/default/../../../logs/server.log: 2010-05-25 13:36:28,919 INFO [http-0.0.0.0-7080-5|~mmcgarry:http-0.0.0.0-7080-5] [] 192.168.0.15:2144 -> track:trackAdd"/>.
- The event type is "org.hyperic.hq.measurement.shared.ResourceLogEvent".

Event Element for an Configuration Event

This is the result returned for a configuration event:

```
<Event resourceId="10781" status="INF"
user="/Applications/hqEE43-1433/server-4.3.0-EE/hq-engine/server/default/conf/jboss-service.xml"
ctime="1274821140999" type="org.hyperic.hq.measurement.shared.ConfigChangedEvent" detail="{Mtime:
May 25 12:58|May 25 13:58}{Gid: 80|20}{Size: 34293|34306}{Inode: 3169904|3203869}"/>
```

These results indicate that:

- The `jboss-service.xml` file was modified on May 25 at 12:58, the Inode for the file was updated on May 25 at 13:58.
- The group IDs of the system groups that have access to the file are "80" and 20".
- The file size changed from 34293 to 34306 bytes.
- The file Inode changed from 3169904 to 3203869.
- The event type is "org.hyperic.hq.measurement.shared.ConfigChangedEvent".

HQApi groovyshell command 13

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

---

groovshell Functionality

The **groovyshell** command allows you to run a groovy script from the command shell.

groovyshell Command Qualifiers

The `groovyshell` command has one qualifier, which is required: `run`.

HQApi group command 14

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

The **group** command has options to output resource group definitions, create or update groups, and delete groups. This is useful if you want to:

- Review current resource group configurations.
- Make multiple changes to the configuration of groups in a single step.

For information about groups in Hyperic, see About Groups in Hyperic in *vFabric Hyperic Administration*.

group Command Options

group list

The **group list** command returns a `GroupsResponse` element, which contains a contains a `Group` element (that defines the attributes of a group) for each group returned.

See Resources, Resource Types and Inventory Types.

group list Command Qualifiers

You can limit the groups returned by the **list** option with these qualifiers.

Qualifier	Description
--compatible	Only compatible groups are returned.
--id	The group with the specified ID is returned.
--mixed	Only mixed groups are returned.
--name	The group with the specified name is returned.

Structure of a GroupsResponse

The `GroupsResponse` element returned by the **group list --compatible** command has this element structure.

```
GroupsResponse
Group
ResourcePrototype
Resource
Role
```

where there is:

- 1 `group` element for each group returned
- 0 or 1 `ResourcePrototype` elements; it is only present for a compatible group, in which case it specifies the resource type contained by the group.
- 1 `Resource` element for each member of the group.
- (In vFabric Hyperic only) 1 `Role` element for each role that has access to the group.

Attributes in a GroupsResponse

Element	Attribute	Required to Create a Group?	Description	Allowable Values
`Group`				
	resourceID			
	id	no - do not supply		
	name	yes		
	description	no		
	location	no		
`ResourcePrototype`				
	id	yes - for a compatible group		
	name	no		
`Resource`				
	id	yes		
	name	no		
`Role`				
	id	yes		
	name	no		
	description	no		

group sync

The **group sync** command creates or updates group definitions in Hyperic using the content of a `GroupsResponse` element.

This is how **group sync** works:

- You can supply the values to sync a group in an XML file or via command line arguments; the latter capability is a functional replacement for the Hyperic Mass plugin, which is deprecated.
- All attributes in the `Group` element, except `id`, can be updated. (When you create a new group, Hyperic assigns the `id` .)
- The `ResourcePrototype` cannot be changed once the group is created. Attempts to update this will result in a not supported error. **group sync** only adds resources to a group, unless you use the `--deleteMissing` qualifier.
- The roles assigned to the group are completely updated based on the `Role` elements present. If you delete a `Role` element, the **sync** will remove that role from the group.

group sync Command Qualifiers

This table defines the command qualifiers you can use to define or update a group from the command line in Hyperic 4.3 and later.

Qualifier	Description
`--addRole`	Use in conjunction with the `name` qualifier. Use `addRole` to add a role to the group whose name is defined by `name`.
`--children`	For filtering group membership. Causes the children of the resources that match other specified filters to be added to the group. The new or updated group will contain the resources that match the membership filter criteria, and also the children of those resources.
`--clear`	For removing all resources from an existing group. The updated group will have no resources.
`--clearRoles`	Use in conjunction with the `name` qualifier. Use `clearRoles` to remove all roles from the group whose name is defined by `name`
`--compatible`	If specified, attempt to make the group compatible
`--delete`	For removing selected resources from an existing group. Resources that match the filter criteria will be removed from the group.
`--deleteMissing`	This switch controls how **sync** behaves when run on an existing group: the updated group will contain only resources that match the specified filter criteria — existing group members that do not match the filter criteria will be removed from the group. Use `deleteMissing` if you want to to completely re-specify the rules for membership in the group. If instead, you wish to simply add additional resources to the group, do not use `deleteMissing`
`--description`	Description of the group to create or update.
`--name`	Name of the group to create or update.
`--platform`	For filtering group membership. Limits the resources in the group to resources on the specified platform.
`--prototype`	For filtering group membership. Limits the resources in the group to resources of the specified resource type - as identified by Hyperic's internal ID for the resource type.
`--regex`	For filtering group membership. Limits the resources in the group to resources whose resource type matches the regex
`--removeRole`	Use in conjunction with the `name` qualifier. Use `removeRole` to remove a role from the group whose name is defined by `name`

group delete

The **delete** option deletes a single specified group from the Hyperic database.

group delete Command Qualifiers

You must identify the group to be deleted using the `--id` qualifier to specify the group's internal ID.

Examples

group list

The `GroupsResponse` element below defines two groups. The "All Servers" group is a mixed group, and is associated with the the "MgmtAdmin" role. (The roles feature is available only in vFabric Hyperic.

"File Mounts" is a compatible group, as indicated by the existence of the `ResourcePrototype` element.

```
bash-3.2$ ./bin/hqapi.sh group list
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GroupsResponse>
<Status>Success</Status>
<Group location="" description="" name="All Servers" id="10001">
<Resource name="localhost HQ Agent 4.1.0-EE" id="10673"/>
<Resource name="localhost HQ JBoss 4.x" id="10667"/>
<Resource name="localhost PostgreSQL 8.2.5" id="10671"/>
<Resource name="localhost Tomcat 6.0" id="10662"/>
<Role name="MgmtAdmin" id="4"/>
</Group>
<Group location="" description="" name="File Mounts" id="10002">
<ResourcePrototype name="FileServer Mount" id="10048"/>
<Resource name="localhost MacOSX File System /dev/disk0s2 mounted on / (local/hfs)" id="10692"/>
<Resource name="localhost MacOSX File System /dev/disk1s3 mounted on /Volumes/hyperic (local/hfs)"
id="10694"/>
<Resource name="localhost MacOSX File System /dev/disk1s5 mounted on /Volumes/spare (local/hfs)"
id="10693"/>
</Group>
</GroupsResponse>
```

group delete

In this example the group whose internal id is 10002 is deleted.

```
$ ./bin/hqapi.sh group delete --id=10002
Successfully deleted group id 10002
$
```

group sync via XML

In this example, the first command writes the all groups in Hyperic to a file called `groups.xml`. The second command sends the `groups.xml` to the `sync` method to update the groups in Hyperic.

```
$ ./bin/hqapi.sh group list > groups.xml
...
...
$ cat groups.xml | ./bin/hqapi.sh group sync
Successfully synced 2 groups.
```

For more information about supplying values to **sync** using an XML file, see

Note: When you use **sync** to create a new item, do not specify the `id` attribute for the new item. See sync (see page 15) for more information.

group sync - Create or Update a Compatible Group from Command Line

The command below creates or updates a compatible group named "Disks - Web", adding FileServer Mounts whose name match the regular expression `.web.` and deleting any resources in the group that don't match criteria specified by `prototype` and `regex`. Because the `--deleteMissing` qualifier is included, after running this command the *only* resources in the group will be FileServer Mounts whose names match the regular expression `.web.`. If the group already exists, any resources that do not match the filter criteria are removed.

```
group sync --name="Disks - Web" --prototype="FileServer Mount" --regex=".*web.*"
--description="All Web Disks" --deleteMissing --compatible
```

group sync - Create or Update a Mixed Group from Command Line

To create a mixed group from the command line, run sync multiple times for the same group, with different filter criteria. Be sure not to specify `--deleteMissing` and `--compatible` when creating and updating a mixed group.

This command creates a mixed group that contains all the resources on a platform.

```
group sync --name=Targol --platform=demo2.hypo.net --description=my group
```

To add more resources to the mixed group, run **group sync** again with appropriate filter criteria.

HQApi maintenance command 15

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

You can use the **maintenance** command to schedule downtime for a resource or a group of resources — it is the command line equivalent of vFabric Hyperic's **Schedule Downtime** feature, available in the **Tools** menu when a resource or a group is selected. At the start of the downtime period, currently active alert definitions for the specified resource(s) in the group are disabled. At the end of the period, those alert definitions are re-enabled.

Note: Only a Hyperic user with the Super User role may schedule downtime.

maintenance Command Options

maintenance schedule

The **maintenance schedule** command defines a period of downtime for a specified resource or resource group. Command qualifiers include:

Qualifier	Description	Required
--groupId	Specifies the internal ID for the group for which maintenance is being scheduled, for example, `--groupId="10007"`	Required if `--resourceId` is not specified.
--resourceId	Specifies the internal ID of the resource for which maintenance is being scheduled.	Required if `--groupId` is not specified.
--start	Specifies the start date and time for the downtime window, for example, `start="8/20/09 3:00 PM"`	yes
--end	Specifies the end date and time for the downtime window, for example, `end="8/20/09 4:00 PM"`	yes

maintenance unschedule

The **maintenance unschedule** command removes the currently scheduled downtime for a specified resource or resource group.

Qualifier	Description	Required
--groupId	Specifies the internal ID for the group, for example, `--groupId="10007"`	Required if `--resourceId` is not specified.
--resourceId	Specifies the internal ID of the resource for which maintenance is being unscheduled.	Required if `--groupId` is not specified.

maintenance get

The **maintenance get** command returns a downtime schedule for a resource, a resource group, or all resources and resource groups. You must supply one of the following command qualifiers:

Qualifier	Description

--all	Returns all maintenance schedulesSpecifies the internal ID for the resource
--groupId	Specifies the internal ID for the group, for example, `--groupId="10007"`
--resourceId	Specifies the internal ID for the resource

Results are returned in this form:

```
Maintenance schedule for group 10007
State: new
Start Time: 8/20/09 3:00 PM
End Time: 8/20/09 4:30 PM
```

Examples

maintenance schedule

```
$ sh hqapi.sh maintenance schedule --groupId="10007" --start="8/20/09 3:00 PM" --end="8/20/09 4:30
PM"
Maintenance scheduled for group 10007
```

maintenance get

```
$ sh hqapi.sh maintenance get --groupId="10007"
Maintenance scheudle for group 10007
State: new
Start Time: 8/20/09 3:00 PM
End Time: 8/20/09 4:30 PM
```

maintenance unschedule

```
$ sh hqapi.sh maintenance unschedule --groupId="10007"
Maintenance for group 10007 unscheduled.
```

Script for Scheduling Recurring Maintenance

If you need to perform an API function multiple times to automate a task, you can call it programmatically rather than using its command line interface. You can use a program to directly invoke the API's Java methods directly or invoke GET/POST methods against the RESTful web service that exposes the API on the Hyperic Server.

