Memset API Documentation

Release 0.20.1

Memset Ltd

August 23, 2025

CONTENTS

CHAPTER

ONE

CONTENTS

1.1 Introduction

[**This API and its documentation are in beta and subject to change before final release**]

Here is the API for Memset systems.

This documentation can be accessed in a number of ways:

- HTML (split into multiple pages, hosted on our website.)
- PDF (downloadable.)

Some parts of this documentation are generated dynamically and may change over time. The online version is therefore recommended.

This document was generated August 23, 2025.

1.1.1 Types

There is a single API specification that is implemented in several interfaces. This section describes a set of types and their mapping to the different interfaces.

This API specification uses the following data types which you will see in the documentation:

- String
- List
- Dictionary
- Boolean
- Float
- Integer
- Date

The API types are mapped to the following implementation types:

API Spec	XML-RPC	JSON	HTTP GET/POST
String	string	String	String
List	array	Array	N/A
Dictionary	struct	Object	N/A
Boolean	boolean	true or false	String
Float	double	Number	String
Integer	int	Number	String
Date	dateTime.iso8601	String	String

1.1.2 Authentication

Authentication for all methods supports standard HTTP basic authorisation over HTTPS (using the HTTP_AUTHORIZATION header).

Your api_key (32 digit hex string) should be provided as the username and the password can be anything.

You can also pass your api_key in as an extra POST or GET parameter named `api_key` when using GET or POST over HTTPS.

1.1.3 Access methods

JSON over HTTPS

GET or POST to this URL. Replace METHOD_NAME with the name of the method from the documentation, e.g. "server.info":

```
https://api.memset.com/v1/json/METHOD_NAME
```

You may also supply the "name" parameter as another part of the URL:

```
https://api.memset.com/v1/json/METHOD_NAME/SERVICE_NAME
```

e.g.:

```
https://api.memset.com/v1/json/server.info/myserver1
```

The authentication may be provided as basic authentication, or as an api_key parameter. See the *cURL Example* for more information and examples.

Parameters may be passed in as GET or POST parameters encoded as strings. Booleans are "0" or "1" and dates are in ISO 8601 format "YYYY-MM-DD HH:MM:SS". It isn't possible to pass List or Dictionary types so those API methods won't work without using JSON encoding.

To pass more complicated parameters use the `parameters` parameter. Its contents should be a JSON encoded dictionary which contains the parameters to the API call. See the *cURL Example* for a walk through.

Errors are returned as HTTP status codes with a JSON encoded return. For example, this was returned with a `404` HTTP status code:

```
{
    "error_type": "ApiErrorDoesNotExist",
    "error_code": 3,
    "error": "Couldn't find service with name 'BADSERVERNAME'"
}
```

See *Errors* for the mapping between HTTP status codes and the various errors returned.

XML-RPC

Post your XMLRPC to this URL using the api_key as the username with BASIC authentication and any value as the password:

```
https://api.memset.com/v1/xmlrpc
```

Some XMLRPC clients like to have the authentication in the URI, in which case write it like this:

```
https://32_hex_digit_api_key:x@api.memset.com/v1/xmlrpc
```

See the examples for a comprehensive selection of XMLRPC in many different languages.

See *Errors* for the mapping to XMLRPC error codes.

JSON-RPC

The API also supports JSON-RPC v 2.0

Post your JSON-RPC to this URL using the api_key as the username with BASIC authentication and any value as the password:

```
https://api.memset.com/v1/jsonrpc
```

Some JSON-RPC clients like to have the authentication in the URI, in which case write it like this:

```
https://32_hex_digit_api_key:x@api.memset.com/v1/jsonrpc
```

The Server only supports named parameters - not argument lists. Batch mode is supported, but only up to 100 items in the batch.

See the implementations page for libraries to use JSON-RPC and see the *Python JSON-RPC Example* example for some code.

See *Errors* for the mapping to JSON-RPC error codes.

1.2 Methods

1.2.1 API Key Methods

API for API keys

The API keys are for accessing the Memset API programatically not using a username and password. Their scope can be limited on creation so that it is possible to create API keys with just a single purpose.

1.2.2 Scopes

API keys can be scoped with “name”, “method” or with both.

API keys created with “name” scopes can only be called with a parameter called “name” with one of the values supplied in the scope when it was created.

API keys created with “method” scopes can only be used to call those named methods and no others.

For example if an API key was created with the scopes:

```
{
    "name": [ "server1", "server2" ]
}
```

Then it could only be used to access methods which take a "name" parameter and with that set to either "server1" or "server2"

If an API key was created with the scopes:

```
{
    "method": [ "server.reboot" ]
}
```

Then it could only be used to call the "server.reboot" method.

If an API key was created with the scopes:

```
{
    "method": [ "server.reboot" ],
    "name": [ "server1", "server2" ]
}
```

Then it could only be used to call the "server.reboot" method on "server1" or "server2".

apikey.**add_scope**()
Adds a single scope to an API key

Parameters

- **id** (*String*) – The API key. API Keys are 32 hex digits.
- **scope_name** (*String*) – The name of the API scope. Acceptable values: 'method', 'name'.
- **value** (*String*) – The value of the API scope. Ensure this value has at most 50 characters.

Returns Dictionary: as described in `apikey.info()`.

apikey.**create**()
Creates a new API key for this account.

Parameters

- **scopes** (*Dictionary, Optional*) – The scopes dictionary as described in `apikey.info()`.
- **comment** (*String, Optional*) – User comment. Ensure this value has at most 255 characters.

Returns a dictionary containing the new api key as described in `apikey.info()`.

apikey.**delete**()
Deletes an API key.

Parameters **id** (*String*) – The API key. API Keys are 32 hex digits.

Returns an empty dictionary.

apikey.**delete_scope**()
Deletes a single scope to an API key. Doesn't raise an error if it didn't exist.

Parameters

- **id** (*String*) – The API key. API Keys are 32 hex digits.
- **scope_name** (*String*) – The name of the API scope. Acceptable values: 'method', 'name'.
- **value** (*String*) – The value of the API scope. Ensure this value has at most 50 characters.

Returns Dictionary: as described in `apikey.info()`.

apikey.**info**()
Describes the API key.

Parameters **id** (*String*) – The API key. API Keys are 32 hex digits.

Returns

a dictionary with the following keys:

key String: 32 digit hex which is the api_key.

scopes Dictionary: A dictionary of Lists of String. The keys are the scope names ("method" or "name") and the values are lists of valid entries for those scopes

comment String: User comment associated to the key (if any).

created Date: Date when the key was created.

apikey.**list**()
List all the API keys for this account.

Returns a list of dictionaries where each dictionary is as described in apikey.info().

1.2.3 Bandwidth Methods

API Methods for dealing with bandwidth information

bandwidth.**info**()
Return bandwidth information.

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns

a dictionary with the following keys:

service_type

Dictionary: a dictionary with the following keys:

bytes_in: Float: bytes in for the current month.

bytes_out: Float: bytes out for the current month.

daily_rate: Float: average total bandwidth usage per day.

predicted: Float: predicted total bandwidth usage by the end of the month.

connection_type String: Optional: 'metered', 'unmetered' or 'inarrears'.

max_burst Float: Optional: maximum burst in Mbps.

remaining Float: Optional: for metered connections, remaining bandwidth for current month.

inclusive_transfer Float: Optional: monthly data transfer in GB.

bandwidth_bank Float: Optional: extra transfer bank in GB per month.

contention Float: Optional: contention ratio.

ips List: Optional: a list of dictionaries. See below for a description of the dictionary keys.

The **ips** attribute is a list of dictionaries with the following keys:

ip_address

Dictionary: a dictionary with the following keys:

bytes_in: Float: bytes in for the current day.

bytes_out: Float: bytes out for the current day.

Some services may return several service types (i.e. Memstore will return cloud storage and CDN bandwidth information).

Notes:

- The fields *connection_type*, *max_burst* and *ips* are only available in servers.
- The fields *remaining*, *inclusive_transfer* and *bandwidth_bank* are only available in servers using 'metered' connection type.
- The field *contention* is only available in servers using 'unmetered' connection type.

bandwidth.**monthly_usage**()

Return monthly bandwidth usage information in a given year.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **year** (*Integer, Optional*) – Period of data starting on 1st January of provided year. Ensure this value is less than or equal to 2100. Ensure this value is greater than or equal to 2000.

Returns

a dictionary with following keys:

service_type

List: a list of dictionaries with following keys:

date Date: period of the record (yyyy-mm).

bytes_in Float: bytes in for that date.

bytes_out Float: bytes out for that date.

Some services, such as Memstore, will return bandwidth information for several service types (for example: the cloud storage itself and the CDN service).

If the year is not provided, current year will be used.

bandwidth.**usage**()

Return bandwidth usage information in last *period*.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **period** (*String, Optional*) – Period Acceptable values: 'day', 'week', 'month', 'quarter', 'halfyear', 'year'.

Returns

a dictionary with following keys:

service_type

List: a list of dictionaries with following keys:

date Date: period of the record.

bps_in Float: bits in per second for that date.

bps_out Float: bits out per second for that date.

Some services, such as Memstore, will return bandwidth information for several service types (for example: the cloud storage itself and the CDN service).

If the period is not provided, 'day' will be used.

1.2.4 Cluster Methods

API Methods for clusters.

`cluster.info()`
Return information for the specified cluster.

Parameters **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns

a dictionary with the following keys:

name String: Name of the cluster.

enabled: Boolean: Whether the cluster is enabled or not.

private_class_c: String: The network address (subnet mask 255.255.255.0) of the VLAN.

vlan_id: Integer: The VLAN id the cluster is in.

`cluster.list()`
Returns a list of clusters.

Parameters **enabled** (*Boolean, Optional*) – Only return enabled/disabled clusters.

Returns a list of dictionaries where each is as described in `cluster.info()`

1.2.5 Cluster Real Service Methods

API methods for cluster real services.

`cluster.real_service.disable()`
Disable the Cluster Server in the Cluster Service.

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – Name of the cluster service.
- **server_name** (*String*) – Name of the cluster server.

Returns If the server is enabled, a dictionary as described in `job.status()`.

Raises May raise `ApiErrorDoesNotExist` if the Cluster, Cluster Service or server does not exist.

`cluster.real_service.disable_fallback()`
Unset the Real Service as fallback.

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – Name of the cluster service.
- **server_name** (*String*) – Name of the cluster server.

Returns If in fallback, a dictionary as described in `job.status()`.

Raises ApiErrorDoesNotExist if the Cluster, Cluster Service or Cluster Server does not exist (in the cluster/account).

When in fallback mode, the real service is used when all other real services are down. Only one Real Service can be in fallback mode.

`cluster.real_service.`**`enable`**`()`

Enable the Cluster Server in the Cluster Real Service.

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – Name of the cluster service.
- **server_name** (*String*) – Name of the cluster server.

Returns If the server is disabled, a dictionary as described in `job.status()`.

Raises May raise `ApiErrorDoesNotExist` if the Cluster, Cluster Service or server does not exist.

`cluster.real_service.`**`enable_fallback`**`()`

Set the Real Service as fallback.

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – Name of the cluster service.
- **server_name** (*String*) – Name of the cluster server.

Returns If not in fallback, a dictionary as described in `job.status()`.

Raises ApiErrorDoesNotExist if the Cluster, Cluster Service or Cluster Server does not exist (in the cluster/account).

Raises ApiErrorPreconditionFailed if a server in the Cluster Service is already set as fallback.

Raises ApiErrorPreconditionFailed if the Real Service is not enabled.

When in fallback mode, the real service is used when all other real services are down. Only one Real Service can be in fallback mode.

`cluster.real_service.`**`info`**`()`

Information about a Cluster Real Service

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – Name of the cluster service.

- **server_name** (*String*) – Name of the cluster server.

Returns

a dictionary with the following keys:

ip_address String: IP Address the service is listening on.

port String: The port the service is on.

weight Integer: The weight for this service.

fallback Boolean: Whether this real service is set as a fallback.

enabled Boolean: Whether this real service is enabled. When disabled, it is not part of the service.

Raises May raise `ApiErrorDoesNotExist` if the cluster, Cluster Service, Cluster Real Service or Cluster Server does exist.

`cluster.real_service.`**`list`**`()`

List the Cluster Real Services for a given Cluster Service.

FIXME: returns

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – Name of the cluster service.

`cluster.real_service.`**`set_weight`**`()`

Change the weight of a server in the service.

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – Name of the cluster service.
- **server_name** (*String*) – Name of the cluster server.
- **weight** (*Integer*) – Weight is an integer specifying the capacity of a server (relative to the others in the pool). A weight of zero means that the server will not receive new jobs.

Returns If new weight is different to existing value, a dictionary as described in `job.status()`.

Raises ApiErrorDoesNotExist if the Cluster, Cluster Service or Cluster Server does not exist (in the cluster/account).

`cluster.real_service.`**`statistics`**`()`

Statistical information about a Cluster Real Service.

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – Name of the cluster service.
- **server_name** (*String*) – Name of the cluster server.

Returns

a dictionary with the following keys:

active Boolean: Whether this real service is active in the cluster service.

active_connections Integer: Current number of active connections

bps_in Integer: Current bits per second in

bps_out Integer: Current bits per second out

connections_per_second Integer: Current number of connections per second

inactive_connections Integer: Current number of inactive connections

last_update Date (with time): The time the status was last updated

persistent_connections Integer persistent connections - Current number of persistent connections

pps_in Integer: Current number of packets per second in

pps_out Integer: Current number of packets per second out

Raises May raise `ApiErrorDoesNotExist` if the cluster, Cluster Service, Cluster Real Service or Cluster Server does exist.

N.B. The data is not live and is thus always at least a few minutes out-of-date. This method can be polled frequently without concern for performance.

1.2.6 Cluster Server Methods

API methods for cluster servers.

`cluster.server.`**`disable`**`()`

Disable the given Cluster Server. When disabled, the cluster server is no longer a member of the cluster.

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **server_name** (*String*) – Name of the cluster server.

Returns If the server is enabled, a dictionary as described in `job.status()`.

Raises May raise `ApiErrorDoesNotExist` if the cluster or Cluster Service does not exist.

`cluster.server.`**`enable`**`()`

Enable the given Cluster Server. When enabled, the cluster server becomes a member of the cluster.

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **server_name** (*String*) – Name of the cluster server.

Returns If the server is disabled, a dictionary as described in `job.status()`.

Raises May raise `ApiErrorDoesNotExist` if the cluster or Cluster Service does not exist.

`cluster.server.`**`info`**`()`

Return information for the specified cluster server.

Parameters **server_name** (*String*) – Name of the cluster server.

Returns

a dictionary with the following keys:

enabled: Boolean: Whether the cluster server is enabled or not. See `cluster.server.disable()` and `cluster.server.enable()`.

server_name: String: The name of the server.

Raises May raise `ApiErrorDoesNotExist` if the cluster_name does not exist (in the account)

`cluster.server.list()`
Returns a list of information about the cluster server in the given cluster.

Parameters **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns a list of dictionaries where each is as described in `cluster.server.info()`

Raises May raise `ApiErrorDoesNotExist` if the cluster_name does not exist (in the account)

1.2.7 Cluster Service Methods

API methods for cluster services.

`cluster.service.disable()`
Disable the given Cluster Service.

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – Name of the cluster service.

Returns If the service is enabled, a dictionary as described in `job.status()`.

Raises May raise `ApiErrorDoesNotExist` if the cluster or Cluster Service does not exist.

`cluster.service.enable()`
Enable the given Cluster Service.

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – Name of the cluster service.

Returns If the service is disabled, a dictionary as described in `job.status()`.

Raises May raise `ApiErrorDoesNotExist` if the cluster or Cluster Service does not exist.

`cluster.service.info()`
Describe a cluster service

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – Name of the cluster service.

Returns

a dictionary with the following keys:

name String: Name of the service.

enabled: Boolean: If the service is enabled or not.

virtual_ip: String: The load-balanced IP (that which should be used in DNS records).

port: Integer: The TCP port number.

service: String: Service type (e.g. http/ftp)

checktype: String: Type of check to perform to determine membership of real servers in the cluster. On means no checking will take place and real servers will always be activated.

checktimeout: Integer: Timeout in seconds for connect, external and ping checks.

checkport: Integer: Number of port to monitor. Sometimes check port differs from service port. 0 is port specified for the real server.

request: String: The request, e.g. 'test.html'. Only for negotiate checktypes.

receive: String: The expected string e.g. 'Test'. Only for negotiate checktypes.

emailalert: String: Email address for alert. An empty string means no alert.

scheduler: String: Scheduler to be used for load balancing:

rr - Robin Robin: distributes jobs equally amongst the available real servers.

wrr - Weighted Round Robin: assigns jobs to real servers proportionally to the real servers' weight. Servers with higher weights receive new jobs first and get more jobs than servers with lower weights. Servers with equal weights get an equal distribution of new jobs.

lc - Least-Connection: assigns more jobs to real servers with fewer active jobs.

wlc - Weighted Least-Connection: assigns more jobs to servers with fewer jobs and relative to the real servers' weight (Ci/Wi). This is the default.

lblc - Locality-Based Least-Connection: assigns jobs destined for the same IP address to the same server if the server is not overloaded and available; otherwise assign jobs to servers with fewer jobs, and keep it for future assignment.

lblcr - Locality-Based Least-Connection with Replication: assigns jobs destined for the same IP address to the least-connection node in the server set for the IP address. If all the nodes in the server set are over loaded, it picks up a node with fewer jobs in the cluster and adds it in the sever set for the target. If the server set has not been modified for the specified time, the most loaded node is removed from the server set, in order to avoid high degree of replication.

dh - Destination Hashing: assigns jobs to servers through looking up a statically assigned hash table by their destination IP addresses.

sh - Source Hashing: assigns jobs to servers through looking up a statically assigned hash table by their source IP addresses.

sed - Shortest Expected Delay: assigns an incoming job to the server with the shortest expected delay. The expected delay that the job will experience is (Ci + 1) / Ui if sent to the ith server, in which Ci is the number of jobs on the the ith server and Ui is the fixed service rate (weight) of the ith server.

nq - Never Queue: assigns an incoming job to an idle server if there is one, instead of waiting for a fast one; if all the servers are busy, it adopts the Shortest Expected Delay policy to assign the job.

persistent: Integer: Number of seconds for persistent client connections.

protocol: String: Protocol to be used.

virtualhost: String: Used when using a negotiate check with HTTP or HTTPS. Sets the host header used in the HTTP request.

forwarding_method: String: Forwarding method for the director to reach the clustered servers.

Raises May raise `ApiErrorDoesNotExist` if the cluster or Cluster Service does exist.

`cluster.service.list()`
List the services in a cluster.

Parameters **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns a list of dictionaries where each is as described in `cluster.service.info()`

Raises May raise `ApiErrorDoesNotExist` if the cluster does not exist.

`cluster.service.status_image()`
View cluster status as a graph, in PNG format.

Parameters

- **cluster_name** (*String*) – Name of the cluster. Cluster names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – Name of the cluster service.

Returns

a dictionary with the following keys:

data: String: A string of binary PNG data.

1.2.8 Create Methods

These API methods allow you to order Memset services.

> **Warning:** Please use the `dry_run=True` argument when you are testing, so that invoices are not generated.

Payment Methods

When creating services using the API, the default payment method for your Memset account will be used. The following constraints on the account payment method apply when using the API to create products:

- A card or billing account must be used to create services which are billed in arrears.
- It is not possible to use PayPal when creating services

If you have consolidated billing enabled for your account, any charges will be added to your next consolidated bill by default. If you wish to be invoiced immediately, you can set `add_to_next_bill=False`, and the default payment method for your Memset account will be used.

See *Payment Method Methods* for more information on the payment methods API.

Operating Systems

When provisioning servers, you must include the required `os` parameter. The `os` parameter can be set to one of our stock operating systems or the name of a Memstore snapshot. For a list of our supported operating systems and the required API names, please see our Operating Systems page for more information.

If you wish to provision a server using a snapshot, please obtain the name of the snapshot using `server.snapshot_list()` and specify the `os_option` as the value for `os`.

See `create.monthly_miniserver()`, `create.hourly_miniserver()` and `create.monthly_fullserver()` for the list of available operating systems for each service type.

Control Panel

We offer cPanel as a chargeable extra for monthly servers, but not hourly Miniservers. Our cPanel image is based on 64-bit CentOS 7.

To specify that you want cPanel, you must specify `os=centos7_cpanel_64`.

Response format for create methods

returns A dictionary with the following keys

sku String: The SKU of the service.

dry_run Boolean: As specified in the API call.

currency String: The currency of the order. You cannot specify the currency in the API request, the account's currency will be used. Currently, the account currency must be changed via the Memset website. Returned value will be one of 'GBP', 'USD', or 'EUR'.

net_setup_price Float: The setup price for the service, net of VAT.

net_monthly_rental Float: The monthly rental for the service, net of VAT.

net_hourly_inarrears_rental_price Float: The hourly rate for this service, net of VAT. This will only be provided when creating hourly miniservers.

net_total Float: The sum of the setup price and any rental payable in advance, net of VAT.

invoice_ref

Integer: The invoice reference for the order. Will not be provided if the method is called with `dry_run=True` or if `add_to_next_bill=True`.

add_to_next_bill

Boolean: If True, any charges will be added to the next consolidated bill. If False, any charges will be invoiced immediately. Will not be provided it the method is called with `dry_run=True`.

period Integer: The period in months of the service. Not provided for services billed in-arrears.

job Dictionary: Provides the status of the job to set up the requested service. Will not be provided when the method is called with `dry_run=True`. See `job.status()` for further details.

Example

The following example shows how to create a monthly miniserver.

First we call `create.monthly_miniserver()` with `dry_run=True`:

```
{
  "sku": "MSVM001",
  "os": "debian_jessie_64",
```

```
  "dry_run": true,
}
```

This will return a response similar to the following:

```
{
  'sku': u'MSVM001'
  'currency': u'GBP',
  'dry_run': True,
  'net_setup_price': 0,
  'net_total': 7.50,
}
```

If we are happy, we can make the same request, but this time with `dry_run=False` so that the service is actually created:

```
{
  "sku": "MSVM001",
  "os": "debian_jessie_64",
  "dry_run": false
}
```

This will give a response similar to the following:

```
{
  'sku': u'MSVM001'
  'currency': u'GBP',
  'dry_run': False,
  'invoice_ref': 100291,
  'job': {'error': False,
          'finished': False,
          'id': '739c99f25ed8d13e48ba94ec6a4a0102',
          'status': 'AWAITING-PAYMENT',
          'type': 'SETUP-NEW-MINISERVER'},
  'net_setup_price': 0,
  'net_total': 7.50
}
```

Note that a job has been created to set up the miniserver, with the status AWAITING-PAYMENT. Once the invoice has been paid, we can check the progress of the setup using the `job.status()` method.

If we provide an invalid parameter, an `ApiErrorBadParameters()` exception will be raised. For example, let's try to provision a Miniserver with Windows and 1GB RAM.:

```
{
  "sku": "MSVM001",
  "os": "win2012serverstd_r2_64",
  "dry_run": false,
}
```

This is an invalid configuration, as Windows miniservers require at least 2 GB of RAM. Therefore we get an exception:

```
ApiErrorBadParameters: os: Please select a valid choice. Microsoft Windows 2012 Server R2, Standard E
```

`create.`**`available`**`()`

Returns the services which can be provisioned using the API.

Returns

a list of dictionaries for each available service with the following keys

sku String: The code used to identify the service.

description String: Description of the service.

type String: The type of service.

default_params Dictionary: The default parameters for this service. The parameters depend on the service type:

miniserver

ram_mb: Integer: The number of megabytes of RAM.

disk_gb Integer: The number of gigabytes of hard disk space.

data_zones List: list of data zone names where the service is available in. The list can be empty if the service is data zone independent. The data zone name can be used in any create method that has a *data_zone* parameter.

`create.`**`extra_bandwidth`**`()`

Adds bandwidth to a server's extra bandwidth bank.

The bandwidth bank is an additional pool of metered bandwidth that will only be used in the event that your monthly inclusive quote is exceeded. It is designed as a "top up tank" in case you have extra usage in any one month.

This is only valid for servers with *metered* `bandwidth_type`.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **amount** (*Integer*) – Amount of bandwidth to be added in GB. Acceptable values are: 50, 100, 250, 500
- **dry_run** (*Boolean*) – If True, then the service is not provisioned but the information is still returned.

`create.`**`hourly_miniserver`**`()`

Warning: Our hourly Miniserver VM products are now deprecated and will be removed in a future API release. For cloud compute instances billed by the hour, please view our new Openstack Public Cloud product: https://www.memset.com/hosting-services/cloud/openstack/

Note that the *os*, *firewall*, *intrusion_detection* and *disk_type* options affect the price per hour.

Parameters

- **connection** (*String, Optional, Deprecated: This parameter is deprecated and will be unsupported in future API releases.*) – For *unmetered* `bandwidth_type` only. Acceptable values are a combination of burst speed and contention ratio e.g. 5mbps_10to1 means 5Mbps burst with contention of 10:1. Dedicated line speed therefore is burst divided by contention, e.g. the dedicated line speed for a line with a connection value of 5mbps_10to1 is 5 / 10 = 0.5Mbps.

sku	connection options

- **monthly_transfer_gb** (*Integer, Optional, Deprecated: This parameter is deprecated and will be unsupported in future API releases.*) – For metered bandwidth_type only. Acceptable values

sku	monthly_transfer_gb options

- **bandwidth_type** (*String, Optional, Deprecated: This parameter is deprecated and will be unsupported in future API releases.*) – Bandwidth type for your connection. Acceptable values

  'metered' Charged per Gb of data transferred.