The script listing in Sample Script Source (see page 87) is an example of the latter method. This script accesses the web service that exposes the `maintenance` API to set a maintenance window for a group.

Perl was chosen for the example for its multi-platform support and flexibility in working with XML. Other scripting languages might also be used to solve the same problem.

Using group_maintenance_api.pl

You can use the `group_maintenance_api.pl` script to schedule, check, and delete a maintenance schedule for a group of resources.

Set Environment Variables

Edit `group_maintenance_api.pl` to set the values of the following variables to match your environment.

- `$hq_server`
- `$username`
- `$password`

Schedule Maintenance for a Group

The command syntax to schedule a maintenance interval for a group of resources is:

```
group_maintenance_api.pl set "GroupName" "Start" "End"
```

where:

- `GroupName` is the name of the group as it appears in the Hyperic user interface.
- `Start` is a time, or a date and time in standard date format. If you set `Start` to a time, without specifying a date, the date defaults to the date following the the current one.
- `End` is a time, or a date and time in standard date form. If you set `Start` to a time, without specifying a date, the date defaults to the date following the the current one.

The following invocation, which specifies date **and** time for the start and end of the maintenance interval, schedules a maintenance window from 3 PM to 4 PM on 10/20/09, for the resources in the group named "HQ Agents":

```
group_maintenance_api.pl set "HQ Agents" "10/20/09 3:00 PM" "10/20/09 4:00 PM"
```

When the script is run with **only** time for the the start and end of the maintenance interval, the maintenance window is scheduled for 3 PM to 4 PM on the following day:

```
group_maintenance_api.pl set "HQ Agents" "3:00 PM" "4:00 PM"
```

Check Maintenance Schedule

The command syntax to check the currently scheduled maintenance interval for a group of resources is:

```
group_maintenance_api.pl get "GroupName"
```

For example, this command:

```
group_maintenance_api.pl get "HQ Agents"
```

returns:

```
Status: Success
Start time: Wed Oct 20 15:00:00 2009
End time: Wed Oct 28 16:00:00 2009
```

Cancel a Maintenance Schedule

The command syntax to cancel the currently scheduled maintenance interval for a group of resources is:

`group_maintenance_api.pl unset "GroupName"`

Where `GroupName` is the name of a group.

Set Up a Recurring Maintenance Schedule

The examples in the previous section schedule a single maintenance window. To schedule a recurring maintenance interval - daily, weekly, or monthly - use your operating system's scheduling facility to run `group_maintenance_api.pl` on a periodic basis.

Use Time Only for Start and End Arguments

A scheduled invocation of `group_maintenance_api.pl` should use time, but no date, for the start and end of the maintenance interval. When you run `group_maintenance_api.pl` with time only, the maintenance date is the following day (the day after the command was run).

For example, to set a maintenance window for the "HQ Agents" group for every Sunday from 12 PM to 2 PM, schedule this command to run every Saturday:

```
group_maintenance_api.pl set "HQ Agents" "12:00 PM" "2:00 PM"
```

Tip - Wrap the Script Invocation

Because `group_maintenance_api.pl` takes arguments in quotation signs, you may find it more convenient and robust to wrap the command invocation in another script, and schedule that script for periodic execution.

For example, you can put the command in a batch script like this:

```
REM=============================================================================
@ECHO OFF
PATH_TO_SCRIPT\group_maintenance_api.pl set "HQ Agents" "12:00 PM" "2:00 PM"
REM=============================================================================
```

Note: Using a wrapper is simply a method to ensure quotation marks survive the scheduling process. Some schedulers break commands by stripping quotation marks (") from them upon scheduling.

Sample Script Source

```
#!/usr/bin/perl

use strict;
use Date::Parse;
use LWP::UserAgent;
```

```
use XML::Simple;
use Data::Dumper;

# user configurable variables

my $hq_server = 'http://hq-server.example.com:7080';
my $username = "hqadmin";
my $password = "hqadmin";

# no configurable variables below this point

my ($do_action_url);
my ($action, $group_name, $start_time, $end_time) = @ARGV;
$group_name =~ s/ +/\%20/g;

if ($start_time) {
      if ($start_time !~ /^\d\d\/\d\d\/\d\d+/) {
            $start_time=get_tomorrow($start_time);
      }

      if ($end_time !~ /^\d\d\/\d\d\/\d\d+/) {
            $end_time=get_tomorrow($end_time);
      }
}

my $start_epoch = str2time($start_time)*1000;
my $end_epoch = str2time($end_time)*1000;
my $group_get_url = "/hqu/hqapi1/group/get.hqu?name=$group_name";
my $data = get_xml($hq_server.$group_get_url);
my $group_id = $data->{Group}->[0]->{id};

if ($group_id) {

      my $schedule_get_url = "/hqu/hqapi1/maintenance/get.hqu?groupId=$group_id";
      my $schedule_set_url =
"/hqu/hqapi1/maintenance/schedule.hqu?groupId=$group_id&start=$start_epoch&end=$end_epoch";
      my $unschedule_url = "/hqu/hqapi1/maintenance/unschedule.hqu?groupId=$group_id";

      if ($action eq 'get') {
            $do_action_url = $hq_server.$schedule_get_url;

      } elsif ($action eq 'set') {
            $do_action_url = $hq_server.$schedule_set_url;

      } elsif ($action eq 'unset') {
            $do_action_url = $hq_server.$unschedule_url;
      }

} else {
      $group_name =~ s/\%20/ /g;
      print "No group_id found for group name \"$group_name\"\n";
      exit 1;
}

if ($do_action_url && $group_id) {
      $data = get_xml($do_action_url);
}

# print Dumper($data);

parse_xml($data);

# subroutines
```

```
sub get_xml {
      my ($url) = @_;
      my $user_agent = new LWP::UserAgent;
      my $request = HTTP::Request->new(GET => $url);
      $request -> authorization_basic($username, $password);
      my $response = $user_agent->request($request);
      my $xml = new XML::Simple (ForceArray=>1, KeyAttr=>[]);
      my $data = $xml->XMLin($response->content);
      return $data;

}

sub parse_xml {
      my ($data) = @_;
      my $status_msg = $data->{Status}->[0];
      my $error_code = $data->{Error}->[0]->{ErrorCode}->[0];
      my $reason_txt = $data->{Error}->[0]->{ReasonText}->[0];
      my ($maintenance_event_start, $maintenance_event_end) = (0,0);
      if ($data->{MaintenanceEvent}->[0]->{startTime}) {

            $maintenance_event_start = localtime($data->{MaintenanceEvent}->[0]->{startTime}/1000);
            $maintenance_event_end = localtime($data->{MaintenanceEvent}->[0]->{endTime}/1000);

      }

      print "Status: $status_msg\n";

      if ($error_code) {
            print "Error Code: $error_code\n";
            print "Reason Text: $reason_txt\n";
      }

      if ($maintenance_event_start) {
            print "Start time: $maintenance_event_start\n";
            print "End time: $maintenance_event_end\n";
      }
}

sub get_tomorrow {
      my ($tval) = @_;
            my @t = (localtime(time+86400))[3..5];
            my $tom_date=sprintf ("%d/%d/%d", $t[1] + 1, $t[0], $t[2] + 1900);
            $tval="$tom_date $tval";

      return $tval;

}
```

HQApi metric command 16

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

The **metric** command has options to list and update the metric collection configuration for an individual resource.

To work with metric collection configurations for resource types, see HQApi metricTemplate command (see page 97).

metric Command Options

metric list

The **metric list** command returns a `MetricsResponse` element that defines the all the metrics that are supported for a specific resource and whether each is currently enabled.

Metric list Command Qualifiers

Qualifier	Description	Optionality
--id	limits results to a single resource	required
--enabled	Limits results to metrics that are enabled.	optional

Structure of a MetricsResponse

```
MetricsResponse
Metric
MetricTemplate
```

where there is:

- 1 `Metric` element for each metric that can be collected for the resource
- 1 `MetricTemplate` for each metric that can be collected for the resource

Attributes in a MetricsResponse

Element	Attribute	Description	Allowable Values
`Metric`			
	collectionType		
	indicator		
	defaultOn		
	enabled		
	interval		
	id		
	name		
`MetricTemplate`			

	defaultInterval
	category
	collectionType
	defaultOn
	indicator
	plugin
	units
	alias
	name
	id

metric sync

The **metric sync** command can be used to enable/disable metrics or change their metric collection intervals. The sync operation will only update the `enabled` and `interval` attributes for a `Metric` element.

metric reschedule

metric reschedule

The **metric reschedule** command reschedules metric collection for one or more resources.

It is useful to reschedule metric collection for affected resources after you redeploy a plugin whose metric collection definitions have been updated, for instance, with a new metric, or with changed default metric collection settings.

Rescheduling metric collection for the resources managed by a updated plugin is an alternative to forcing the agent to reload all plugins by restarting it.

Plugin changes that add resource types require agent restart.

If you update a plugin to define a new resource type, you must restart the Hyperic Agent for the changes to take effect. You can only reschedule metric collection for resources that already exist in inventory.

The **metric reschedule** command accepts a `ResourcesResponse` element, which can contain one or more resources. You can create a file containing a `ResourcesResponse` element using the resource list command. See **HQApi resource** command for more information.

Examples

metric list

This command returns the metrics that are enabled for a resource whose internal id is 10661.

```
bash-3.2$ ./bin/hqapi.sh metric list --id=10661 --enabled
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MetricsResponse>
<Status>Success</Status>
<Metric collectionType="0" indicator="true" defaultOn="false" enabled="true"
interval="60000" id="10001" name="Availability">
<MetricTemplate defaultInterval="60000" category="AVAILABILITY" collectionType="0"
defaultOn="true" indicator="true" plugin="system" units="percentage"
alias="Availability" name="Availability" id="10436"/>
</Metric>
<Metric collectionType="0" indicator="true" defaultOn="false" enabled="true"
interval="300000" id="10007" name="Free Memory">
<MetricTemplate defaultInterval="300000" category="UTILIZATION" collectionType="0"
defaultOn="true" indicator="true" plugin="system" units="B"
alias="MemFree" name="Free Memory" id="10373"/>
</Metric>
<Metric collectionType="0" indicator="true" defaultOn="false" enabled="true"
interval="300000" id="10010" name="Load Average 5 Minutes">
<MetricTemplate defaultInterval="300000" category="UTILIZATION" collectionType="0"
defaultOn="true" indicator="true" plugin="system" units="none"
alias="LoadAverage1" name="Load Average 5 Minutes" id="10429"/>
</Metric>
<Metric collectionType="0" indicator="true" defaultOn="false" enabled="true"
interval="300000" id="10067" name="Swap Used">
<MetricTemplate defaultInterval="300000" category="UTILIZATION" collectionType="0"
defaultOn="true" indicator="true" plugin="system" units="B"
alias="SwapUsed" name="Swap Used" id="10414"/>
</Metric>
</MetricsResponse>
```

metric sync

In this example, the first command writes metrics that are enabled for a resource whose internal id is 10661 to a file called metrics.xml. The second command sends metrics.xml to the sync method to update the metric collection configuration in HQ.

```
$ ./bin/hqapi.sh metric list --id=10661 --enabled > metrics.xml
...
...
$ cat metrics.xml | ./bin/hqapi.sh metric sync
Successfully synced 4 metrics.
```

metric reschedule

The command below reschedules metric collection for the resources defined in the `resources.xml` file.

```
$ ./bin/hqapi.sh metric reschedule --file=resources.xml
Successfully rescheduled 2 resources
```

HQApi metricData command 17

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

The **metricData** command can be used to list all measurements for a particular metric for a particular resource in HQ.