  'unmetered' The server has part or all of a dedicated connection; no extra charges

- **backups** (*Integer, Optional*) –
- **lvs** (*Integer, Optional, Deprecated and ignored: This parameter is deprecated and will be unsupported in future API releases.*) – Load-balancing and auto-failover
- **intrusion_detection** (*String, Optional, Deprecated: This parameter is deprecated and will be unsupported in future API releases.*) – Intrusion Detection Acceptable values

  'none' No intrusion detection

  'basic' Intrusion Detection Self-monitored

- **monitoring_level** (*String, Optional, Deprecated: This parameter is deprecated and will be unsupported in future API releases.*) – Server Monitoring. Acceptable values

  'basic' Server Monitoring Basic

  'advanced' Server Monitoring Advanced Self-monitored

- **vulnscan** (*String, Optional, Deprecated: This parameter is deprecated and will be unsupported in future API releases.*) – Vulnerability Management - Powered by F-Secure Radar Acceptable values

  'none' No scanning

  'basic' Self-monitored

- **database** (*String, Optional*) – Microsoft SQL server. Available only for Windows servers.

  Acceptable values:

- **antivirus** (*String, Optional*) – Antivirus software. Windows servers only. Acceptable values

  'none' None

  'sophos_antivirus' Sophos Anti-virus

- **firewall** (*String, Optional*) – The type of firewall you require. For more information see Memset's Firewalling page. 'managed' only available with `support_level=managed`. Acceptable values

  'none' No firewalling

  'basic' Basic

  'self_managed' Self-managed

  'managed' Memset-managed

- **firewall_rule_group** (*String, Optional*) – The name of an existing account firewall rule group to apply. (For the following firewall types only: 'managed', 'self_managed')
- **sku** (*String*) – The SKU of the service you wish to provision. Acceptable values are

sku	description

See also: `create.available()` method to get a list with descriptions of available services.

- **dry_run** (*Boolean*) – If True, then the service is not provisioned but the information is still returned.
- **discount_code** (*String, Optional*) – Discount code to be applied.
- **add_to_next_bill** (*Boolean, Optional*) – If True, any charges will be added to the next consolidated bill if possible. If False, an invoice will be generated using the account's default payment method. Defaults to True for accounts with consolidated billing enabled, otherwise False. It is only possible to set *add_to_next_bill=True* for accounts with consolidated billing enabled.
- **os** (*String*) – The Operating System. Acceptable values include a Memstore snapshot name or one of the operating systems below. Please note the snapshot name needs to be the `os_option` value obtained from the `server.snapshot_list()` method.

os	os_bits	description	min ram	max ram
centos6_64	64	CentOS 6 64-bit	n/a	n/a
win2012serverstd_r2_64	64	Windows Server 2012 Standard R2 64-bit	2GB	n/a
ubuntu_trusty_64	64	Ubuntu 14.04 LTS (Trusty Tahr) 64-bit	n/a	n/a
centos7_64	64	CentOS 7 64-bit	n/a	n/a
ubuntu_trusty_docker_64	64	Ubuntu 14.04 LTS (Trusty Tahr, Docker pre-installed) 64-bit	n/a	n/a
centos7_cpanel_64	64	CentOS 7 cPanel 64-bit	2GB	n/a
win2016serverstd_64	64	Windows Server 2016 Standard 64-bit	2GB	n/a
ubuntu_xenial_docker_64	64	Ubuntu 16.04 LTS (Xenial Xerus, Docker pre-installed) 64-bit	n/a	n/a
debian_stretch_64	64	Debian 9 (Stretch) 64-bit	n/a	n/a
ubuntu_bionic_64	64	Ubuntu 18.04 LTS (Bionic Beaver) 64-bit	n/a	n/a
ubuntu_bionic_docker_64	64	Ubuntu 18.04 LTS (Bionic Beaver, Docker pre-installed) 64-bit	n/a	n/a
win2019serverstd_64	64	Windows Server 2019 Standard	2GB	n/a
debian_buster_64	64	Debian 10 (Buster) 64-bit	n/a	n/a

See also: Operating systems for further information.

- **os_bits** (*String, Optional, Deprecated and ignored: This parameter is deprecated and will be unsupported in future API releases.*) – Whether to use a 32 or 64-bit system. Note that some operating systems may not be available in both versions. Deprecated for newly provisioned servers - operating systems of all newly provisioned servers are 64 bit. Acceptable values: '32', '64'.
- **support_level** (*String, Optional*) – The level of technical support. Default value is "standard" unless the sku is a Standard Cloud VPS sku, in which case the default and only available value is "infrastructure_only". See our support matrix for further information. Acceptable values

  'standard' Standard Support

  'premium' Premium Support

'premium' Premium Plus Support

'managed_infrastructure' Standard Support

'managed_platform' Premium Support

'infrastructure_only' For use with Standard VPS and deprecated classic Miniserver VMs.

'basic' Deprecated support level for use with our classic Miniserver VMs.

'managed' Deprecated Support level for use with our classic Miniserver VMs.

- **vlan** (*String, Optional*) – The name of a vLAN product to put the server in when it is created.
- **network_zone** (*String, Optional*) – The Network Zone to deploy your Miniserver VM in. The default network zone is 'dunsfold'. Acceptable values are

network_zone	location
reading	Maidenhead, Berkshire, UK
dunsfold	Dunsfold, Surrey, UK

- **data_zone** (*String, Optional*) – The Data Zone in which to deploy your Miniserver VM. For regular accounts, defaults to 'Memset Public Cloud'. For community cloud accounts, defaults to that account's community cloud.
- **period** (*Integer, Optional*) – The rental period in months. Valid rental period values are: 1, 3, 6, 12. The default period is 1.
- **pub_ssh_key** (*String, Optional*) – Public SSH key to be installed in the server (*root* user). Only available for Linux servers.
- **disk_type** (*String, Optional, Deprecated: This parameter is deprecated and will be unsupported in future API releases.*) – The disk type required. Acceptable values

  'hdd' Standard rotating magnetic media

  'ssd' High performance Solid State Disk

create.**memstore**()

Provisions a Memstore instance.

See the Memstore cloud storage page for pricing and more information.

Parameters

- **sku** (*String*) – The SKU of the service you wish to provision. Acceptable values are

sku	description

  See also: `create.available()` method to get a list with descriptions of available services.

- **dry_run** (*Boolean*) – If True, then the service is not provisioned but the information is still returned.
- **discount_code** (*String, Optional*) – Discount code to be applied.
- **add_to_next_bill** (*Boolean, Optional*) – If True, any charges will be added to the next consolidated bill if possible. If False, an invoice will be generated using the account's default payment method. Defaults to True for accounts with consolidated billing enabled, otherwise False. It is only possible to set *add_to_next_bill=True* for accounts with consolidated billing enabled.

- **network_zone** (*String, Optional*) – The Network Zone to deploy your Memstore. The default network zone is 'reading'. Acceptable values are

network_zone	location
reading	Maidenhead, Berkshire, UK
dunsfold	Dunsfold, Surrey, UK

- **data_zone** (*String, Optional*) – The Data Zone to deploy your Memstore. The default data zone depends on whether the account belongs to a community cloud or not. For regular accounts, defaults to 'Memset Public Cloud'

`create.`**`monthly_fullserver`**`()`

Provisions a full server for a period of months.

See the Fully Dedicated Server packages page for pricing and more information.

Note that the following optional extras will affect the monthly price: *os*, *firewall*, *disk*, *ram*, *net_card_speed_gbps*, *bandwidth_type*, *connection*, *monthly_transfer_gb*, *period support_level*, *monitoring_level*, *backups*, *vulnscan*, *database* and *antivirus*.

Parameters

- **bandwidth_type** (*String, Optional, Deprecated: This parameter is deprecated and will be unsupported in future API releases.*) – Bandwidth type for your connection. Acceptable values

  'metered' Charged per Gb of data transferred.

  'unmetered' The server has part or all of a dedicated connection; no extra charges

- **backups** (*Integer, Optional*) – Managed daily backups. Capacity required in Gb. Not available for Standard VPS (i.e. skus beginning with VPS0)

  Acceptable values for Linux operating systems:

  Acceptable values for Windows operating systems:

- **lvs** (*Integer, Optional, Deprecated and ignored: This parameter is deprecated and will be unsupported in future API releases.*) – Load-balancing and auto-failover

- **intrusion_detection** (*String, Optional*) – Intrusion Detection. "monitored" and "protected" only available with `support_level=managed`. Acceptable values

  'none' No intrusion detection

  'basic' Intrusion Detection Self-monitored

  'monitored' Intrusion Detection Memset-monitored

  'protected' Intrusion Detection Memset-protected

- **monitoring_level** (*String, Optional*) – Server Monitoring. "managed" only available with `support_level=managed`. Acceptable values

  'basic' Server Monitoring Basic

  'advanced' Server Monitoring Advanced Self-monitored

  'managed' Server Monitoring Advanced Memset-monitored

- **vulnscan** (*String, Optional*) – Vulnerability Management - Powered by F-Secure Radar. If you have not already done so for your account, you will be emailed with instructions on how to activate this service. "managed" only available with `support_level=managed`. Acceptable values

  'none' No vulnerability management

'basic' Self-monitored

'managed' Memset-monitored

- **database** (*String, Optional*) – Microsoft SQL server. Available only for Windows servers.

  Acceptable values:

- **antivirus** (*String, Optional*) – Antivirus software. Windows servers only. Acceptable values

  'none' None

  'sophos_antivirus' Sophos Anti-virus

- **firewall** (*String, Optional*) – The type of firewall you require. For more information see Memset's Firewalling page. 'managed' only available with `support_level=managed`. Acceptable values

  'none' No firewalling

  'basic' Basic

  'self_managed' Self-managed

  'managed' Memset-managed

- **firewall_rule_group** (*String, Optional*) – The name of an existing account firewall rule group to apply. (For the following firewall types only: 'managed', 'self_managed')

- **sku** (*String*) – The SKU of the service you wish to provision. Acceptable values are

sku	description

  See also: `create.available()` method to get a list with descriptions of available services.

- **dry_run** (*Boolean*) – If True, then the service is not provisioned but the information is still returned.

- **discount_code** (*String, Optional*) – Discount code to be applied.

- **add_to_next_bill** (*Boolean, Optional*) – If True, any charges will be added to the next consolidated bill if possible. If False, an invoice will be generated using the account's default payment method. Defaults to True for accounts with consolidated billing enabled, otherwise False. It is only possible to set *add_to_next_bill=True* for accounts with consolidated billing enabled.

- **os** (*String*) – The Operating System. Acceptable values include a Memstore snapshot name or one of the operating systems below. Please note the snapshot name needs to be the `os_option` value obtained from the `server.snapshot_list()` method.

os	os_bits	description	min ram	max ram
centos6_64	64	CentOS 6 64-bit	n/a	n/a
win2012serverstd_r2_64	64	Windows Server 2012 Standard R2 64-bit	2GB	n/a
ubuntu_trusty_64	64	Ubuntu 14.04 LTS (Trusty Tahr) 64-bit	n/a	n/a
centos7_64	64	CentOS 7 64-bit	n/a	n/a
win2012serverdc_r2_64	64	Windows Server 2012 Datacenter R2 64-bit	2GB	n/a
centos7_cpanel_64	64	CentOS 7 cPanel 64-bit	2GB	n/a
win2016serverstd_64	64	Windows Server 2016 Standard 64-bit	2GB	n/a
win2016serverdc_64	64	Windows Server 2016 Datacenter 64-bit	2GB	n/a
debian_stretch_64	64	Debian 9 (Stretch) 64-bit	n/a	n/a
ubuntu_bionic_64	64	Ubuntu 18.04 LTS (Bionic Beaver) 64-bit	n/a	n/a
win2019serverstd_64	64	Windows Server 2019 Standard	2GB	n/a
debian_buster_64	64	Debian 10 (Buster) 64-bit	n/a	n/a

See also: Operating systems for further information.

- **os_bits** (*String, Optional, Deprecated and ignored: This parameter is deprecated and will be unsupported in future API releases.*) – Whether to use a 32 or 64-bit system. Note that some operating systems may not be available in both versions. Deprecated for newly provisioned servers - operating systems of all newly provisioned servers are 64 bit. Acceptable values: '32', '64'.
- **support_level** (*String, Optional*) – The level of technical support. Default value is "standard" unless the sku is a Standard Cloud VPS sku, in which case the default and only available value is "infrastructure_only". See our support matrix for further information. Acceptable values

  'standard' Standard Support

  'premium' Premium Support

  'premium' Premium Plus Support

  'managed_infrastructure' Standard Support

  'managed_platform' Premium Support

  'infrastructure_only' For use with Standard VPS and deprecated classic Miniserver VMs.

  'basic' Deprecated support level for use with our classic Miniserver VMs.

  'managed' Deprecated Support level for use with our classic Miniserver VMs.
- **vlan** (*String, Optional*) – The name of a vLAN product to put the server in when it is created.
- **network_zone** (*String, Optional*) – The Network Zone to deploy your server. The default network zone is 'dunsfold'. Acceptable values are

network_zone	location
reading	Maidenhead, Berkshire, UK
dunsfold	Dunsfold, Surrey, UK

- **data_zone** (*String, Optional*) – The Data Zone to deploy your server. The default data zone depends on whether the account belongs to a community cloud or not. For regular accounts,

defaults to 'Memset Public Cloud'

- **period** (*Integer, Optional*) – The rental period in months. Valid rental period values are: 1, 3, 6, 12. The default period is 1.
- **pub_ssh_key** (*String, Optional*) – Public SSH key to be installed in the server (*root* user). Only available for Linux servers.
- **disk** (*String, Optional*) – RAID disk size.

  Valid options

sku	disk options

- **ram** (*String, Optional*) – The amount of RAM for your server.

sku	ram options

- **nvme** (*String, Optional*) – NVMe SSDs (PCIE based flash storage). Only available on some product specs

  Valid options

sku	nvme options

- **net_card_speed_gbps** (*String, Optional*) – The network card speed for the server (Gbps). Acceptable values are 1 or 10. 10Gbps networking is only available for the following skus:
- **partitioning** (*String, Optional*) – Disk partitioning scheme. Acceptable values depend on the `sku`, `os` and `disk` options. For 4-disk chassis types (SKU *UFS20* and *UFS21*), the value *all_root* is only valid for Linux operating systems. For other chassis types, the value *all_root* is only valid when the disk array is less than or equal to 2 TB. Acceptable values:

  all_root For Linux operating systems: Single large root partition. For Windows operating systems: Single large drive C:

  data_home For Linux operating systems, 20 GB root partition and the remainder in /home. For Windows operating systems: 50 GB drive C: and the remainder in drive D:

  two_volumes Two volumes, C: and D:. Only applicable to 4-disk dedicated servers with Windows operating systems.

  custom Custom partitioning. If you select this option, you must include your requirements using the `setup_requirements` parameter.
- **setup_instructions** (*String, Optional*) – Special setup instructions, for example custom partitioning requirements.

`create.`**`monthly_miniserver`**`()`

Provisions a Miniserver VM instance for a period of months.

See the Miniserver VM packages page for pricing and more information.

> **Warning:** Please note that the following parameters are deprecated and only supported for our classic Miniservers (e.g. skus beginning with MS0):
> *disk_type*, *disk_gb*, *bandwidth_type*, *connection*, *monthly_transfer_gb*

The following optional extras will affect the monthly price: *os*, *firewall*, *disk_type*, *disk_gb*, *bandwidth_type*, *connection*, *monthly_transfer_gb*, *period*, *support_level*, *monitoring_level*, *backups*, *vulnscan*, *database*, *antivirus* and *intrusion_detection*

Parameters

- **connection** (*String, Optional, Deprecated: This parameter is deprecated and will be unsupported in future API releases.*) – For *unmetered* `bandwidth_type` only. Acceptable values are a combination of burst speed and contention ratio e.g. 5mbps_10to1 means 5Mbps burst with contention of 10:1. Dedicated line speed therefore is burst divided by contention, e.g. the dedicated line speed for a line with a connection value of 5mbps_10to1 is 5 / 10 = 0.5Mbps.

sku	connection options

- **monthly_transfer_gb** (*Integer, Optional, Deprecated: This parameter is deprecated and will be unsupported in future API releases.*) – For metered bandwidth_type only. Acceptable values

sku	monthly_transfer_gb options

- **bandwidth_type** (*String, Optional, Deprecated: This parameter is deprecated and will be unsupported in future API releases.*) – Bandwidth type for your connection. Acceptable values

  'metered' Charged per Gb of data transferred.

  'unmetered' The server has part or all of a dedicated connection; no extra charges

- **backups** (*Integer, Optional*) – Managed daily backups. Capacity required in Gb. Not available for Standard VPS (i.e. skus beginning with VPS0)

  Acceptable values for Linux operating systems:

  Acceptable values for Windows operating systems:

- **lvs** (*Integer, Optional, Deprecated and ignored: This parameter is deprecated and will be unsupported in future API releases.*) – Load-balancing and auto-failover

- **intrusion_detection** (*String, Optional*) – Intrusion Detection. "monitored" and "protected" only available with `support_level=managed`. Acceptable values

  'none' No intrusion detection

  'basic' Intrusion Detection Self-monitored

  'monitored' Intrusion Detection Memset-monitored

  'protected' Intrusion Detection Memset-protected

- **monitoring_level** (*String, Optional*) – Server Monitoring. "managed" only available with `support_level=managed`. Acceptable values

  'basic' Server Monitoring Basic

  'advanced' Server Monitoring Advanced Self-monitored

  'managed' Server Monitoring Advanced Memset-monitored

- **vulnscan** (*String, Optional*) – Vulnerability Management - Powered by F-Secure Radar. If you have not already done so for your account, you will be emailed with instructions on how to activate this service. "managed" only available with `support_level=managed`. Acceptable values

  'none' No vulnerability management

  'basic' Self-monitored

'managed' Memset-monitored

- **database** (*String, Optional*) – Microsoft SQL server. Available only for Windows servers.

  Acceptable values:

- **antivirus** (*String, Optional*) – Antivirus software. Windows servers only. Acceptable values

  'none' None

  'sophos_antivirus' Sophos Anti-virus

- **firewall** (*String, Optional*) – The type of firewall you require. For more information see Memset's Firewalling page. 'managed' only available with `support_level=managed`. Acceptable values

  'none' No firewalling

  'basic' Basic

  'self_managed' Self-managed

  'managed' Memset-managed

- **firewall_rule_group** (*String, Optional*) – The name of an existing account firewall rule group to apply. (For the following firewall types only: 'managed', 'self_managed')

- **sku** (*String*) – **Premium Cloud VPS skus**

sku	description

Standard Cloud VPS skus

sku	description

Classic Miniserver skus

sku	description

Please note that our classic Miniserver skus are deprecated and will be unsupported in future API releases. See also: `create.available()` method to get a list with descriptions of available services.

- **dry_run** (*Boolean*) – If True, then the service is not provisioned but the information is still returned.

- **discount_code** (*String, Optional*) – Discount code to be applied.

- **add_to_next_bill** (*Boolean, Optional*) – If True, any charges will be added to the next consolidated bill if possible. If False, an invoice will be generated using the account's default payment method. Defaults to True for accounts with consolidated billing enabled, otherwise False. It is only possible to set *add_to_next_bill=True* for accounts with consolidated billing enabled.

- **os** (*String*) – The Operating System. Acceptable values include a Memstore snapshot name or one of the operating systems below. Please note the snapshot name needs to be the `os_option` value obtained from the `server.snapshot_list()` method.

os	os_bits	description	min ram	max ram
centos6_64	64	CentOS 6 64-bit	n/a	n/a
win2012serverstd_r2_64	64	Windows Server 2012 Standard R2 64-bit	2GB	n/a
ubuntu_trusty_64	64	Ubuntu 14.04 LTS (Trusty Tahr) 64-bit	n/a	n/a
centos7_64	64	CentOS 7 64-bit	n/a	n/a
ubuntu_trusty_docker_64	64	Ubuntu 14.04 LTS (Trusty Tahr, Docker pre-installed) 64-bit	n/a	n/a
centos7_cpanel_64	64	CentOS 7 cPanel 64-bit	2GB	n/a
win2016serverstd_64	64	Windows Server 2016 Standard 64-bit	2GB	n/a
ubuntu_xenial_docker_64	64	Ubuntu 16.04 LTS (Xenial Xerus, Docker pre-installed) 64-bit	n/a	n/a
debian_stretch_64	64	Debian 9 (Stretch) 64-bit	n/a	n/a
ubuntu_bionic_64	64	Ubuntu 18.04 LTS (Bionic Beaver) 64-bit	n/a	n/a
ubuntu_bionic_docker_64	64	Ubuntu 18.04 LTS (Bionic Beaver, Docker pre-installed) 64-bit	n/a	n/a
win2019serverstd_64	64	Windows Server 2019 Standard	2GB	n/a
debian_buster_64	64	Debian 10 (Buster) 64-bit	n/a	n/a

See also: Operating systems for further information.

- **os_bits** (*String, Optional, Deprecated and ignored: This parameter is deprecated and will be unsupported in future API releases.*) – Whether to use a 32 or 64-bit system. Note that some operating systems may not be available in both versions. Deprecated for newly provisioned servers - operating systems of all newly provisioned servers are 64 bit. Acceptable values: '32', '64'.
- **support_level** (*String, Optional*) – The level of technical support. Default value is "standard" unless the sku is a Standard Cloud VPS sku, in which case the default and only available value is "infrastructure_only". See our support matrix for further information. Acceptable values

  'standard' Standard Support

  'premium' Premium Support

  'premium' Premium Plus Support

  'managed_infrastructure' Standard Support

  'managed_platform' Premium Support

  'infrastructure_only' For use with Standard VPS and deprecated classic Miniserver VMs.

  'basic' Deprecated support level for use with our classic Miniserver VMs.

  'managed' Deprecated Support level for use with our classic Miniserver VMs.
- **vlan** (*String, Optional*) – The name of a vLAN product to put the server in when it is created.
- **network_zone** (*String, Optional*) – The Network Zone to deploy your Miniserver VM. The default network zone is 'dunsfold'. Acceptable values are

network_zone	location
reading	Maidenhead, Berkshire, UK
dunsfold	Dunsfold, Surrey, UK

- **data_zone** (*String, Optional*) – The Data Zone to deploy your Miniserver VM. The default data zone depends on whether the account belongs to a community cloud or not. For regular accounts, defaults to 'Memset Public Cloud'
- **period** (*Integer, Optional*) – The rental period in months. Valid rental period values are: 1, 3, 6, 12. The default period is 1.
- **pub_ssh_key** (*String, Optional*) – Public SSH key to be installed in the server (*root* user). Only available for Linux servers.
- **disk_type** (*String, Optional, Deprecated: This parameter is deprecated and will be unsupported in future API releases.*) – For classic Miniservers only. All current Miniserver specifications have the 'ssd' disk_type. Acceptable values

  'hdd' Standard rotating magnetic media

  'ssd' High performance Solid State Disk

create.**openstack_project**()

Provisions a Cloud IaaS (powered by OpenStack) project.

See the Cloud IaaS page for pricing and more information.

Parameters

- **sku** (*String*) – The SKU of the service you wish to provision. Acceptable values are

sku	description

  See also: `create.available()` method to get a list with descriptions of available services.
- **dry_run** (*Boolean*) – If True, then the service is not provisioned but the information is still returned.
- **discount_code** (*String, Optional*) – Discount code to be applied.
- **add_to_next_bill** (*Boolean, Optional*) – If True, any charges will be added to the next consolidated bill if possible. If False, an invoice will be generated using the account's default payment method. Defaults to True for accounts with consolidated billing enabled, otherwise False. It is only possible to set *add_to_next_bill=True* for accounts with consolidated billing enabled.

create.**verify_discount_code**()

Returns information about a discount code.

Parameters **discount_code** (*String*) – Discount code to verify.

Returns

if the discount code is valid, this method returns a dictionary with following keys:

sku String: The code used to identify the service to apply the discount code.

description String: Description of the discount code.

incompatible List: parameter values that aren't compatible with the discount (format "param:value"). It can be empty.

required List: parameter values that are implicitly set when using this discount (format "param:value"). Any other value in parameters in this list will be ignored. It can be empty.

In order to see how a discount code affects a product price, use the appropriate create method providing the discount code and setting dry_run parameter to True.

Raises `ApiErrorDoesNotExist` if the discount code doesn't exist or is not valid for this account.

1.2.9 DNS Methods

API for DNS

This provides an API for the Memset DNS (Domain Name Service).

The Memset DNS API has three main objects

- A zone domain
- A zone record
- A zone

A zone is composed of a number of zone domains and a number of zone records. The zone domains specify which domains the nameservers will return this zone for, and the zone records specify which records are returned.

Zone records are usually relative, which means that they have the zone domain appended to them. If you make a zone record with *name* or *www* and a domain of *example.com* then the record will be served for queries matching *www.example.com*. If you were to add *example.co.uk* to the zone domains for this zone then *www.example.co.uk* would be served with the same value.