Command Options

list

list Command Output

The data returned by the **list** option varies by the command qualifiers you supply. See Examples (see page 95).

list Command Qualifiers

The **list** command requires one of the following qualifiers: `groupID`, `metricID`, or `resourceID`.

Qualifier	Description
--file	If specified, use the given file for commands that take XML input. If not specified, stdin will be used.
--formatDates	When specified timestamps will be formatted using the given format. Defaults to yyyy-MM-dd HH:mm:ss
--groupId	The group id to query for metric data. Only supported for compatible groups. To determine the ID for the desired group, use the group list (see page 78) command, using the --compatible qualifer.
--hours	The number of hours of data to query. Defaults to 8.
--metricId	The internal ID for the metric to list. To determine the ID for the metric you wish to list, use the metric list (see page 92) command.
--resourceId	The resource id to query for metric data. To determine the ID for the desired resource, use the resource list (see page 102) command.

Examples

List Measurements for a Metric

The query below returns measurements for the metric whose ID is 10001 — the Availability metric for a platform in inventory. The timestamp and the value are returned for each data point.

For brevity, only the last three measurements returned are shown.

```
sh bin/hqapi.sh metricData list --metricId=10001
,Value
1289412780000,0.5
1289412840000,1.0
1289412900000,1.0
```

List Measurements for a Resource

The query below returns measurements of all metrics collected in the last hour for a resource whose id is 10826 — a platform in inventory.

The first row returned is comma-separated list of the metrics returned, in `MetricName(id=MetricID` format.

Each remaining row lists a timestamp, and the values collected for each metric at that time.

For brevity, only the measurements collected in the last 10 minutes are shown. Note that Availability is reported every minute, and the other metrics are reported every five minutes.

```
sh bin/hqapi.sh metricData list  --resourceId=10826 --hours=1
,Availability(id=10001),Free Memory(id=10007),Load Average 5 Minutes(id=10010),Swap Used(id=10067)
1289413200000,1.0,9.4183424E7,0.99805,2.468708352E9
1289413260000,1.0,,,
1289413320000,1.0,,,
1289413380000,1.0,,,
1289413440000,1.0,,,
1289413500000,1.0,8.6974464E7,0.56104,2.468708352E9
1289413560000,1.0,,,
1289413620000,1.0,,,
1289413680000,1.0,,,
1289413740000,1.0,,,
1289413800000,1.0,,,
```

List Measurements for a Compatible Group

The query below returns measurements for a compatible group whose id is 10826.

The first row returned is comma-separated list of the columns returned:

- Metric Name
- Template ID
- Min, Max, and Ave — The minimum value for the metric over the last 8 hours. If the `--hours` qualify was specified, period specified.
- Last — The last value reported for the metric.

Each remaining row is a comma-separated list of the values of each column for a measurement.

```
sh bin/hqapi.sh metricData list --groupId=10001
Metric Name, Template Id, Min, Max, Avg, Last
Number Of Row Inserts per Minute,20915,0.0,4.9E-324,0.0,0.0
Data Space Used,20909,8.0,8.0,8.0,240.0
Index Space Used,20899,16.0,48.0,38.4,1152.0
Sequential Scans per Minute,20904,0.0,0.00383,2.0157894736842104E-4,0.0038299999999999996
Availability,20910,0.0,1.0,0.1284722222222222,0.1284722222222222
```

HQApi metricTemplate command 18

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

The **metricTemplate** command has options to output the metric collection configuration for a specified resource type. This is useful for:

- Reviewing the metric collection configuration for a resource type.
- Updating the metric collection configuration for a resource type.

To work with the metric collection configuration for a specific resource, see the HQApi metric command (see page 90).

metricTemplate Command Options

metricTemplate list

The **metricTemplate list** returns an MetricTemplatesResponse element that defines all of the metrics that are supported for a specific escalations defined in HQ.

metricTemplate list Command Qualifiers

The --prototype qualifier, which specifies a resource type, is required for the **list** option.

Structure of a metricTemplatesResponse

The MetricTemplatesResponse returned by the **metricTemplate list** has this element structure.

```
MetricTemplatesResponse
      MetricTemplate
```

where there is 1 MetricTemplate element for each metric that can be collected for the resource type.

Attributes in a MetricTemplatesResponse

Element	Attribute	Description	Allowable Values
`MetricTemplate`			
	defaultInterval		
	category		
	collectionType		
	defaultOn		
	indicator		
	plugin		
	units		
	alias		
	name		

id

metricTemplate sync

The **metricTemplate sync** option can be used to update the values of the `defaultInterval`, `defaultOn`, and `indicator` attributes. The values of other attributes in the `MetricTemplate` element are not written to HQ.

Examples

metricTemplate list

In this example, the metrics supported for the resource type "CPU" are listed.

```
$ ./bin/hqapi.sh metricTemplate list --prototype="CPU"
<MetricTemplatesResponse>
    <Status>Success</Status>
    <MetricTemplate defaultInterval="600000" category="AVAILABILITY" collectionType="0"
                    defaultOn="true" indicator="true" plugin="system" units="percentage"
                    alias="Availability" name="Availability" id="11014"/>
    <MetricTemplate defaultInterval="300000" category="UTILIZATION" collectionType="0"
                    defaultOn="true" indicator="true" plugin="system" units="percentage"
                    alias="CpuIdle" name="Cpu Idle" id="11016"/>
    <MetricTemplate defaultInterval="600000" category="UTILIZATION" collectionType="2"
                    defaultOn="false" indicator="false" plugin="system" units="ms"
                    alias="CpuIdleSec" name="Cpu Idle Time" id="11010"/>
    <MetricTemplate defaultInterval="600000" category="UTILIZATION" collectionType="0"
                    defaultOn="false" indicator="false" plugin="system" units="ms"
                    alias="CpuIdleSec1m" name="Cpu Idle Time per Minute" id="11018"/>
    <MetricTemplate defaultInterval="300000" category="UTILIZATION" collectionType="0"
                    defaultOn="true" indicator="true" plugin="system" units="percentage"
                    alias="CpuUsage" name="Cpu Usage" id="11015"/>
    <MetricTemplate defaultInterval="600000" category="UTILIZATION" collectionType="2"
                    defaultOn="false" indicator="false" plugin="system" units="ms"
                    alias="CpuWaitSec" name="Cpu Wait Time" id="11013"/>
    <MetricTemplate defaultInterval="600000" category="UTILIZATION" collectionType="0"
                    defaultOn="false" indicator="false" plugin="system" units="ms"
                    alias="CpuWaitSec1m" name="Cpu Wait Time per Minute" id="11011"/>
    <MetricTemplate defaultInterval="300000" category="UTILIZATION" collectionType="0"
                    defaultOn="true" indicator="true" plugin="system" units="percentage"
                    alias="CpuSys" name="System Cpu" id="11012"/>
    <MetricTemplate defaultInterval="600000" category="UTILIZATION" collectionType="2"
                    defaultOn="false" indicator="false" plugin="system" units="ms"
                    alias="CpuSysSec" name="System Cpu Time" id="11021"/>
    <MetricTemplate defaultInterval="600000" category="UTILIZATION" collectionType="0"
                    defaultOn="false" indicator="false" plugin="system" units="ms"
                    alias="CpuSysSec1m" name="System Cpu Time per Minute" id="11020"/>
    <MetricTemplate defaultInterval="300000" category="UTILIZATION" collectionType="0"
                    defaultOn="true" indicator="true" plugin="system" units="percentage"
                    alias="CpuUser" name="User Cpu" id="11009"/>
    <MetricTemplate defaultInterval="600000" category="UTILIZATION" collectionType="2"
                    defaultOn="false" indicator="false" plugin="system" units="ms"
                    alias="CpuUserSec" name="User Cpu Time" id="11019"/>
    <MetricTemplate defaultInterval="600000" category="UTILIZATION" collectionType="0"
                    defaultOn="false" indicator="false" plugin="system" units="ms"
                    alias="CpuUserSec1m" name="User Cpu Time per Minute" id="11017"/>
</MetricTemplatesResponse>
```

metricTemplate sync

In this example, the first command writes the `MetricTemplate` for each metric supported for the "CPU" resource type to a file called cpu-metrics.xml.

The second command sends the cpu-metrics.xml to the sync method to update the values of the `defaultInterval`, `defaultOn`, and `indicator` attributes of each metric.

```
$ ./bin/hqapi.sh metricTemplate list --prototype="CPU" > cpu-metrics.xml
...
...
$ cat cpu-metrics.xml | ./bin/hqapi.sh metricTemplate sync
Successfully synced 13 templates.
```

HQApi resource command 19

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

The **resource** command can list resource properties, create or update resources, and delete resources.

resource Command Options

resource list

The **resource list** command returns an `ResourcesResponse` element that specifies inventory and configuration properties for selected resources.

resource list Command Qualifiers

The command qualifiers for **resource list** limit the resources returned and control how much detail is provided.

Note that you must supply one of:

- `agentId`
- `id`
- `platform`
- `prototype`

Qualifier	Description
--aeid	List the resource with the specified `aeid` (`AppdefEntityID`), which is its `typeID:instanceID` combination. See Attributes in an ResourcesResponse (see page 103) for definitions of `typeID` and `instanceID` and `instanceID`. The `aeid` qualifier is useful if you do not know a resource's resource ID (see `-id` below), but you do know its `aeid`.
--agentId	List the resources managed by the agent with the specified resource ID. This is useful for listing the resources that an agent monitors remotely, such as network services.
--children	Include child resources. Use of this qualifier increases the length of time for **resource list** to return results.
--id	List the resource with the specified resource ID.
--name	Limit the resources returned to those whose names match the specified regular expression
--parentPlatform	May be used (only in combination with the `-id` qualifer) to list the parent platform of the resource identified by `-id`.
--platform	List resources for the given platform name.
--prototype	Only list resources of the given type.
--id	Only list the resource with the given id.
--verbose	Include resource properties and configuration. Use of this qualifier increases the length of time for **resource list** to return results.

Structure of a ResourcesResponse

```
ResourcesResponse
Resource
ResourceConfig
ResourceProperty
ResourcePrototype
Agent
Ip
ResourceInfo
```

where there is:

- 1 `Resource` element for each resource returned, which contains:
  - Optionally, if `--verbose` qualifier is used, a `ResourceConfig` element for each configuration option for a resource.
  - Optionally, if `--verbose` qualifier is used, a `ResourceProperty` element for each configuration option for a resource.
  - 1 `ResourcePrototype` element.
  - 1 `Agent` element.
  - If a resource is a platform, 1 `Ip` element for each network interface on the platform.
  - 1 `ResourceInfo` element.