Zone records are relative if the relative flag is *True*. If the relative flag is *False* and the name contains dots, then they are treated as absolute, otherwise they are treated as relative. For example if you make a zone record with name *www.example.com* it will be served under any domain as *www.example.com*, however if you set the relative flag to *True* it will be served under the domain specified. The relative flag is essential if you want to host zones under multiple domains where part of the name contains dots, e.g. if name was *a.www* and domain was *example.com* then you would need to set the relative flag in the zone record if you wanted it served as *a.www.example.com* rather than *a.www*.

`dns.`**`reload`**`()`

Reloads the name servers with recent changes.

Running this will ensure that the current state of your DNS records is synced with your DNS server. DNS records are pushed out the the server every 15 minutes normally, but running this will speed up the process.

You may poll the job returned to discover when this has been done. We don't guarantee any particular timescale for this, but normally it will only take a few seconds.

Returns A dictionary as described in `job.status()`.

`dns.`**`reverse_map_list`**`()`

Query the reverse maps for all servers, or a single server if name is supplied, or a single address if address is supplied. If name and address are supplied then it will only show address if it is attached to that server.

Parameters

- **name** (*String, Optional*) – Name of the server. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **address** (*String, Optional*) – The IPv4 or IPv6 address to list. Ensure this value has at most 250 characters.

Returns

a list of dictionary with the following keys:

name String: Which server name this is attached to

address String: IPv4 or IPv6 address

reverse_map String: what this IP address reverse maps to

dns.**reverse_map_update**()
Updates a reverse map. If name is supplied then it checks that address is attached to that server before updating it.

Parameters

- **name** (*String, Optional*) – Name of the server. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **address** (*String*) – The IPv4 or IPv6 address to change the reverse map for. Ensure this value has at most 250 characters.
- **reverse_map** (*String*) – The name this IP address reverse maps to. Must be a valid hostname. Ensure this value has at most 250 characters.

Returns a list of dictionaries as described in dns.reverse_map_list().

dns.**zone_create**()
Create a zone, optionally providing records and domains.

Parameters

- **nickname** (*String*) – A customer specified name for the zone file. Not guaranteed unique. Ensure this value has at most 250 characters.
- **ttl** (*Integer, Optional*) – Time to live - 0 for use this zone's default TTL. Acceptable values

  0 Default

  300 5 minutes

  600 10 minutes

  900 15 minutes

  1800 30 minutes

  3600 1 hour

  7200 2 hours

  10800 3 hours

  21600 6 hours

  43200 12 hours

  86400 24 hours
- **records** (*List, Optional*) – A List of Dictionaries describing zone records to create as described in dns.zone_record_create(). Note that the dictionaries shouldn't include the zone_id parameter as this will be automatically inserted.
- **domains** (*List, Optional*) – A List of Dictionaries describing zone domains to create as described in dns.zone_domain_create(). Note that the dictionaries shouldn't include the zone_id parameter as this will be automatically inserted.

Returns a dictionary with information about the created zone as decribed in dns.zone_info().

dns.**zone_delete**()
Deletes a zone

NB this will delete any zone records and zone domains that are part of this zone.

Parameters **id** (*String*) – The Zone ID. Zone IDs are 32 hex digits.

Returns a dictionary with zone in before it was deleted as returned by dns.zone_info()

dns.**zone_domain_create**()
Create a zone domain.

Parameters

- **domain** (*String*) – The zone domain name. Ensure this value has at most 250 characters.
- **zone_id** (*String, Optional*) – The Zone ID. Zone IDs are 32 hex digits.

Returns a dictionary containing the contents of the zone domain after addition as described in dns.zone_domain_info().

dns.**zone_domain_delete**()
Deletes the zone domain

Parameters **domain** (*String*) – The zone domain name. Ensure this value has at most 250 characters.

Returns a dictionary containing the contents of the zone domain before it was deleted as described in dns.zone_domain_info().

dns.**zone_domain_info**()
Info about a zone domain

Parameters **domain** (*String*) – The zone domain name. Ensure this value has at most 250 characters.

Returns

a dictionary with the following keys

domain String: The domain name in question, e.g. 'example.com'

zone_id String, Optional: zone id (32 hex digits) that this domain is part of. If it isn't present then this domain is not part of a zone, or part of a zone you can't change.

dns.**zone_domain_list**()
Returns all the zone domains for this account

Returns list of dictionaries as described in dns.zone_domain_info().

dns.**zone_domain_update**()
Updates the zone domain.

Parameters

- **domain** (*String*) – The zone domain name. Ensure this value has at most 250 characters.
- **zone_id** (*String, Optional*) – The Zone ID. Zone IDs are 32 hex digits.

Returns a dictionary containing the contents of the zone domain after addition as described in dns.zone_domain_info().

dns.**zone_info**()
Info about a zone

Parameters **id** (*String*) – The Zone ID. Zone IDs are 32 hex digits.

Returns

a dictionary with the following keys

id String: zone id (32 hex digits).

nickname String: your name for this zone

ttl Integer: the default TTL for records in this zone in seconds, 0 means use Memset default of 1800 seconds.

records List: a list of dictionaries describing zone records that are used in this zone `dns.zone_record_info()`.

domains List: a list of dictionaries describing zone domains that are used for this zone see `dns.zone_domain_info()`.

`dns.`**`zone_list`**`()`
Lists all the zones in this account

Returns a list of dictionaries as described in `zone_info()`

`dns.`**`zone_record_create`**`()`
Create a zone record in the zone supplied

Parameters

- **zone_id** (*String*) – The Zone ID. Zone IDs are 32 hex digits.
- **type** (*String*) – Domain record type, e.g. 'A'. Acceptable values: 'A', 'AAAA', 'CNAME', 'MX', 'NS', 'SRV', 'TXT'.
- **record** (*String, Optional*) – Name of record, e.g. 'www'. Use the empty string '' to create a record for the domain itself. Must be a valid zone record name. Ensure this value has at most 250 characters.
- **address** (*String*) – Address of record, e.g. '1.2.3.4'. Only printable ASCII characters are allowed. Ensure this value has at most 250 characters.
- **ttl** (*Integer, Optional*) – Time to live - 0 for use this zone's default TTL. Acceptable values

  0 Default

  300 5 minutes

  600 10 minutes

  900 15 minutes

  1800 30 minutes

  3600 1 hour

  7200 2 hours

  10800 3 hours

  21600 6 hours

  43200 12 hours

  86400 24 hours
- **priority** (*Integer, Optional*) – Priority for MX and SRV records, eg 10. Ensure this value is greater than or equal to 0. Ensure this value is less than or equal to 999.
- **relative** (*Boolean, Optional*) – If set then we add the current domain onto the address field for CNAME, MX, NS and SRV record types.

Returns a dictionary with the zone record in as described in `dns.zone_record_info()`

`dns.`**`zone_record_delete`**`()`
Delete a zone record

Parameters **id** (*String*) – The Zone Record ID. Zone Record IDs are 32 hex digits.

Returns the zone record information before it was deleted as a dictionary as described in `dns.zone_record_info()`.

dns.**zone_record_info**()
Info about a zone record

Parameters **id** (*String*) – The Zone Record ID. Zone Record IDs are 32 hex digits.

Returns

a dictionary with the following keys

id String: id (32 hex digits) of this zone record

zone_id String: zone id (32 hex digits) that this zone record is part of.

type String: Domain record type: 'A', 'AAAA', 'CNAME', 'MX', 'NS', 'SRV', 'TXT'

record String: Name of record, e.g. 'www'

address String: Address of record, e.g. '1.2.3.4'

ttl Integer: Time to live - 0 for use this zone's default TTL

priority Integer: Priority for MX and SRV records, e.g. 10, not used for other record types

relative Boolean: If true we never assume the *name* is absolute. Normally if the *name* does not contain any dots we assume it is relative to the current domain. The domain is added for CNAME, MX, NS and SRV record types which are deemed to be relative.

dns.**zone_record_list**()
Returns all the zone records for this account

Returns list of dictionaries as described in dns.zone_record_info().

dns.**zone_record_update**()
Update a zone record. Supply any parameters you want updated.

Parameters

- **id** (*String*) – The Zone Record ID. Zone Record IDs are 32 hex digits.
- **zone_id** (*String, Optional*) – The Zone ID. Zone IDs are 32 hex digits.
- **type** (*String, Optional*) – Domain record type, e.g. 'A'. Acceptable values: 'A', 'AAAA', 'CNAME', 'MX', 'NS', 'SRV', 'TXT'.
- **record** (*String, Optional*) – Name of record, e.g. 'www'. Use the empty string '' to create a record for the domain itself. Must be a valid zone record name. Ensure this value has at most 250 characters.
- **address** (*String, Optional*) – Address of record, eg '1.2.3.4'. Only printable ASCII characters are allowed. Ensure this value has at most 250 characters.
- **ttl** (*Integer, Optional*) – Time to live - 0 for use this zone's default TTL. Acceptable values

  0 Default

  300 5 minutes

  600 10 minutes

  900 15 minutes

  1800 30 minutes

  3600 1 hour

  7200 2 hours

  10800 3 hours

21600 6 hours

43200 12 hours

86400 24 hours

- **priority** (*Integer, Optional*) – Priority for MX and SRV records, e.g. 10. Ensure this value is greater than or equal to 0. Ensure this value is less than or equal to 999.
- **relative** (*Boolean, Optional*) – If set then we add the current domain onto the address field for CNAME, MX, NS and SRV record types.

Returns a dictionary with the zone record in as described in `dns.zone_record_info()`

`dns.`**`zone_update`**`()`

Modifies a zone

Any parameters passed in will be used to update the zone.

If you want to create, update or delete zone records and zone domains then use the zone_record and zone_domain methods with the `zone_id` parameter.

Parameters

- **id** (*String*) – The Zone ID. Zone IDs are 32 hex digits.
- **nickname** (*String, Optional*) – A customer specified name for the zone file. Not guaranteed unique. Ensure this value has at most 250 characters.
- **ttl** (*Integer, Optional*) – Time to live - 0 for use this zone's default TTL. Acceptable values

0 Default

300 5 minutes

600 10 minutes

900 15 minutes

1800 30 minutes

3600 1 hour

7200 2 hours

10800 3 hours

21600 6 hours

43200 12 hours

86400 24 hours

Returns a dictionary with zone in after modification as returned by `dns.zone_info()`.

1.2.10 Firewalling Methods

API for Firewalling services.

This API can be used to retrieve and change firewalling configuration for your servers. Read only methods are provided for all servers. Configuration may only be changed for servers with Self-managed or Memset-managed firewalling.

Firewall rules are grouped together in firewall rule groups. When creating firewall rule groups, the rule group name will be created by the system.

Information about the firewall rule group currently applied to a server can be found in the `firewall_rule_group` entry of the dictionary returned by `server.info()`.

An *example* is provided in Python.

`firewalling.`**`rule_create`**`()`

Create a firewall rule within a rule group. Rules cannot be created in public rule groups.

The rule will be validated to ensure it is not malformed.

Parameters

- **rule_group_name** (*String*) – The name of the firewall rule group to which this rule is to be added.
- **ip_version** (*String, Optional*) – The Internet Protocol version. Defaults to "any". Acceptable values

  'any' any

  'ipv4' IPv4

  'ipv6' IPv6
- **action** (*String*) – The action for this rule. Acceptable values: 'ACCEPT', 'DROP', 'REJECT'.
- **source_ips** (*String, Optional*) – 'any' or a comma separated list of source IPv4 addresses without spaces. These can be CIDR notation, eg 1.2.3.4/24.
- **source_ports** (*String, Optional*) – 'any' or a comma separated list of port numbers without spaces
- **dest_ips** (*String, Optional*) – 'any' or a comma separated list of source IPv4 addresses without spaces.
- **source_ip6s** (*String, Optional*) – 'any' or a comma separated list of source IPv6 addresses without spaces. These can be CIDR notation, eg 2001:db8::/120.
- **dest_ip6s** (*String, Optional*) – 'any' or a comma separated list of source IPv6 addresses without spaces.
- **dest_ports** (*String*) – 'any' or a comma separated list of port numbers without spaces
- **protocols** (*String, Optional*) – The protocol(s). If 'any', the protocol and dst ports are not used for matching. Acceptable values

  'tcp' TCP

  'udp' UDP

  'icmp' ICMP

  'tcp,udp' TCP,UDP

  'gre' GRE

  'esp' ESP

  'ah' AH

  'ipip' IPIP

  'sctp' SCTP

  'any' any
- **ordering** (*Integer*) – The ordering for this rule. Ensure this value is greater than or equal to 1. Ensure this value is less than or equal to 30.
- **comment** (*String, Optional*) – Optional comment about this rule.

- **direction** (*String, Optional*) – The direction of traffic that this rule should be applied to. Either 'Inbound' or 'Outbound'. Defaults to Inbound Acceptable values

  'Inbound' Inbound

  'Outbound' Outbound

Returns A dictionary of the newly created rule as described in `firewalling.rule_info()`.

Raises May raise:

- `ApiErrorPreconditionFailed` if the rule group has the maximum number of rules already.
- `ApiErrorDoesNotExist` if the firewall rule group with provided name or nickname does not exist.
- `ApiErrorBadParameters` if the firewall rule parameters are invalid or malformed.

`firewalling.`**`rule_delete`**`()`

Delete the firewall rule with id `rule_id`.

Parameters **rule_id** (*String*) – The unique id of the rule

Returns The id of the successfully deleted rule.

Raises May raise:

- `ApiErrorDoesNotExist` if the firewall rule with id `rule_id` does not exist for this account. Rules within public firewall groups may not be deleted and therefore won't be searched.

`firewalling.`**`rule_group_create`**`()`

Add a new firewall rule group for this account.

Parameters

- **nickname** (*String*) – A nickname for this firewall rule group. Ensure this value has at most 225 characters.
- **notes** (*String, Optional*) – Notes about this firewall rule group.
- **rules** (*List, Optional*) – A List of dictionaries describing firewall rules to create as described in `firewalling.rule_info()` excluding the `rule_group_name` parameter (which is created automatically). Maximum number of rules per rule group: 30
- **default_outbound_policy** (*String, Optional*) – The default policy to be applied to outbound traffic. Supports 'Accept', 'Drop' or 'Reject'. Defaults to 'Accept'. This option is only available for customers using the outbound firewalling beta Acceptable values

  'RETURN' Accept

  'REJECT' Reject

  'DROP' Drop

Returns A dictionary as detailed in `firewalling.rule_group_info()` for the newly created firewall rule group.

Raises May raise:

- `ApiErrorPreconditionFailed` if the nickname is already in use for another firewall rule group on the account.
- `ApiErrorBadParameters` if more than the maximum number of rules for a group provided.

- ApiErrorBadParameters if any of the provided firewall rules are invalid or malformed.

firewalling.**rule_group_delete**()
Delete a firewall rule group for this account.

Parameters **rule_group_name** (*String*) – The name of the firewall rule group.

Returns the name of the deleted rule group.

Raises May raise:

- ApiErrorDoesNotExist if the rule group is not found. Public rule groups may not be deleted and therefore will not be searched.
- ApiErrorPreconditionFailed if there are servers using the rule group.

firewalling.**rule_group_info**()
Get information about a rule group.

Parameters **rule_group_name** (*String*) – The name of the firewall rule group.

Returns

A dictionary with the following keys

name String: The unique identifier for this rule group.

nickname String: The nickname of this rule group.

public Boolean: Whether this rule group is public.

notes String: Any notes associated with this rule group.

default_outbound_policy String: The default policy applied to outbound traffic.

rules Dictionary: As provided by firewalling.rule_info().

Raises ApiErrorDoesNotExist if the rule group does not exist.

firewalling.**rule_group_list**()
Retrieve a list of firewall rule groups for this account.

Parameters **include_public** (*Boolean, Optional*) – Include public rule groups. Default: True.

Returns A list of dictionaries as described in firewalling.rule_group_info().

firewalling.**rule_group_status**()
Check the status of a rule group for a specific server.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **rule_group_name** (*String*) – The name of the firewall rule group.

Returns

String: The status of the rule group for a server. The status may be one of the following:

active rule group is active and up to date for this server

pending rule group is active but there are pending changes which are not yet loaded in the firewall

na rule group is not currently applicable to this server

Raises ApiErrorDoesNotExist if the *name* or *rule_group_name* does not exist.

firewalling.**rule_info**()

Retrieve firewall rule information.

Parameters **rule_id** (*String*) – The unique id of the rule

Returns

A dictionary with the following keys:

rule_id String: The unique id of the rule.

rule_group_name String: The name of the firewall rule group this rule belongs to.

ip_version String: 'ipv4', 'ipv6' or 'all'.

action String: 'ACCEPT', 'DROP' or 'REJECT'.

source_ips String: 'any' or a comma separated list of source IP addresses without spaces. These can be CIDR notation, eg 1.2.3.4/24.

source_ip6s String: 'any' or a comma separated list of source IPv6 addresses without spaces. These can be CIDR notation, eg 2001:db8::/120.

dest_ips String: 'any' or a comma separated list of destination IP addresses without spaces.

dest_ip6s String: 'any' or a comma separated list of destination IPv6 addresses without spaces.

source_ports String: 'any' or a comma separated list of port numbers without spaces.

dest_ports String: 'any' or a comma separated list of port numbers without spaces.

protocols String: 'any' or a comma separated list of protocols without spaces.

ordering Integer: The position of this rule within the rule group.

comment String: Any comment associated with the rule.

direction String: 'Inbound' or 'Outbound'. This option is only supported for customers using the outbound firewalling beta.

Raises May raise:

- `ApiErrorBadParameters` if `rule_id` is malformed.
- `ApiErrorDoesNotExist` if the firewall rule with id `rule_id` is not found.

firewalling.**rule_update**()

Update a firewall rule. Rules which are part of public rule groups may not be changed and therefore will not be searched.

Cannot be used to change the direction of a rule.

Parameters

- **rule_id** (*String*) – The unique id of the rule
- **ip_version** (*String, Optional*) – The Internet Protocol version. Acceptable values

  'any' any

  'ipv4' IPv4

  'ipv6' IPv6
- **action** (*String, Optional*) – The action for this rule. Acceptable values: 'ACCEPT', 'DROP', 'REJECT'.
- **source_ips** (*String, Optional*) – 'any' or a comma separated list of source IPv4 addresses without spaces. These can be CIDR notation, eg 1.2.3.4/24.

- **dest_ips** (*String, Optional*) – 'any' or a comma separated list of source IPv4 addresses without spaces.
- **source_ip6s** (*String, Optional*) – 'any' or a comma separated list of source IPv6 addresses without spaces. These can be CIDR notation, eg 2001:db8::/120.
- **dest_ip6s** (*String, Optional*) – 'any' or a comma separated list of source IPv6 addresses without spaces.
- **source_ports** (*String, Optional*) – 'any' or a comma separated list of port numbers without spaces
- **dest_ports** (*String, Optional*) – 'any' or a comma separated list of port numbers without spaces
- **protocols** (*String, Optional*) – 'any' or a comma separated list of protocols without spaces. If 'any', the protocol and dst ports are not used for matching. Acceptable values

  'tcp' TCP

  'udp' UDP

  'icmp' ICMP

  'tcp,udp' TCP,UDP

  'gre' GRE

  'esp' ESP

  'ah' AH

  'ipip' IPIP

  'sctp' SCTP

  'any' any
- **ordering** (*Integer, Optional*) – The ordering for this rule. Ensure this value is greater than or equal to 1. Ensure this value is less than or equal to 30.
- **comment** (*String, Optional*) – Optional comment about this rule

Returns A dictionary as detailed in `firewalling.rule_info()` for the modified firewall rule.

Raises May raise:

- `ApiErrorDoesNotExist` if the firewall rule with id `rule_id` is not found.
- `ApiErrorBadParameters` if `rule_id` is malformed.
- `ApiErrorBadParameters` if the submitted rule parameters result in an invalid/malformed rule.

`firewalling.`**`update`**`()`

Apply a different firewall rule group to `name`.

The specified firewall rule group can be either a private rule group or one of Memset's public rule groups.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **rule_group_name** (*String*) – The name of the firewall rule group to apply to this server.

Returns A dictionary as provided by `firewalling.rule_group_info()` for the server.

Raises May raise:

- `ApiErrorDoesNotExist` if the server `name` does not exist or if the rule group with the given `rule_group_name` does not exist.
- `ApiErrorPreconditionFailed` if the server `name` does not have 'managed' firewalling type.

1.2.11 Internal Methods

API for Internal things

`internal.echo()`
Echo back parsed parameters. Useful for testing.

All parameters are optional.

Parameters

- **string** (*String, Optional*) – A String
- **list** (*List, Optional*) – A List
- **dictionary** (*Dictionary, Optional*) – A Dictionary
- **boolean** (*Boolean, Optional*) – A Boolean
- **float** (*Float, Optional*) – A Float
- **integer** (*Integer, Optional*) – An Integer
- **date** (*Date, Optional*) – A Date

Returns a dictionary of the parsed values with the same names as passed in. If a value wasn't passed in then a default value for that type will be returned.

`internal.noop()`
Does nothing. Still does all the authentication though.

Takes no parameters

Returns an empty dictionary

`internal.release()`
Returns the API release

Returns

A dictionary with the following keys

release String: release number, e.g. "0.9.19."

`internal.version()`
Returns the API version

Returns

A dictionary with the following keys

version String: version number, e.g. "0.9"

1.2.12 Invoice Methods

API for Invoice Methods.

`invoice.info()`
Gets details for the invoice with reference number *ref*

Parameters **ref** (*Integer*) – The invoice number

Returns

A dictionary with the following keys:

ref Integer: A unique reference to identify this invoice

amount Float: The amount for this invoice

currency String: The currency of this invoice

vat Float: The VAT for this invoice

status String: The status of the invoice

proforma Boolean: Whether this is a proforma

payment_method String: The payment method of this invoice

items List of three tuples (Item description, amount, period): The items on the invoice.

date Date: The date of the invoice

Raises `ApiErrorDoesNotExist` if the invoice does not exist.

`invoice.list()`
Retrieves details for invoices according to the search parameters.

Parameters

- **date_gte** (*Date, Optional*) – Invoice date on or after
- **date_lte** (*Date, Optional*) – Invoice date on or before
- **status** (*String, Optional*) – The status of the order Acceptable values: 'NEW', 'AWAITING-PAYMENT', 'RENEWAL', 'REFUND', 'CANCELLED', 'DONE', 'PROCESSING'.

Returns list of dictionaries as described in `invoice.info()`. Empty if no invoices found for the given parameters.

1.2.13 Job Methods

API interface for jobs

`job.status()`
Reads the status of the job passed in.

Parameters

- **name** (*String, Optional*) – Name of the service - only show jobs that are attached to this service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **id** (*String*) – The Job ID. Job IDs are 32 hex digits.

Returns

A dictionary with the following keys:

id String: job id (32 hex digits).

type String: textual description of what the job is doing, e.g. "SETUP-NEW-MINISERVER".

status String: some textual description of what the job is doing now.

service (optional) String: The name of the service this is job is related to, if available. Services don't have a name until they have been created.

finished Boolean: set if job has finished.

error Boolean: set if the job didn't finish properly or was cancelled.

1.2.14 Loadbalancer Server Methods

API methods for adding and removing servers to/from Load Balancers.

`loadbalancer.server.add()`

Add the given server to the service. If the server is a miniserver, this method may reboot it in order to reconfigure its network. For further information, please see the VLAN documentation.

Parameters

- **load_balancer** (*String*) – The name of the load balancer. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – The name of the load balancer service. Service names must only consist of letters, numbers, underscores and hyphens. Ensure this value has at most 64 characters.
- **server_name** (*String*) – Name of the server.
- **ip_address** (*String, Optional*) – The IP address that the load balancer should forward incoming connections to. Defaults to choosing one of the server's VLAN IP addresses.
- **port** (*Integer, Optional*) – The port number that the load balancer should forward incoming connections to. Defaults to the service's port. Ensure this value is greater than or equal to 1. Ensure this value is less than or equal to 65535.
- **enabled** (*Boolean, Optional*) – Whether this server is enabled. Defaults to false.
- **fallback** (*Boolean, Optional*) – Whether this server is a fallback server. Defaults to false.
- **weight** (*Integer, Optional*) – Weight used for some scheduling algorithms. Defaults to 10.

Returns A dictionary as described in `loadbalancer.server.info()`.

Raises Will raise:

- `ApiErrorDoesNotExist` if one of the cluster, cluster service or server with the provided name does not exist, or if no IP address is specified and a default IP address cannot be found.
- `ApiErrorBadParameters` if the given IP address does not belong to the given server.
- `ApiErrorPreconditionFailed` if one of the following preconditions is not satisfied.

Preconditions The following preconditions must be satisfied:

- The server must be in the same VLAN as the load balancer.
- The server must not be in the middle of a reboot operation.
- The server's SLA must not be lower than the load balancer's SLA.
- The server must not already be attached to this service.

loadbalancer.server.**info**()

Return information about the specified server, as attached to the given service.

Parameters

- **load_balancer** (*String*) – The name of the load balancer. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – The name of the load balancer service. Service names must only consist of letters, numbers, underscores and hyphens. Ensure this value has at most 64 characters.
- **server_name** (*String*) – Name of the server.