Attributes in an ResourcesResponse

This table defines the attributes in a `ResourcesResponse`.

Element	Attribute	Description	Values
`resource`			
	`id`	Internal Hyperic ID for the resource. This value is assigned by Hyperic Server when the resource is created. When you create an resource using the API, you do not supply the value of `id`.	Numeric value, for example, "10002"
	`name`	Name or the resource.	"Tech-x-MacBook-Air.local"
	`description`	Description of the resource.	
	`location`	Location of the resource, if one was defined.	
	`instanceId`	Uniquely identifies a resource among resources with the same `TypeId`. Forms a portion of a resource's `AppdefEntityID`), which is `TypeId:instanceID`	

	`typeId`	Identifies the inventory level of the resource. Forms a portion of a resource's `AppdefEntityID`), which is `TypeId:instanceID`.	"1" — platform "2" — server "3" — service "4" — application "5" — group
`ResourceConfig`			
	`key`	Name of a resource configuration option.	
	`value`	Value of a resource configuration option.	
`ResourceProperty`			
	`key`	Name of a resource property.	
	`value`	Value of a resource property.	
`ResourcePrototype`			
	`resourceTypeId`	Identifies the inventory level of the resource type.	"1" — platform "2" — server "3" — service "4" — application "5" — group
	`id`	The internal HQ ID for a resource type.	Example value:"10329"
	`instanceId`	Uniquely identifies a resource type among resources with the same `resourceTypeId`.	Example value:"10329"
	`name`	The name of the resource type.	Example values: ■ "Linux" ■ "Tomcat 6.0 Connector"
`Agent`	Present only if the resource is a platform type.		
	`id`	Internal Hyperic ID for the agent `color:red` that manages the platform.	Numeric value, for example, "10002"
	`address`	The IP address on the agent that the server uses to contact the agent.	172.14.216.1
	`port`	The port on the agent's listen address the server uses to contact the agent.	2144
	`version`	Agent version	

	`unidirectional`	Whether or not the agent is configured to use unidirectional communication with the Hyperic Agent.	"true" or "false"
`Ip`	Present only if the resource is a platform type.		
	`address`	IP address of a network interface on the platform.	
	`mac`	Mac address of the network interface.	
	`netmask`	Netmask for the network interface.	
`ResourceInfo`			
	`key`	Name of a resource property.	For a platform — "fqdn" For a server or service ---"autoIdentifier" and "installPath"
	`value`	Value of a resource property.	Tech-x-MacBook-Air.local

resource sync

The **resource sync** command creates or updates resources in Hyperic inventory from the content of a `ResourcesResponse` object in a file you specify.

You can use the results of a **resource list** command to create a starting point for your sync file. Use appropriate qualifiers to return the XML elements for resource or resources you want to update or use as a prototype to create new resources.

The **resource sync** command requires a single **--file** qualifier to specify the sync file.

This qualifiers is optional:

Qualifier	Description
--batchSize	Sync resources in batches of the size specified. Supply an integer value.
--file	Use to specify a file that contains the `ResourcesResponse` with the resource definitions to sync.
--setConfig	Set or update the specified `ResourceConfigUse`. Supply a list of key value pairs.

Note: You cannot change a resource's type.

resource delete

The **resource delete** command requires a single **--id** qualifier to specify the internal ID of the resource to delete. You can obtain resource IDs with the **resource list** command.

resource createPlatform

The **createPlatform** command is used to create new platforms in Hyperic.

You can also create new resources with the resource sync (see page 105) command.

resource createPlatform Command Qualifiers

The **createPlatform** command requires the following arguments:

Qualifier	Description
--prototype	The platform type to create.
--agentId	The id of the agent that will service the new platform. Agent ids may be retrieved using the agent list command.
--fqdn	The FQDN of the newly created platform.
--ip	The IP address of the newly created platform.
--name	The name of the newly created platform.

In addition to the required arguments, you can specify resource inventory and configuration properties on the command line as "extra arguments". Extra arguments are provided after the required arguments, preceded by an empty "--". For each property you want to define, you supply a key=value pair. See the resource createPlatform (see page 105) example.

The properties available for resources vary by resource type. If you do not know the properties supported for a particular resource, you can get them by running a **resource list --prototype=*ResourceType* --verbose** command for the resource type, as shown below in List Resources of the Same Type Verbosely (see page 108).

resource createServer

The **createServer** command is used to create new servers in Hyperic.

You can also create new resources with the resource sync (see page 105) command.

resource createServer Command Qualifiers

The **createServer** command requires the following arguments:

Qualifier	Description
--prototype	The server type to create.
--resourceId	The platform Resource id to create this server Resource.
--name	The name of the newly created server.

As described in resource createPlatform (see page 105) you can also specify optional resource inventory and configuration properties on the command line as "extra arguments". See the resource createServer (see page 106) example.

resource createService

The **createService** command is used to create new service Resources in Hyperic.

You can also create new resources with the resource sync (see page 105) command.

resource createService Command Qualifiers

The **createService** command requires the following arguments:

Qualifier	Description
--prototype	The service Resource type to create.
--resourceId	The parent server or platform Resource id.
--name	The name of the service to create

As described in resource createPlatform (see page 105) you can also specify optional resource inventory and configuration properties on the command line as "extra arguments". See the resource createService (see page 106) example.

resource move

Use the **resource move** command to move a manually created resource from one platform to another, most typically a platform service. For instance, if you have an HTTP check configured on one platform and decide you prefer a different Hyperic Agent to perform the check, you can move it to the target platform, retaining its configuration and history. Similarly, you can move a manually created server whose child resources are manually created platform services.

Do Not Move Auto-discovered Resources

Do not use **resource move** move a resource that was auto-discovered; this will result in duplicate resources and errors to `server.log`. Do not move vSphere Hosts, vSphere VMs, or resources that are children of VMs.

resource move Command Qualifiers

The **resource move** command requires the following.

Qualifier	Description
--id	The internal Hyperic ID of a server or platform service to move.
--to	The internal Hyperic ID of the destination platform.
--setName	Optional. Sets the resource name to the name specified.

Note: You can determine a resource's internal Hyperic ID using the **resource list** command.

Examples

List Resources of the Same Type

This command returns all resources whose resource type is "macOSX". In this example there is only one instance of that type. Only key properties are returned for each resource returned.

```
$ sh bin/hqapi.sh resource list --prototype="MacOSX"
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesResponse>
<Status>Success</Status>
<Resource id="10880" name="Marrys-MacBook-Air.local" description="Mac OS X Snow Leopard"
location="" instanceId="10001" typeId="1">
<ResourcePrototype resourceTypeId="1" instanceId="10002" id="10002" name="MacOSX"/>
<Agent id="10001" address="127.0.0.1" port="2144" version="4.6.0.BUILD-SNAPSHOT"
unidirectional="false"/>
<Ip address="172.16.0.1" mac="00:50:56:C0:00:01" netmask="255.255.255.0"/>
<Ip address="127.0.0.1" mac="00:00:00:00:00:00" netmask="255.0.0.0"/>
<Ip address="172.16.216.1" mac="00:50:56:C0:00:08" netmask="255.255.255.0"/>
<Ip address="192.168.0.11" mac="10:93:E9:0E:8D:BC" netmask="255.255.255.0"/>
<Ip address="192.168.0.5" mac="10:9A:DD:40:C6:27" netmask="255.255.255.0"/>
<Ip address="10.16.244.59" mac="00:00:00:00:00:00" netmask="255.255.255.255"/>
<Ip address="10.16.245.222" mac="00:00:00:00:00:00" netmask="255.255.255.255"/>
<ResourceInfo key="fqdn" value="Marie-McGarrys-MacBook-Air.local"/>
</Resource>
</ResourcesResponse>
```

List Resources of the Same Type Verbosely

This command is the same as the previous example, with the `--verbose` qualifier. The results include all configuration and inventory properties for a MacOSX platform.

```
$ sh bin/hqapi.sh resource list --prototype="MacOSX" --verbose
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesResponse>
<Status>Success</Status>
<Resource id="10880" name="Marrys-MacBook-Air.local" description="Mac OS X Snow Leopard"
location="" instanceId="10001" typeId="1">
<ResourceConfig key="platform.log_track.enable" value="true"/>
<ResourceConfig key="platform.log_track.level" value="Warn"/>
<ResourceConfig key="platform.log_track.include" value=""/>
<ResourceConfig key="platform.log_track.exclude" value=""/>
<ResourceConfig key="platform.log_track.files" value=""/>
<ResourceConfig key="platform.config_track.enable" value="true"/>
<ResourceConfig key="platform.config_track.files" value=""/>
<ResourceProperty key="arch" value="i386"/>
<ResourceProperty key="version" value="10.6.7"/>
<ResourceProperty key="vendor" value="Apple"/>
<ResourceProperty key="vendorVersion" value="10.6"/>
<ResourceProperty key="ram" value="2048 MB"/>
<ResourceProperty key="cpuSpeed" value="2 @ 1860 MHz (1x2)"/>
<ResourceProperty key="ip" value="192.168.0.11"/>
<ResourceProperty key="primaryDNS" value="10.16.65.110"/>
<ResourceProperty key="secondaryDNS" value="10.16.65.111"/>
<ResourceProperty key="defaultGateway" value="192.168.0.1"/>
<ResourcePrototype resourceTypeId="1" instanceId="10002" id="10002" name="MacOSX"/>
<Agent id="10001" address="127.0.0.1" port="2144" version="4.6.0.BUILD-SNAPSHOT"
unidirectional="false"/>
<Ip address="172.16.0.1" mac="00:50:56:C0:00:01" netmask="255.255.255.0"/>
<Ip address="127.0.0.1" mac="00:00:00:00:00:00" netmask="255.0.0.0"/>
<Ip address="172.16.216.1" mac="00:50:56:C0:00:08" netmask="255.255.255.0"/>
<Ip address="192.168.0.11" mac="10:93:E9:0E:8D:BC" netmask="255.255.255.0"/>
<Ip address="192.168.0.5" mac="10:9A:DD:40:C6:27" netmask="255.255.255.0"/>
<Ip address="10.16.244.59" mac="00:00:00:00:00:00" netmask="255.255.255.255"/>
<Ip address="10.16.245.222" mac="00:00:00:00:00:00" netmask="255.255.255.255"/>
<ResourceInfo key="fqdn" value="Marie-McGarrys-MacBook-Air.local"/>
</Resource>
</ResourcesResponse>
```

Write Resource Data to a File

This command writes key property for all resources of the type "HTTP" to a
file called "http-resources.xml". Once the file is created, the property values in it can be edited, and as shown in the following example, you can update Hyperic with the contents of the file.

```
$ ./bin/hqapi.sh resource list --prototype="HTTP" > http-resources.xml
```

Update Resource Properties

This command pipes the contents of the "http-resources.xml" file to the `resource sync` command, which updates existing resources in Hyperic with the property values defined in the XML.

```
$ cat http-resources.xml | ./bin/hqapi.sh resource sync
Successfully synced 10 resources.
```

Delete a Resource

The command below deletes the resource whose internal ID is 10654.

```
$ ./bin/hqapi.sh resource delete --id=10654
Successfully deleted resource id 10654
```

Create a New Platform

The command below creates a new platform in Hyperic. Note that the command includes:

- Required qualifiers that supply the platform's resource type, the internal ID of its Hyperic Agent, FQDN, IP address, and name.
- Optional qualifiers that follow the empty "--" supply SNMP configuration properties as name=value pairs.

```
$ ./bin/hqapi.sh resource createPlatform --prototype="Network Device" --agentId=10001
--fqdn=netdevice.hyperic.com --ip=10.0.0.1 --name="Hyperic Router" -- interface.index=ifDescr
snmpIp=10.0.0.1 snmpPort=161 snmpVersion=v2c snmpCommunity=public
Successfully created 'Hyperic Router' (id=11518)
$
```

Additional qualifiers could be used to set additional configuration properties.