Returns

a dictionary with the following keys:

name: String: The name of the server.

ip_address: String: The IP address of the server. This is the address that the load balancer will forward incoming connections to

port: Integer: The port to which the load balancer will forward incoming connections.

enabled: Boolean: Whether this server is enabled. If the server is not enabled, it won't be used to handle requests.

fallback: Boolean: If true, this server is only used as a fallback. The server won't be used if there are any non-fallback servers available (enabled and responding to connections.

weight: Integer: Weight used for load balancing. Higher numbers mean more connections will come to this server.

Raises Will raise `ApiErrorDoesNotExist` if the load balancer, service or server does not exist. Will raise `ApiErrorPreconditionFailed` if the server is not attached to the given service.

loadbalancer.server.**remove**()

Remove a server from the given service. The server must not be in the middle of rebooting. If the server is a miniserver, this method may reboot it in order to reconfigure its network. For further information, please see the VLAN documentation. If you just want to disable a server temporarily, use `loadbalancer.server.update()` to set *enabled* to *false*.

Parameters

- **load_balancer** (*String*) – The name of the load balancer. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – The name of the load balancer service. Service names must only consist of letters, numbers, underscores and hyphens. Ensure this value has at most 64 characters.
- **server_name** (*String*) – Name of the server.

Returns The name of the server that was removed.

Raises Will raise `ApiErrorDoesNotExist` if the named load balancer, service or server does not exist. Will raise `ApiErrorPreconditionFailed` if the server is not attached to the specified service, or if the server is being rebooted.

loadbalancer.server.**update**()

Update the settings of a server as attached to the given service. Settings that are not provided will not be updated.

Parameters

- **load_balancer** (*String*) – The name of the load balancer. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – The name of the load balancer service. Service names must only consist of letters, numbers, underscores and hyphens. Ensure this value has at most 64 characters.
- **server_name** (*String*) – Name of the server.
- **ip_address** (*String, Optional*) – The IP address that the load balancer should forward incoming connections to.
- **port** (*Integer, Optional*) – The port number that the load balancer should forward incoming connections to. Ensure this value is greater than or equal to 1. Ensure this value is less than or equal to 65535.
- **enabled** (*Boolean, Optional*) – Whether this server is enabled.
- **fallback** (*Boolean, Optional*) – Whether this server is a fallback server.
- **weight** (*Integer, Optional*) – Weight used for some scheduling algorithms.

Returns A dictionary as described in `loadbalancer.server.info()`.

Raises Will raise:

- `ApiErrorDoesNotExist` if the named load balancer, service or server does not exist.
- `ApiErrorPreconditionFailed` if the named server is not attached to this service.
- `ApiErrorBadParameters` if the given IP address does not belong to the given server.

1.2.15 Loadbalancer Service Methods

API methods for managing Load Balancer services.

`loadbalancer.service.add()`

Define a new service on the given load balancer. The service must be given a name. This is used to refer to the service later.

Parameters

- **load_balancer** (*String*) – The name of the load balancer. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – The name of the load balancer service. Service names must only consist of letters, numbers, underscores and hyphens. Ensure this value has at most 64 characters.
- **protocol** (*String*) – The protocol to be used by the load balacer. Acceptable values: 'tcp', 'http', 'https'.
- **virtual_ip** (*String*) – The virtual IP address. This is the address clients will connect to.
- **port** (*Integer*) – The port number clients will connect to. Ensure this value is greater than or equal to 1. Ensure this value is less than or equal to 65535.
- **enabled** (*Boolean, Optional*) – Whether the service is enabled or not. Defaults to True.

Returns A dictionary of the newly defined service as described in `loadbalancer.service.info()`.

Raises Will raise:

- `ApiErrorDoesNotExist` if the load balancer with the provided name does not exist.

- ApiErrorBadParameters if the service's parameters are invalid.
- ApiErrorNotUnique if the load balancer already has a service with this name.

loadbalancer.service.**info**()
Describe a load balancer service.

Parameters

- **load_balancer** (*String*) – The name of the load balancer. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – The name of the load balancer service. Service names must only consist of letters, numbers, underscores and hyphens. Ensure this value has at most 64 characters.

Returns

A dictionary with the following keys:

name String: Name of the service.

enabled: Boolean: If the service is enabled or not.

virtual_ip: String: The load-balanced IP (that which should be used in DNS records).

port: Integer: The TCP port number.

protocol: String: Service type (e.g. http)

servers: A list of dictionaries as described in loadbalancer.server.info().

Raises Will raise ApiErrorDoesNotExist if the load balancer or named service does not exist.

loadbalancer.service.**list**()
List a load balancer's services.

Parameters **load_balancer** (*String*) – The name of the load balancer. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns A list of dictionaries as described in loadbalancer.service.info().

loadbalancer.service.**remove**()
Remove a service. The service must have no servers attached to it.

Parameters

- **load_balancer** (*String*) – The name of the load balancer. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – The name of the load balancer service. Service names must only consist of letters, numbers, underscores and hyphens. Ensure this value has at most 64 characters.

Returns The name of the removed service.

Raises Will raise:

- ApiErrorDoesNotExist if the load balancer or service with the provided name does not exist.
- ApiErrorPreconditionFailed if there are still servers attached to the service.

loadbalancer.service.**update**()
Update a service's settings. Settings that are not provided will not be updated.

Parameters

- **load_balancer** (*String*) – The name of the load balancer. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service_name** (*String*) – The name of the load balancer service. Service names must only consist of letters, numbers, underscores and hyphens. Ensure this value has at most 64 characters.
- **protocol** (*String, Optional*) – The protocol to be used by the load balacer. Acceptable values: 'tcp', 'http', 'https'.
- **virtual_ip** (*String, Optional*) – The virtual IP address. This is the address clients will connect to.
- **port** (*Integer, Optional*) – The port number clients will connect to. Ensure this value is greater than or equal to 1. Ensure this value is less than or equal to 65535.
- **enabled** (*Boolean, Optional*) – Whether the service is enabled or not.

Returns A dictionary of the updated load balancer service as described in `loadbalancer.service.info()`.

Raises Will raise:

- `ApiErrorDoesNotExist` if the load balancer or service with the provided name does not exist.
- `ApiErrorBadParameters` if the new parameters are invalid.
- `ApiErrorPreconditionFailed` if the virtual IP address is already in use.

1.2.16 Memshell Methods

API for MemShell

`memshell.disable()`
Disables MemShell access on the service. The current password must be specified.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **password** (*String*) – The current Memshell password (as returned by `memshell.enable()`.).

`memshell.enable()`
Enables MemShell access on the service.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **password** (*String, Optional*) – The password set set for authentication. A password will be generated and returned if this parameter is not supplied. Ensure this value has at least 10 characters.

Returns

a dictionary with following keys:

password String: The password used for authentication with Memshell, either the one you passed in or a randomly generated one if you didn't.

memshell.**info**()
Information about the MemShell service.

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns

a dictionary with following keys:

enabled Boolean: True if MemShell access is enabled; otherwise False.

hostname String: The hostname to connect to.

port Integer: The TCP port to connect to

username: String: The username to authenticate as.

memshell.**set_password**()
Sets the Memshell password for the service. Memshell must already be enabled.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **current_password** (*String*) – The current Memshell password (as returned by memshell.enable().).
- **new_password** (*String*) – The new password to set. Ensure this value has at least 10 characters.

1.2.17 Memstore Methods

API for Memstore

These methods are a complement to the standard Memstore API .

memstore.**usage**()
Retrieves information about usage of a Memstore instance.

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns

a dictionary with the following keys:

bytes Float: Space used in the storage in bytes.

containers Integer: Number of containers in the storage.

objs Integer: Number of objects in the storage.

bandwidth

bytes_in Float: Number of bytes in for the last 24 hours.

bytes_out Float: Number of bytes out for the last 24 hours.

requests Float: Number of requests for the last 24 hours.

cdn_bandwidth

bytes_in Float: Number of bytes in the CDN for the last 24 hours.

bytes_out Float: Number of bytes out the CDN for the last 24 hours.

requests Float: Number of requests for the last 24 hours.

1.2.18 Memstore Container Methods

These methods are provided here as a convenient way to manage containers regarding the Access Control List (ACL) and the Content Delivery Network (CDN), but the same functionality can be achieved using the OpenStack Object Storage API.

`memstore.container.acl()`
Retrieve ACL information associated to a container.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **container** (*String*) – Name of the container.

Returns

a dictionary with the following keys:

read_acl List: The usernames (Strings) with read access to the container.

write_acl List: The usernames (Strings) with write access to the container.

The returned lists may be empty if there's no user in the ACL.

`memstore.container.cdn()`
Retrieve CDN information associated to a container.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **container** (*String*) – Name of the container. Container name can only contain numbers, letters and hyphens. Hyphens aren't allowed at the beginning or the end of the name.

Returns

a dictionary with the following keys:

URL String: public URL for the container.

SSL_URL String: public SSL URL for the container.

public Boolean: If the container CDN is enabled (public container).

ttl Integer: Time to live in seconds for content cached by the CDN.

listing Boolean: If directory listing is enabled.

index String: Name of the file used as default index. '0' means the functionality is disabled.

notfound String: Path of the file used for the HTTP 404 (Not Found) page. '0' means the functionality is disabled.

cors String: Cross-Origin Resource Sharing (CORS) header value, being a specific site or '*' for any. '0' means the functionality is disabled.

The CDN information is stored in the container's metadata, other content not listed here is ignored by the method.

`memstore.container.create()`
Create a container with the given name.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **container** (*String*) – Name of the container.

Returns No value if the container is created successfully, or raises an `ApiError`.

`memstore.container.`**`set_acl`**`()`

Set ACL information associated to a container.

Write access implies read access.

When a new ACL is set, the existing ACL is discarded. To clear a list, an empty list will be used.

Any user added to the ACL must be created and enabled in the container's Memstore instance for the ACL to have effect.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **container** (*String*) – Name of the container.
- **read_acl** (*List*) – List of strings with usernames to have read access to the container
- **write_acl** (*List*) – List of strings with usernames to have write access to the container

`memstore.container.`**`set_public_cdn`**`()`

Set a container public enabling the CDN service.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **container** (*String*) – Name of the container. Container name can only contain numbers, letters and hyphens. Hyphens aren't allowed at the beginning or the end of the name.
- **ttl** (*Integer*) – Time to live in seconds for content cached by the CDN.
- **listing** (*Boolean, Optional*) – If directory listing is enabled.
- **index** (*String, Optional*) – Name of the file to be used as default index.
- **notfound** (*String, Optional*) – Path of the file to be used for the HTTP 404 (Not Found) page.
- **cors** (*String, Optional*) – Cross-Origin Resource Sharing (CORS) header value.

Returns the public URL for the container.

Return type String

The CDN information is stored in the container's metadata and any unrelated existing metadata information won't be altered.

`memstore.container.`**`unset_public_cdn`**`()`

Unset a public container disabling the CDN service.

The CDN information is stored in the container's metadata and any unrelated existing metadata information won't be altered.

The container and its data won't be modified.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **container** (*String*) – Name of the container. Container name can only contain numbers, letters and hyphens. Hyphens aren't allowed at the beginning or the end of the name.

1.2.19 Memstore User Methods

These methods allow you to manage Memstore users.

This functionality is not available through the OpenStack Object Storage API.

`memstore.user.create()`
Create a new user in a Memstore instance.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **username** (*String*) – Name of the user. Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.
- **password** (*String*) – User password.
- **enabled** (*Boolean*) – If the user is enabled or not after being created.

Returns a dictionary as described in `memstore.user.info()`.

The user name must be unique for the cloud storage - if not it returns `ApiErrorNotUnique`.

User name can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot.

`memstore.user.delete()`
Delete a user in a Memstore instance.

The *admin* user can't be deleted.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **username** (*String*) – Name of the user. Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.

`memstore.user.disable()`
Disable a user in a Memstore instance.

A disabled user can't login through any of the cloud storage interfaces.

The *admin* user can't be disabled.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **username** (*String*) – Name of the user. Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.

memstore.user.**enable**()
Enable a user in a Memstore instance.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **username** (*String*) – Name of the user. Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.

memstore.user.**info**()
Describe an existing user in a Memstore instance.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **username** (*String*) – Name of the user. Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.

Returns

a dictionary with the following keys:

username String: Name of the user.

enabled: Boolean: If the user is enabled or not.

admin Boolean: If the user is the administrator of the storage.

memstore.user.**list**()
List all existing users in a Memstore instance.

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns a list of dictionaries where each dictionary is as described in memstore.user.info().

The returned user name must be concatenated to the storage name using a dot to get the username for authentication purposes (ie. *msstore1* and *admin* would be *msstore1.admin*).

memstore.user.**set_password**()
Set a new password for an existing user in a Memstore instance.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **username** (*String*) – Name of the user. Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.
- **password** (*String*) – User new password.