Create a New Server

The command below creates a new server in Hyperic. Note that the command includes:

- Required qualifiers that supply the servers's resource type, the internal ID for its host platform, name, and name.
- Optional qualifiers additional properties as name=value pairs.

```
$ ./bin/hqapi.sh resource createServer --prototype="Apache httpd" --resourceId=10661 --name="Test
Apache" -- hostname=localhost port=80 sotimeout=10 path=/server-status
Successfully created 'Test Apache' (id=10976)
```

Create a New Service

The command below creates a new service in Hyperic. Note that the command includes:

- Required qualifiers that supply the service's resource type, the internal ID for its host platform or server, name, and name.
- Optional qualifiers that follow the empty "--" supply additional properties as name=value pairs.

Additional qualifiers could be used to set additional configuration properties.

```
$ ./bin/hqapi.sh resource createService --prototype="HTTP" --resourceId=10661 --name="HTTP check"
-- hostname=localhost port=7080 sotimeout=60 path=/ method=HEAD
Successfully created 'HTTP check' (id=10978)
$
```

HQApi resourceprototype command 20

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

The **resourceprototype** command returns the internal Hyperic ID and the name for resource types. You can list all resource types supported by Hyperic, or only those resource types that exist in inventory.

resourceprototype Command Options

This section defines command options.

resourceprototype list

The **resourceprototype list** command returns a list of resource types — all supported resource types, or if the `--existing` qualifier is specified, only those that exist in inventory

Command qualifiers:

Qualifier	Description	
`--existing`	Results will be limited to resource types that exist in inventory.	no

Attributes in an ResourcePrototype Element

This table defines the attributes in a `<ResourcePrototype>` element.

Attribute	Description and Values
`id`	Internal Hyperic ID for the resource type.
`name`	Name of the resource type.

resourceprototype Command Examples

resourceprototype list --existing

This command returns a list of resource types in inventory.

```
sh bin/hqapi.sh resourceprototype list --existing
```

The command results are:

```
<ResourcePrototypesResponse>
    <Status>Success</Status>
    <ResourcePrototype id="10077" name="ActiveMQ Embedded 5.3"/>
    <ResourcePrototype id="10083" name="ActiveMQ Embedded 5.3 Broker"/>
    <ResourcePrototype id="10105" name="ActiveMQ Embedded 5.3 Connector"/>
    <ResourcePrototype id="10093" name="ActiveMQ Embedded 5.3 Topic"/>
    <ResourcePrototype id="10449" name="Apache Tomcat 6.0"/>
    <ResourcePrototype id="10454" name="Apache Tomcat 6.0 Cache"/>
    <ResourcePrototype id="10467" name="Apache Tomcat 6.0 Global Request Processor"/>
    <ResourcePrototype id="10453" name="Apache Tomcat 6.0 HQ Internals"/>
    <ResourcePrototype id="10462" name="Apache Tomcat 6.0 Hibernate Session Factory"/>
    <ResourcePrototype id="10460" name="Apache Tomcat 6.0 Hyperic Data Source"/>
    <ResourcePrototype id="10470" name="Apache Tomcat 6.0 JSP Monitor"/>
    <ResourcePrototype id="10450" name="Apache Tomcat 6.0 Servlet Monitor"/>
    <ResourcePrototype id="10464" name="Apache Tomcat 6.0 Thread Pools"/>
    <ResourcePrototype id="10465" name="Apache Tomcat 6.0 Web Module Stats"/>
    <ResourcePrototype id="10020" name="CPU"/>
    <ResourcePrototype id="10011" name="FileServer"/>
    <ResourcePrototype id="10019" name="FileServer Mount"/>
    <ResourcePrototype id="10209" name="HQ Agent"/>
    <ResourcePrototype id="10002" name="MacOSX"/>
    <ResourcePrototype id="10329" name="Net Services"/>
    <ResourcePrototype id="10013" name="NetworkServer"/>
    <ResourcePrototype id="10014" name="NetworkServer Interface"/>
    <ResourcePrototype id="10389" name="PostgreSQL 8.2"/>
    <ResourcePrototype id="10400" name="PostgreSQL 8.2 Table"/>
    <ResourcePrototype id="10012" name="ProcessServer"/>
</ResourcePrototypesResponse>
```

HQApi role command 21

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

The **role** command has options to list, update, and create non-system roles.

You cannot modify roles that are internal to HQ, such as the 'Super User Role', or delete roles using the **role** command

Command Options

list

list Command Output

The **list** command option returns a `RolesResponse` that contains information about one or all of the roles defined in HQ. The `RolesResponse` element has a `Role` element for each role returned, which contains an `Operation` element for each permission granted to the role, and a `User` element for each user assigned to the role.

```
RolesResponse
     Role
          description
          name
          id
          Operation
          User
                htmlEmail
                active
                phoneNumber
                SMSAddress
                emailAddress
                department
                lastName
                firstName
                name
                id
```

list Command Qualifiers

You may optionally supply a command qualifier to return a single role.

Qualifier	Description
--id	Only list the role with the given id.
--name	Only list the role with the given name.

sync

The **sync** option can be used to create or update roles, taking the XML via standard in.

To add update an existing role, use **role sync** with the **id** or **name** qualifier to output the `Role` element for the role to be updated to a file. Edit the file to:

- Add or remove `Operation` elements as desired.

- Add or remove `User` elements as desired. When adding a user to the role, identify the user by either username or the internal HQ ID for the user. Do not enter values for other `User` element attributes - they will overwrite existing values in the HQ database.

To create a new role, create a `RolesResponse` with a `Role` element for the new role, omitting the `id` attribute for the role; HQ will assign an `id` to the new role. Include an `Operation` for each permission for the role, and a element `User` element for each user to add to the role, identifying a user by either username or the internal HQ ID for the user.

Samples

role list

```
$ ./bin/hqapi.sh role list
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RolesResponse>
    <Status>Success</Status>
    <Role description="" name="Guest Role" id="2">
        <Operation>viewApplication</Operation>
        <Operation>viewPlatform</Operation>
        <Operation>viewResourceGroup</Operation>
        <Operation>viewRole</Operation>
        <Operation>viewServer</Operation>
        <Operation>viewService</Operation>
        <Operation>viewSubject</Operation>
        <User htmlEmail="false" active="false" phoneNumber="415-225-0057" SMSAddress=""
emailAddress="localhost"
              department="" lastName="User" firstName="Guest" name="guest" id="2"/>
    </Role>
</RolesResponse>
```

role sync

You can create and update roles with the **roll sync** command. In this example, the **role list** command is used to write a `RolesResponse` element containing the `Role` element for a role named "Darwinner" to a file named "roles.xml" - you edit this file to add or remove `Operation` or `User` elements.

The second command pipes the contents of the edited "roles.xml" to the **role sync** command.

```
$ ./bin/hqapi.sh role list --name=Darwinner > roles.xml
...
...
$ cat roles.xml | ./bin/hqapi.sh role sync
Successfully synced 1 roles.
```

Understanding Role Permissions

The `<operation>` elements you define in `<role>` element are listed below in Operations List (see page 117).

Operations List

Operation	Description

manageGroupAlerts	view, create, edit, delete, acknowledge, fix alerts on resource group
managePlatformAlerts	view, create, edit, delete, acknowledge, fix alerts on platforms
manageServerAlerts	view, create, edit, delete, acknowledge, fix alerts on servers
manageServiceAlerts	view, create, edit, delete, acknowledge, fix alerts on services
createApplication	create new application
modifyApplication	change application properties and add/remove services
removeApplication	delete application
viewApplication	view-only access to applications
createEscalation	create new escalation
modifyEscalation	change escalation
removeEscalation	delete escalation
modifyResourceGroup	change group properties and membership
removeResourceGroup	delete group
viewResourceGroup	view-only access to groups
controlPlatform	perform control action on platforms
createPlatform	create new platform
modifyPlatform	change platform properties and add/remove servers to platforms
removePlatform	delete platform
viewPlatform	view platforms
createRole	create new role
modifyRole	change role permissions membership
removeRole	delete role
viewRole	view roles
addServer	create a server
controlServer	perform control action on servers
modifyServer	change server properties and add/remove services to servers
removeServer	delete server from inventory
viewServer	view servers
addService	create a service
controlService	perform control action on services
modifyService	change service properties
removeService	delete services from inventory

viewService	view services
createSubject	create new user
modifySubject	change user properties
removeSubject	delete user account
viewSubject	view users

HQApi serverConfig command 22

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

An authorized user can use the **serverConfig** command to update selected Hyperic Server configuration properties, which are listed and defined in Server Configuration Reference (see page 123) below. For the most part, these are the properties configurable on the **HQ Server Settings** page in the Hyperic user interface.

serverConfig Command Options

serverConfig get

The **serverConfig get** command returns a `<ServerConfigResponse>` element that lists selected Hyperic Server configuration properties and their values.

Structure of a ServerConfigResponse Element

The `ServerConfigResponse` object returned by the **serverConfig get** command has this element structure:

```
<serverConfigResponse>
     <ServerConfig>
```

where there there is:

- one `<serverConfig>` element for each Hyperic Server configuration property returned.

Attributes in an ServerConfigResponse

This table defines the attributes in a `ServerConfigResponse`.

Element	Attribute	Description and Values
ServerConfig		
	key	One of the Hyperic server properties listed in Server Configuration Reference (see page 123) below.
	value	An allowed value for the server property. See Server Configuration Reference (see page 123) below for definitions and example values.

Example - serverConfig get

This command:

```
serverConfig get
```

returns results similar to the following following:

```
<ServerConfigResponse>
    <Status>Success</Status>
    <ServerConfig key="ALERT_PURGE" value="2678400000"/>
    <ServerConfig key="ARC_SERVER_URL" value=""/>
    <ServerConfig key="CAM_BASELINE_DATASET" value="604800000"/>
    <ServerConfig key="CAM_BASELINE_FREQUENCY" value="259200000"/>
    <ServerConfig key="CAM_BASELINE_MINSET" value="40"/>
    <ServerConfig key="CAM_BASE_URL" value="http://10.0.0.120:7080/"/>
    <ServerConfig key="CAM_DATA_MAINTENANCE" value="3600000"/>
    <ServerConfig key="CAM_DATA_PURGE_RAW" value="172800000"/>
    <ServerConfig key="CAM_EMAIL_SENDER" value="hqadmin@intranet.hyperic.net"/>
    <ServerConfig key="CAM_LDAP_BASE_DN" value="DC=testad,DC=hyperic,DC=net"/>
    <ServerConfig key="CAM_LDAP_BIND_DN"
value="CN=Administrator,CN=Users,DC=testad,DC=hyperic,DC=net"/>
    <ServerConfig key="CAM_LDAP_BIND_PW" value="xxxxxx"/>
    <ServerConfig key="CAM_LDAP_FILTER" value=""/>
    <ServerConfig key="CAM_LDAP_LOGIN_PROPERTY" value="sAMAccountName"/>
    <ServerConfig key="CAM_LDAP_NAMING_PROVIDER_URL" value="ldap://10.0.0.34"/>
    <ServerConfig key="CAM_LDAP_PROTOCOL" value="ssl"/>
    <ServerConfig key="CAM_SMTP_HOST" value="127.0.0.1"/>
    <ServerConfig key="DATA_REINDEX_NIGHTLY" value="true"/>
    <ServerConfig key="EVENT_LOG_PURGE" value="2678400000"/>
    <ServerConfig key="EXTERNAL_HELP" value="true"/>
    <ServerConfig key="HQ_ALERTS_ENABLED" value="true"/>
    <ServerConfig key="HQ_ALERT_NOTIFICATIONS_ENABLED" value="true"/>
    <ServerConfig key="HQ_ALERT_THRESHOLD" value="0"/>
    <ServerConfig key="HQ_ALERT_THRESHOLD_EMAILS" value=""/>
    <ServerConfig key="HQ_HIERARCHICAL_ALERTING_ENABLED" value="true"/>
    <ServerConfig key="KERBEROS_DEBUG" value="false"/>
    <ServerConfig key="KERBEROS_KDC" value=""/>
    <ServerConfig key="KERBEROS_REALM" value=""/>
    <ServerConfig key="OOB_ENABLED" value="true"/>
    <ServerConfig key="SNMP_AGENT_ADDRESS" value=""/>
    <ServerConfig key="SNMP_AUTH_PASSPHRASE" value=""/>
    <ServerConfig key="SNMP_AUTH_PROTOCOL" value=""/>
    <ServerConfig key="SNMP_COMMUNITY" value="public"/>
    <ServerConfig key="SNMP_CONTEXT_NAME" value=""/>
    <ServerConfig key="SNMP_DEFAULT_NOTIFICATION_MECHANISM" value="v1 Trap"/>
    <ServerConfig key="SNMP_ENGINE_ID" value=""/>
    <ServerConfig key="SNMP_ENTERPRISE_OID" value=""/>
    <ServerConfig key="SNMP_GENERIC_ID" value=""/>
    <ServerConfig key="SNMP_PRIVACY_PROTOCOL" value=""/>
    <ServerConfig key="SNMP_PRIV_PASSPHRASE" value=""/>
    <ServerConfig key="SNMP_SECURITY_NAME" value=""/>
    <ServerConfig key="SNMP_SPECIFIC_ID" value=""/>
    <ServerConfig key="SNMP_TRAP_OID" value="1.2.3.4"/>
    <ServerConfig key="SNMP_VERSION" value=""/>
</ServerConfigResponse>
```

See Server Configuration Reference (see page 123) for property definitions.

Example - Write Results of serverConfig get to a file

This command writes the `<ServerConfigResponse>` element to a new file:

```
hqapi.sh serverConfig get >> ServerConfiguration.xml
```

serverConfig set

The **serverConfig set** command takes updates server properties with the values specified in a `<ServerConfigResponse>`.

This command writes updates the Hyperic Server configuration with the property values specified in the `<ServerConfigResponse>` element in the `ServerConfiguration.xml` file.

```
cat ServerConfiguration.xml | ./bin/hqapi.sh serverConfig set
```

serverConfig getParameter

The **serverConfig getParameter** command returns the value of a specified Hyperic Server configuration property. Specify the name of the property with the `--key` qualifier.

For example, this command returns the value of the Hyperic Server's ALERT_PURGE property:

```
hqapi.sh serverConfig getParameter --key=ALERT_PURGE
```

The results returned are:

```
Current value for ALERT_PURGE = 2678400000
```

serverConfig setParameter

The **serverConfig setParameter** command updates the value of a specified Hyperic Server configuration property. Specify the property with the `--key` qualifier and the new value with the `-value` qualifier.

For example, this command sets Hyperic Server's HQ_ALERTS_ENABLED property to "false":

```
hqapi.sh serverConfig setParameter --key=ALERT_PURGE --value=false
```

The results returned are:

```
Successfully updated HQ configuration.
```

Server Configuration Reference

This section lists and defines the Hyperic Server properties that you can configure with the **serverConfig** command.

Hyperic Email Configuration Properties

These properties configure how Hyperic composes and sends notification emails.

- "CAM_BASE_URL" - Corresponds to the **Base URL** property configurable in the **Email Properties** section of the **HQ Server Settings** page. See the definition below.

- "CAM_EMAIL_SENDER" - Corresponds to the **From Email** property configurable in the **Email Properties** section of the **HQ Server Settings** page. See the definition below.

HQ Server email configuration properties are used to form notifications that HQ sends for a fired alert.

Base URL	The address:port where HQ Server listens for web application requests. The initial value of **Base URL** is web application listen port configured when HQ Server was installed, for example:
	`http://Ms-MacBook-Pro-15.local:7080`
	Base URL forms the prefix of an URL to any HQ appends the remainder of the URL that points to the **Al Detail** page for the fired alert. For example:
	`http://Ms-MacBook-Pro-15.local:7080/alerts/Alerts.do?mode=viewAlert&eid=5:10611&a=1643`

From Email Address	The email address listed as the sender of the alert emails. For example: `hq@demo2.hyperic.net`

- "CAM_SMTP_HOST" - Domain name of the host of the mail server used for sending Hyperic email notifications. This property is not configurable in the **HQ Server Settings** page - its original value is configured during Hyperic Server installation.

Data Manager Configuration Properties

These properties control Hyperic Server's metric, alert, and event data management.

Most of these properties correspond to properties displayed in the **Data Manager Properties** section of the **HQ Server Settings** page.

- "DATA_REINDEX_NIGHTLY" - See **Reindex Metric Data Tables Nightly** in the table below. Supply value as "true" or "false"
- "EVENT_LOG_PURGE" - See **Delete Events and Logs Older Than** in the table below. Supply value in milliseconds, for example,"2678400000" for 31 days.
- "ALERT_PURGE" - See **Delete Alerts Older Than** in the table below. Supply value in milliseconds, for example,"2678400000" for 31 days."

These properties control how HQ condenses and purges the contents of the HQ database. Regardless of these settings HQ will retain two years of compressed metric history, but you can control how long detailed metric data is retained. Retaining fewer days of detailed metric data and deleting alerts and other events on a timely basis can improve HQ performance.

Option	Description	Notes
Run Database Maintenance Every	Controls how frequently HQ compresses and archives detailed metric data that is older than the age specified by the following property.	By default, HQ does database maintenance every hour.
Delete Detailed Metric Data Older Than	Controls how many days of detailed metric data HQ retains before compressing it into hourly averages with highs and lows and archiving those values.	The default setting is 2 days. HQ does not support a value greater than 7 days.
Reindex Metric Data Tables Nightly	Controls whether HQ reindexes metric data tables every night. If configured to re-index nightly, HQ re-indexes the tables around midnight.	
Delete Alerts Older Than	Controls how long HQ stores alert event data.	The default value is 31 days.
Delete Events and Logs Older Than	Controls how long HQ stores other HQ event and log data.	The default value is 31 days.

Note: You must restart the Hyperic Server for data management changes to take effect.

Global Alert Properties

These properties are global controls over alert processing. For more information, see Manage Alert and Notification Volume.

The properties correspond to the properties displayed in the **Global Alert Properties** section of the **HQ Server Settings** page.

- "HQ_ALERTS_ENABLED"" - See the definition for **Alerts** below.
- "HQ_ALERT_NOTIFICATIONS_ENABLED" - See the definition for **Alert Notifications** below.
- "HQ_HIERARCHICAL_ALERTING_ENABLED" - See the definition for **Hierarchical Alerting** below.

These properties enable immediate and global control of alert processing.

- **Alerts** — Disable or enable all alert definitions for all resources immediately. Disabling stops any alerts from firing; notifications defined in escalations that are currently in progress will be completed.
- **Alert Notifications** — Disable or enable alert notifications for all resources immediately. Disabling stops all notifications, include those for alerts with escalations currently in progress.
- **Hierarchical Alerting** * — In vFabric Hyperic, this setting controls whether alerts are evaluated using the hierarchical alerting method. When hierarchical alerting is enabled, before firing an alert for a resource, Hyperic considers the availability and alert status of the resource's parent. The purpose of hierarchical alerting is to avoid firing alerts for every resource affected by a single root cause. For more information, see Hierarchical Alerting Prevents a Cascade of Alerts in Resource Hierarchies.

Note: You can extend the effect of hierarchical alerting in vFabric Hyperic by configuring the relationship between a network device or virtual host and the platforms that depend on it using the **Network and Host Dependency Manager** available in the "Plugins" section of the **Administration** tab. For more information see Configure Network Host Dependencies for Hierarchical Alerting.

Notification Throttling Configuration Properties

Available only in vFabric Hyperic

These properties are global controls over alert processing. For more information, see Manage Alert and Notification Volume.

These properties correspond to the properties displayed in the **Notification Throttling Configuration Properties** section of the **HQ Server Settings** page.

Server Property	Description	Example Value
"HQ_ALERT_THRESHOLD"		"0"
"HQ_ALERT_THRESHOLD_EMAILS"		""

Automatic Baseline Configuration Properties

Available only in vFabric Hyperic

These properties correspond to the properties displayed in the **Automatic Baseline Configuration Properties** section of the **HQ Server Settings** page.

- "CAM_BASELINE_DATASET" - See **Baseline Dataset** in the table below. Supply value in milliseconds, for example, "604800000" for 7 day
- "CAM_BASELINE_FREQUENCY" - See **Baseline Frequency** in the table below. Supply value in milliseconds, for example, "259200000" for three days.
- "CAM_BASELINE_MINSET" "40" - See **Baseline Minimum Data Points** in the table below.

In vFabric Hyperic, these properties control the baselining process. Changing the data set used to calculate baselines can affect baseline accuracy.

Server Setting	Description	Default
Baseline Frequency	The frequency with which Hyperic calculates a baseline for each metric.	3 days
Baseline Dataset	The time range of metric data used in calculating the baseline.	7 days
Baseline Minimum Data Points	The minimum number of data points used in calculating a baseline.	40
Track Out-of-Bounds Metrics	Controls whether or not Hyperic tracks out-of-bounds metrics — measurements that are greater than expected high range for a metric, or less than the expected low range for a metric.	off

LDAP Configuration Properties

Available only in vFabric Hyperic

Most of the LDAP properties correspond to properties displayed in the **LDAP Configuration Properties** section of the **HQ Server Settings** page.

- "CAM_LDAP_BASE_DN" - See definition of **Search Base** below.
- "CAM_LDAP_BIND_DN" - See definition of **Username** below.
- "CAM_LDAP_BIND_PW" - See definition of **Password** below.
- "CAM_LDAP_FILTER" - See definition of **Search Filter** below.
- "CAM_LDAP_LOGIN_PROPERTY" - See definition of **Login Property** below.
- "CAM_LDAP_NAMING_PROVIDER_URL - See definition of **URL** below.
- "CAM_LDAP_PROTOCOL" - See definition of **SSL** below.

Couldn't find a page to include called: LdapProps

Kerberos Configuration Properties

Available only in vFabric Hyperic

These properties correspond to the properties displayed in the **Keberos Properties** section of the **HQ Server Settings** page.

Server Property	Description	Example Value
"KERBEROS_DEBUG"	Enables debug logging	"true" or "false"
"KERBEROS_KDC"	KDC	

"KERBEROS_REALM"	Identifies the Kerberos Realm

SNMP Properties

These properties correspond to the properties displayed in the **SNMP Properties** section of the **HQ Server Settings** page. For definitions and usage see the section for the appropriate SNMP version below.

Server Property	Corresponds To
"SNMP_AGENT_ADDRESS"	Agent Address
"SNMP_AUTH_PASSPHRASE"	Authentication Passphrase.
"SNMP_AUTH_PROTOCOL"	Authentication Protocol
"SNMP_COMMUNITY"	Community
"SNMP_CONTEXT_NAME"	Context Name
"SNMP_ENGINE_ID"	Context Engine ID
"SNMP_DEFAULT_NOTIFICATION_MECHANISM"	
"SNMP_ENTERPRISE_OID"	Enterprise OID
"SNMP_GENERIC_ID"	Generic ID
"SNMP_PRIVACY_PROTOCOL"	Privacy Protocol
"SNMP_PRIV_PASSPHRASE"	Privacy Passphrase
"SNMP_SECURITY_NAME"	Security Name
"SNMP_SPECIFIC_ID"	Specific ID
"SNMP_TRAP_OID"	SNMP Trap OID
"SNMP_VERSION"	SNMP Protocol Version

Configure HQ Server for SNMP v1

Select "v1" from the **SNMP Protocol Version** pulldown and supply values for the properties defined in the table below.

The table below defines the properties for configuring HQ Server for SNMP V1 communications with an NMS.

Configuration Option	Description	Allowable Values
SNMP Trap OID	The OID of the notification to be sent. Supplies the value of `snmpTrapOID.0` - the second varbind in a trap or inform that HQ Server generates. (The first varbind is `SysUpTime.0`.)	
Default Notification Mechanism	Your selection governs the notification type that will appear as the default notification type option in the "Notification Mechanism" pulldown list that is presented in configuration dialogs when user configures an SNMP notification as an alert action, or as a step in an escalation.	For v1 of the SNMP protocol, choose V1 Trap. This is the only trap type you can generate for SNMP v1.

Enterprise OID	Enterprise OID.	
Community	The community name to be sent with the trap.	
Generic ID	Single digit identifier of the trap type.	0 - coldStart 1 - warmStart 2 - linkDown 3 - linkUp 4 - authenticationFailure 5 - egpNeighborLoss 6 - enterpriseSpecific
Specific ID	The specific trap code for an enterprise-specific trap (when **Generic ID** is set to to 6).	
Agent Address	Address of the managed object that generates the trap.	

Configure HQ Server for SNMP v2c

Configuration Option	Description	Allowable Values
SNMP Trap OID	The OID of the notification to be sent. Supplies the value of `snmpTrapOID.0` - the second varbind in a trap or inform that HQ Server generates. (The first varbind is `SysUpTime.0`.)	
Default Notification Mechanism	Specifies the default notification type that will appear in configuration dialogs when an authorized user configures an SNMP notification as an alert action, or as a step in an escalation. This choice simply defines the default option - the user configuring an alert action or escalation can choose a different message type.	▪ V1 Trap ▪ V2c Trap ▪ Inform
Community	The community name to be sent with the trap.	

Configure HQ Server for SNMP v3

This section lists the properties for enabling vFabric Hyperic to send SNMP notifications to an NMS. When HQ is so enabled, you can use SNMP notifications in alert definitions - as alert actions and escalation steps.

Configuration Option	Description	Allowable Values
SNMP Trap OID	The OID of the notification to be sent. Supplies the value of `snmpTrapOID.0` - the second varbind in a trap or inform that HQ Server generates. (The first varbind is `SysUpTime.0`.)	
Default Notification Mechanism	Specifies the default notification type that will appear in configuration dialogs when an authorized user configures an SNMP notification as an alert action, or as a step in an escalation. This choice simply defines the default option - the user configuring an alert action or escalation can choose a different message type.	▪ V1 Trap ▪ V2c Trap ▪ Inform

Security Name	The username HQ's SNMP agent should use when sending notifications to the NMS.	Required.
Local Engine ID	ID of HQ's SNMP agent; this value appears automatically, and is not user-configurable.	
Auth Protocol	The SNMP authentication protocol HQ Server should use for communications with the NMS.	▪ none ▪ MD5 ▪ SHA
Auth Passphrase	The SNMP authorization passphrase configured for use when communication with the NMS.	
Privacy Protocol	The SNMP Privacy Protocol HQ Server should use for communication with the NMS.	▪ none ▪ DES ▪ 3DES ▪ AES-128, ▪ AES-192 ▪ AES-256
Privacy Passphrase	The SNMP privacy passphrase configured for use when communication with the NMS.	
Context Engine ID	The EngineID of the NMS. This, along with Context Name, identifies the SNMP context for accessing management data.	Required for v1 and v2c traps. Do not supply for Inform.
Context Name	The name of the SNMP context that provides access to management information on the NMS. A context is identified by the Context Name and Context Engine ID.	

Server Configuration Reference

Hyperic Email Configuration Properties

These properties configure how Hyperic composes and sends notification emails.

- "CAM_BASE_URL" - Corresponds to the **Base URL** property configurable in the **Email Properties** section of the **HQ Server Settings** page. See the definition below.
- "CAM_EMAIL_SENDER" - Corresponds to the **From Email** property configurable in the **Email Properties** section of the **HQ Server Settings** page. See the definition below.

HQ Server email configuration properties are used to form notifications that HQ sends for a fired alert.

Base URL	The address:port where HQ Server listens for web application requests. The initial value of **Base URL** is web application listen port configured when HQ Server was installed, for example:
	`http://Ms-MacBook-Pro-15.local:7080`
	Base URL forms the prefix of an URL to any HQ appends the remainder of the URL that points to the **Al** **Detail** page for the fired alert. For example:
	`http://Ms-MacBook-Pro-15.local:7080/alerts/Alerts.do?mode=viewAlert&eid=5:10611&a=1643`

From Email Address	The email address listed as the sender of the alert emails. For example: `hq@demo2.hyperic.net`

- "CAM_SMTP_HOST" - Domain name of the host of the mail server used for sending Hyperic email notifications. This property is not configurable in the **HQ Server Settings** page - its original value is configured during Hyperic Server installation.

Data Manager Configuration Properties

These properties control Hyperic Server's metric, alert, and event data management.

Most of these properties correspond to properties displayed in the **Data Manager Properties** section of the **HQ Server Settings** page.

- "DATA_REINDEX_NIGHTLY" - See **Reindex Metric Data Tables Nightly** in the table below. Supply value as "true" or "false"
- "EVENT_LOG_PURGE" - See **Delete Events and Logs Older Than** in the table below. Supply value in milliseconds, for example,"2678400000" for 31 days.
- "ALERT_PURGE" - See **Delete Alerts Older Than** in the table below. Supply value in milliseconds, for example,"2678400000" for 31 days."

These properties control how HQ condenses and purges the contents of the HQ database. Regardless of these settings HQ will retain two years of compressed metric history, but you can control how long detailed metric data is retained. Retaining fewer days of detailed metric data and deleting alerts and other events on a timely basis can improve HQ performance.

Option	Description	Notes
Run Database Maintenance Every	Controls how frequently HQ compresses and archives detailed metric data that is older than the age specified by the following property.	By default, HQ does database maintenance every hour.
Delete Detailed Metric Data Older Than	Controls how many days of detailed metric data HQ retains before compressing it into hourly averages with highs and lows and archiving those values.	The default setting is 2 days. HQ does not support a value greater than 7 days.
Reindex Metric Data Tables Nightly	Controls whether HQ reindexes metric data tables every night. If configured to re-index nightly, HQ re-indexes the tables around midnight.	
Delete Alerts Older Than	Controls how long HQ stores alert event data.	The default value is 31 days.
Delete Events and Logs Older Than	Controls how long HQ stores other HQ event and log data.	The default value is 31 days.

Note: You must restart the Hyperic Server for data management changes to take effect.

Global Alert Properties

These properties are global controls over alert processing. For more information, see Manage Alert and Notification Volume.

The properties correspond to the properties displayed in the **Global Alert Properties** section of the **HQ Server Settings** page.

- "HQ_ALERTS_ENABLED"" - See the definition for **Alerts** below.
- "HQ_ALERT_NOTIFICATIONS_ENABLED" - See the definition for **Alert Notifications** below.
- "HQ_HIERARCHICAL_ALERTING_ENABLED" - See the definition for **Hierarchical Alerting** below.

These properties enable immediate and global control of alert processing.

- **Alerts** — Disable or enable all alert definitions for all resources immediately. Disabling stops any alerts from firing; notifications defined in escalations that are currently in progress will be completed.
- **Alert Notifications** — Disable or enable alert notifications for all resources immediately. Disabling stops all notifications, include those for alerts with escalations currently in progress.
- **Hierarchical Alerting** * — In vFabric Hyperic, this setting controls whether alerts are evaluated using the hierarchical alerting method. When hierarchical alerting is enabled, before firing an alert for a resource, Hyperic considers the availability and alert status of the resource's parent. The purpose of hierarchical alerting is to avoid firing alerts for every resource affected by a single root cause. For more information, see Hierarchical Alerting Prevents a Cascade of Alerts in Resource Hierarchies.

Note: You can extend the effect of hierarchical alerting in vFabric Hyperic by configuring the relationship between a network device or virtual host and the platforms that depend on it using the **Network and Host Dependency Manager** available in the "Plugins" section of the **Administration** tab. For more information see Configure Network Host Dependencies for Hierarchical Alerting.

Notification Throttling Configuration Properties

Available only in vFabric Hyperic

These properties are global controls over alert processing. For more information, see Manage Alert and Notification Volume.

These properties correspond to the properties displayed in the **Notification Throttling Configuration Properties** section of the **HQ Server Settings** page.

Server Property	Description	Example Value
"HQ_ALERT_THRESHOLD"		"0"
"HQ_ALERT_THRESHOLD_EMAILS"		""

Automatic Baseline Configuration Properties

Available only in vFabric Hyperic

These properties correspond to the properties displayed in the **Automatic Baseline Configuration Properties** section of the **HQ Server Settings** page.

- "CAM_BASELINE_DATASET" - See **Baseline Dataset** in the table below. Supply value in milliseconds, for example, "604800000" for 7 day
- "CAM_BASELINE_FREQUENCY" - See **Baseline Frequency** in the table below. Supply value in milliseconds, for example, "259200000" for three days.
- "CAM_BASELINE_MINSET" "40" - See **Baseline Minimum Data Points** in the table below.

In vFabric Hyperic, these properties control the baselining process. Changing the data set used to calculate baselines can affect baseline accuracy.

Server Setting	Description	Default
Baseline Frequency	The frequency with which Hyperic calculates a baseline for each metric.	3 days
Baseline Dataset	The time range of metric data used in calculating the baseline.	7 days
Baseline Minimum Data Points	The minimum number of data points used in calculating a baseline.	40
Track Out-of-Bounds Metrics	Controls whether or not Hyperic tracks out-of-bounds metrics — measurements that are greater than expected high range for a metric, or less than the expected low range for a metric.	off

LDAP Configuration Properties

Available only in vFabric Hyperic

Most of the LDAP properties correspond to properties displayed in the **LDAP Configuration Properties** section of the **HQ Server Settings** page.

- "CAM_LDAP_BASE_DN" - See definition of **Search Base** below.
- "CAM_LDAP_BIND_DN" - See definition of **Username** below.
- "CAM_LDAP_BIND_PW" - See definition of **Password** below.
- "CAM_LDAP_FILTER" - See definition of **Search Filter** below.
- "CAM_LDAP_LOGIN_PROPERTY" - See definition of **Login Property** below.
- "CAM_LDAP_NAMING_PROVIDER_URL - See definition of **URL** below.
- "CAM_LDAP_PROTOCOL" - See definition of **SSL** below.

Couldn't find a page to include called: LdapProps

Kerberos Configuration Properties

Available only in vFabric Hyperic

These properties correspond to the properties displayed in the **Keberos Properties** section of the **HQ Server Settings** page.

Server Property	Description	Example Value
"KERBEROS_DEBUG"	Enables debug logging	"true" or "false"
"KERBEROS_KDC"	KDC	

"KERBEROS_REALM"	Identifies the Kerberos Realm

SNMP Properties

These properties correspond to the properties displayed in the **SNMP Properties** section of the **HQ Server Settings** page. For definitions and usage see the section for the appropriate SNMP version below.

Server Property	Corresponds To
"SNMP_AGENT_ADDRESS"	Agent Address
"SNMP_AUTH_PASSPHRASE"	Authentication Passphrase.
"SNMP_AUTH_PROTOCOL"	Authentication Protocol
"SNMP_COMMUNITY"	Community
"SNMP_CONTEXT_NAME"	Context Name
"SNMP_ENGINE_ID"	Context Engine ID
"SNMP_DEFAULT_NOTIFICATION_MECHANISM"	
"SNMP_ENTERPRISE_OID"	Enterprise OID
"SNMP_GENERIC_ID"	Generic ID
"SNMP_PRIVACY_PROTOCOL"	Privacy Protocol
"SNMP_PRIV_PASSPHRASE"	Privacy Passphrase
"SNMP_SECURITY_NAME"	Security Name
"SNMP_SPECIFIC_ID"	Specific ID
"SNMP_TRAP_OID"	SNMP Trap OID
"SNMP_VERSION"	SNMP Protocol Version

Configure HQ Server for SNMP v1

Select "v1" from the **SNMP Protocol Version** pulldown and supply values for the properties defined in the table below.

The table below defines the properties for configuring HQ Server for SNMP V1 communications with an NMS.

Configuration Option	Description	Allowable Values
SNMP Trap OID	The OID of the notification to be sent. Supplies the value of `snmpTrapOID.0` - the second varbind in a trap or inform that HQ Server generates. (The first varbind is `SysUpTime.0`.)	
Default Notification Mechanism	Your selection governs the notification type that will appear as the default notification type option in the "Notification Mechanism" pulldown list that is presented in configuration dialogs when user configures an SNMP notification as an alert action, or as a step in an escalation.	For v1 of the SNMP protocol, choose V1 Trap. This is the only trap type you can generate for SNMP v1.

Enterprise OID	Enterprise OID.	
Community	The community name to be sent with the trap.	
Generic ID	Single digit identifier of the trap type.	0 - coldStart 1 - warmStart 2 - linkDown 3 - linkUp 4 - authenticationFailure 5 - egpNeighborLoss 6 - enterpriseSpecific
Specific ID	The specific trap code for an enterprise-specific trap (when **Generic ID** is set to to 6).	
Agent Address	Address of the managed object that generates the trap.	

Configure HQ Server for SNMP v2c

Configuration Option	Description	Allowable Values
SNMP Trap OID	The OID of the notification to be sent. Supplies the value of `snmpTrapOID.0` - the second varbind in a trap or inform that HQ Server generates. (The first varbind is `SysUpTime.0`.)	
Default Notification Mechanism	Specifies the default notification type that will appear in configuration dialogs when an authorized user configures an SNMP notification as an alert action, or as a step in an escalation. This choice simply defines the default option - the user configuring an alert action or escalation can choose a different message type.	▪ V1 Trap ▪ V2c Trap ▪ Inform
Community	The community name to be sent with the trap.	

Configure HQ Server for SNMP v3

This section lists the properties for enabling vFabric Hyperic to send SNMP notifications to an NMS. When HQ is so enabled, you can use SNMP notifications in alert definitions - as alert actions and escalation steps.

Configuration Option	Description	Allowable Values
SNMP Trap OID	The OID of the notification to be sent. Supplies the value of `snmpTrapOID.0` - the second varbind in a trap or inform that HQ Server generates. (The first varbind is `SysUpTime.0`.)	
Default Notification Mechanism	Specifies the default notification type that will appear in configuration dialogs when an authorized user configures an SNMP notification as an alert action, or as a step in an escalation. This choice simply defines the default option - the user configuring an alert action or escalation can choose a different message type.	▪ V1 Trap ▪ V2c Trap ▪ Inform

Security Name	The username HQ's SNMP agent should use when sending notifications to the NMS.	Required.
Local Engine ID	ID of HQ's SNMP agent; this value appears automatically, and is not user-configurable.	
Auth Protocol	The SNMP authentication protocol HQ Server should use for communications with the NMS.	▪ none ▪ MD5 ▪ SHA
Auth Passphrase	The SNMP authorization passphrase configured for use when communication with the NMS.	
Privacy Protocol	The SNMP Privacy Protocol HQ Server should use for communication with the NMS.	▪ none ▪ DES ▪ 3DES ▪ AES-128, ▪ AES-192 ▪ AES-256
Privacy Passphrase	The SNMP privacy passphrase configured for use when communication with the NMS.	
Context Engine ID	The EngineID of the NMS. This, along with Context Name, identifies the SNMP context for accessing management data.	Required for v1 and v2c traps. Do not supply for Inform.
Context Name	The name of the SNMP context that provides access to management information on the NMS. A context is identified by the Context Name and Context Engine ID.	

HQApi user command 23

*Topics marked with * relate to features available only in vFabric Hyperic.*

Related Topics

This page provides information for running an HQ API from the command line. Related topics include:

- vFabric Hyperic Web Services API (see page 6) - brief introduction to the API.
- HQApi Command-Line Tools (see page 11) - how to get started with the command line tools.
- HQApi Java API (see page 141) - about accessing the APIs programmatically.

Functionality

The **user** command has options to list, update, and create users.

Command Options

list

list command output

The **list** command lists attributes for one or all HQ users.

```
UsersResponse
     User
          passwordHash
          htmlEmail
          active
          phoneNumber
          SMSAddress
          emailAddress
          department
          lastName
          firstName
          name
          id
```

list Command Qualifiers

--id	Only list the user with the given id
--name	Only list the user with the given name.

sync

The `usersResponse` object that is returned by the "list" option can be edited and passed to the "sync" command create or update users. Any attribute that is changed in the User element will be updated in HQ. Any additional User elements will create the User in HQ. When creating new Users, omit the id attribute as HQ will assign the user an id automatically.

Examples

user list

```
$ ./bin/hqapi.sh user list
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<UsersResponse>
    <Status>Success</Status>
    <User passwordHash="" htmlEmail="false" active="false" phoneNumber="415-541-4975" SMSAddress=""

          emailAddress="localhost" department="" lastName="User" firstName="Guest" name="guest"
id="2"/>
    <User passwordHash="XfLzwfNQujo/CxxaYX3OCg==" htmlEmail="false" active="true" phoneNumber=""
SMSAddress=""
          emailAddress="localhost" department="" lastName="Administrator" firstName="HQ"
name="hqadmin" id="1"/>
</UsersResponse>
```

user sync

The first command in this example writes the `UsersResponse` object to a file called "users.xml". The second command writes the contents of "users.xml" to HQ.

```
$ ./bin/hqapi.sh user list > users.xml
...
...
$ cat users.xml | ./bin/hqapi.sh user sync
Successfully synced 2 users.
```

HQApi Java API 24

The Java API is thoroughly documented in the JavaDocs that are included in the client download. Usage of the Java API requires that hqapi1.jar as well as all libraries in the lib directory be included within the classpath.

API Classes

The entry point to the Java API is the *HQApi* class. The constructor for this class takes a series of arguments that describes the connection to be made to HQ. From the HQApi object, APIs to all HQ's subsystems may be accessed.

Return Objects

All APIs return response Objects that extend the *Response* class. This class provides the caller information on the success or failure of the API call via the *getStatus()* method. This can return one of *ResponseStatus.SUCCESS* or *ResponseStatus.FAILURE*. On the case of failure, additional information on the reason for the failure may be obtained through *getError()*. See the JavaDocs on the *Response* class for more information.

In general API calls will return 1 of 3 different types of responses:

Response Object	Description
StatusResponse	Think of this as a *void* method signature. The *StatusResponse* simply carries with it the base *Response* information on the success or failure of the API call.
Single Object Response	These methods return an object similar to the *StatusResponse*, but it also contains a single entity. Examples of this include *UserResponse* and *ResourceResponse*.
Multiple Object Response	Again, this is similar to the single object response, but instead of a single object, a *List* of objects is returned. Examples of this include *UsersResponse* and *ResourcesResponse*.

Code Example

Example: Create a new User

```
package org.hyperic.hq.hqapi1.test;

import org.hyperic.hq.hqapi1.HQApi;
import org.hyperic.hq.hqapi1.UserApi;
import org.hyperic.hq.hqapi1.types.User;
import org.hyperic.hq.hqapi1.types.UserResponse;
import org.hyperic.hq.hqapi1.types.Response;
import org.hyperic.hq.hqapi1.types.ResponseStatus;

public class Test {

    private static void assertSuccess(Response r) {
        if (!r.getStatus().equals(ResponseStatus.SUCCESS)) {
            System.err.println("Error :" + r.getError().getReasonText());
        } else {
            System.out.println("Command completed successfully");
        }
    }

    public static void main(String[] args) throws Exception {

        HQApi api = new HQApi("localhost", 7080, false, "hqadmin", "hqadmin");
        UserApi userApi = api.getUserApi();

        User newUser = new User();
        newUser.setName("hyperic");
        newUser.setFirstName("Hyperic");
        newUser.setLastName("HQ");
        newUser.setEmailAddress("support@hyperic.com");
        newUser.setHtmlEmail(true);

        UserResponse status = userApi.createUser(newUser, "hyperic");
        assertSuccess(status);
    }
}
```

Code Samples

There are no formal code samples included in the client download, however the test suite and tools packages provide many examples of API usage.

- Test Suite — "https://github.com/hyperic/hqapi/"
- Tools — "https://github.com/hyperic/hqapi/"