1.2.20 Openstack Methods

API for Cloud IaaS (powered by OpenStack) services.

These methods complement the standard OpenStack APIs .

openstack.**list_projects**()
List live Cloud IaaS (powered by OpenStack) projects.

Returns

A dictionary of dictionaries keyed by project name with values with the following keys:

project_id String: The unique id of the project.

auth_url String: The URL of the identity endpoint (Keystone)

firewall_type String: The type of firewalling for this project

firewall_rule_group Dictionary: A dictionary as described in `firewalling.rule_group_info()` if applicable.

openstack.**sync_users**()
Synchronise user changes in relevant OpenStack projects.

Running this will ensure that the current state of your account OpenStack users is synced with relevant OpenStack projects in our cloud.

You may poll the job returned to discover when this has been done. We don't guarantee any particular timescale for this, but normally it will only take a few seconds.

Returns A dictionary as described in `job.status()`.

1.2.21 Openstack Project Methods

These methods allow you to manage OpenStack projects for your account.

These methods complement the standard OpenStack APIs .

openstack.project.**add_user**()
Adds an account OpenStack user to an OpenStack project resulting in access being granted to the user in the Horizon dashboard and via the OpenStack APIs.

Parameters

- **project_id** (*String*) – The tenant/project ID of the OpenStack project Invalid API key. Ensure this value has at most 32 characters.
- **username** (*String*) – The username Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.

openstack.project.**list_users**()
Lists users for an OpenStack project.

Parameters **project_id** (*String*) – The tenant/project ID of the OpenStack project Invalid API key. Ensure this value has at most 32 characters.

Returns a list of dictionaries where each dictionary is as described in

`openstack.user.info()`.

openstack.project.**remove_user**()
Removes an account OpenStack user from an OpenStack project resulting in access being revoked from the user in the Horizon dashboard and via the OpenStack APIs.

Parameters

- **project_id** (*String*) – The tenant/project ID of the OpenStack project Invalid API key. Ensure this value has at most 32 characters.

- **username** (*String*) – The username Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.

1.2.22 Openstack User Methods

These methods allow you to manage OpenStack users for your account.

This functionality is not available through the OpenStack Identity API.

`openstack.user.create()`
Create a new OpenStack user.

Parameters

- **username** (*String*) – The username Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.
- **password** (*String*) – User password.
- **description** (*String, Optional*) – Description of the user Ensure this value has at most 255 characters.
- **email** (*String*) – The email address of the user Must be a valid email address. Ensure this value has at most 255 characters.
- **enabled** (*Boolean*) – If the user is enabled or not after being created.

Returns a dictionary as described in `openstack.user.info()`.

The user name must be unique for your account - if not it returns `ApiErrorNotUnique`.

User name can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. "@<account-name>" will be appended to the user name for use in the Horizon control panel and OpenStack APIs.

`openstack.user.delete()`
Delete an OpenStack user.

Parameters **username** (*String*) – The username Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.

Returns A dictionary as described in `job.status()`.

`openstack.user.disable()`
Disable a OpenStack user.

A disabled user can't login to any OpenStack projects or use the OpenStack APIs.

Ensure that `openstack.sync_users()` is called in order to affect the change in the cloud.

Parameters **username** (*String*) – The username Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.

`openstack.user.enable()`
Enable an OpenStack user.

Ensure that `openstack.sync_users()` is called in order to affect the change in the cloud.

Parameters **username** (*String*) – The username Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.

openstack.user.**info**()
Information for an existing OpenStack user.

Parameters **username** (*String*) – The username Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.

Returns

a dictionary with the following keys:

username String: Name of the user.

description String: Description of the user

email String: Email address of the user

enabled: Boolean: If the user is enabled or not.

projects List of Dictionaries: The OpenStack projects that this user has access to described as a dictionary with the following keys:

name String: The name of the project

id String: The id of the project

openstack.user.**list**()
List all existing OpenStack users for an account.

Returns a list of dictionaries where each dictionary is as described in openstack.user.info().

The returned user name must be prepended to '@<account-name>' to get the keystone username for authentication purposes in the Horizon control panel or OpenStack APIs (e.g. the username *alice* for account *aliciaa* would be *alice@aliciaa*).

openstack.user.**set_password**()
Set a new password for an existing OpenStack user.

Parameters

- **username** (*String*) – The username Username can only contain numbers, letters, dots, dashes and underscores, and can't start with a dot. Ensure this value has at most 50 characters.
- **password** (*String*) – User new password.

Returns A dictionary as described in job.status().

1.2.23 Partner Account Methods

partner.account.**create**()
Creates a new Memset account which will be associated with the calling partner.

Parameters

- **user_email** (*String*) – Email address of the end user. Must be a valid email address.
- **user_title** (*String*) – Title of the end user. Acceptable values

  'Mr' Mr

  'Mrs' Mrs

  'Ms' Ms

  'Miss' Miss

'Dr' Dr

'Rev' Rev

- **user_forename** (*String*) – Forename of the end user. Ensure this value has at most 50 characters.
- **user_surname** (*String*) – Surname of the end user. Ensure this value has at most 50 characters.
- **user_company** (*String, Optional*) – Company of the end user. Ensure this value has at most 200 characters.
- **user_phone** (*String*) – Phone number of the end user. Ensure this value has at most 50 characters.
- **user_address** (*String*) – Address of end user. Ensure this value has at most 250 characters.
- **user_town** (*String*) – Town of end user. Ensure this value has at most 100 characters.
- **user_statecounty** (*String*) – State/county of end user. Ensure this value has at most 100 characters.
- **user_country** (*String*) – ISO 3166-1 alpha-2 code for the country of the end user. Ensure this value has at most 2 characters.
- **user_postcode** (*String*) – Postal code for the address of the end user. Ensure this value has at most 20 characters.
- **account_currency** (*String, Optional*) – Currency to use for billing on the new account, defaults to 'GBP'. Acceptable values

  'GBP' GBP

  'EUR' EUR

  'USD' USD

Returns

A dictionary with the following keys

account the name of the account generated

api_key information about a key used to manage the account, same format as `apikey.info()`

`partner.account.list()`

Returns a list of account names which are associated with the calling partner.

Returns A list of strings which are the names of accounts associated with the calling partner.

1.2.24 Partner Apikey Methods

`partner.apikey.create()`

Create a new key for managing an individual product.

Returns the same as `apikey.info()`.

Raises `ApiErrorAccountDoesNotExist` if *account_name* does not match any of the accounts associated with the calling partner.

Raises `ApiErrorServiceDoesNotExist` if *product* does not match the name of any active product which is managed by the partner, on any account associated with the calling partner.

Parameters

- **account_name** (*String*) – The name of the account to create the API key for. Ensure this value has at most 20 characters.
- **service** (*String*) – The name of the service to scope the API key to. Ensure this value has at most 255 characters.
- **methods** (*List, Optional*) – A list of methods the API key is scoped to, an empty list allows all methods, default allows all.
- **comment** (*String*) – A comment to associate with the API key. Ensure this value has at most 255 characters.

`partner.apikey.list()`
Returns a list of API keys in associated accounts which are enabled.

Raises `ApiErrorAccountDoesNotExist` if *account_name* is provided and does not match the name of any account associated with the calling partner.

Parameters **account_name** (*String, Optional*) – The name of the account to limit results to. Ensure this value has at most 20 characters.

Returns A list of dictionaries in the same format as `apikey.info()`

1.2.25 Partner Service Methods

`partner.service.list()`
Return a list information about partner managed services on associated accounts.

Raises `ApiErrorAccountDoesNotExist` if *account_name* is provided and does not match the name of any account associated with the calling partner.

Parameters **account_name** (*String, Optional*) – The name of the account to limit results to. Ensure this value has at most 20 characters.

Returns A list of dictionaries with the same keys as `service.info()`.

1.2.26 Payment Method Methods

API for Payment Methods.

`payment_method.available()`
Returns a list of available payment methods for this account.

Returns A list of dictionaries as described in `payment_method.info()`

`payment_method.info()`
Describes a payment method.

Parameters **payment_method_id** (*String*) – the unique id of the payment method

Returns

A dictionary with the following keys:

id String: A unique id to identify this payment method

type String: One of 'account', 'card', 'bank_transfer', 'cheque', 'paypal', 'transfer_or_cheque', or 'notset'.

default Boolean: Indicates whether this is the account's default payment method.

The following keys are only included where the payment method is a card or billing account.

status String: Either 'AVAILABLE' or 'EXPIRED'.

name String: A nickname that the customer gave the payment method.

payment_method.**remove**()
Remove a payment card from the account.

To remove a payment method that is the default payment method for the account, you must first set the default to a different payment method using payment_method.set_default().

Parameters **payment_method_id** (*String*) – the unique id of the payment method

Returns A dictionary as described in payment_method.info(), with status='DELETED'.

payment_method.**set_default**()
Set the default payment method for the account.

Parameters **payment_method_id** (*String*) – the unique id of the payment method

Returns A dictionary as described in payment_method.info()

1.2.27 Server Methods

API Methods for dealing with servers

server.**info**()
Describes the server returning a dictionary.

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns

A dictionary with the following keys:

name String: name of this server (same as service name), e.g. "testaa1".

host_name String: host name for this server, e.g. "testaa1.miniserver.com".

primary_ip String: primary IP address for this server, e.g. "1.2.3.4".

ips List of Dictionaries: All the IP addresses assigned to this server described as a dictionary with the following keys:

address String: IP address, e.g. "1.2.3.4".

reverse_map String: Reverse mapping for IP address, e.g. "server.example.com".

bytes_in_yesterday Integer: Bytes received yesterday to this IP address.

bytes_out_yesterday Integer: Bytes sent yesterday from this IP address.

bytes_in_today Integer: Bytes received today so far to this IP address.

bytes_out_today Integer: Bytes sent today so far from this IP address.

vlans

Dictionary with the following keys:

tagged: List of Strings: The names of all the tagged vLANs this server is in.

untagged: List of Strings: The names of all the untagged vLANs this server is in.

os String: describing the Operation System installed, e.g. "debian_wheezy_64", "win2012serverstd_r2_64".

backups Boolean: whether this server has a backup service.

control_panel String: describing the control panel, e.g. "none", or "cpanel".

firewall_type String: describing the firewall type, e.g. "none", "basic", "self_managed" or "managed".

firewall_rule_group Dictionary: describing the firewall rule group applied to the server as returned by `firewalling.rule_group_info()`. Empty if `firewall_type` is "none".

monitoring_level Strong: describing the monitoring level, "basic", "advanced" or "managed".

monitor Boolean: whether we are monitoring this server.

ignore_monitoring_off Boolean: the customer has acknowledged that they aren't being monitored.

no_nrpe Boolean: Disable NRPE agent for this server.

no_auto_reboot Boolean: Don't auto reboot this server.

support_level String: support level "infrastructure_only", "basic" or "managed".

vulnscan String: whether this server is vulnerability scanned, 'none' = no scanning, 'basic' = Self-monitored, 'managed' = Memset-monitored.

intrusion_detection String: The intrusion detection support level for this server e.g. "none", "basic" = Self-monitored, "monitored" = Memset-monitored or "protected" = Memset-protected

intrusion_detection_alert_level Integer: The alert level that is set for Intrusion Detection (1-15) or 0 (intrusion_detection="none")

The keys as described in `service.info()` will also be returned.

`server.`**`list`**`()`

List all the servers for an account.

If the status isn't passed in then it will find all the servers with status=LIVE.

Parameters **status** (*String, Optional*) – Status of the server to look for. Acceptable values: 'LIVE', 'CANCELLED'.

Returns Returns a list of dictionaries where each dictionary is as described in func:*server_info*.

`server.`**`move_ips`**`()`

Moves a list of public IPv4 address from one server to another.

This cannot be used to move IPs that are private, or where the source or destination server are in assigned to a load balancer.

The list of IPs will need to be configured on the host once the job has completed.

If the destination host is a Linux Miniserver, the IPs can be automatically applied by triggering a reboot with `server.reboot()`.

Please note that if the destination host is a Miniserver without a vLAN, it will require a reboot using `server.reboot()` for the new IPs to be routed correctly.

The source and destination servers will be unavailable for a short period during this move.

Parameters

- **source_ips** (*List*) – List of public IP addresses to be moved.
- **destination_name** (*String*) – Name of the server. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns A dictionary as described in `job.status()`.

Raises `ApiErrorDoesNotExist` if the *destination_name* or any ip in *source_ips* cannot be found.

`ApiErrorBadParameters` if an empty list of IPs is provided.

`ApiErrorPreconditionFailed` will be raised if:

- Any server is in a load balancer.
- Any IP cannot be moved to the destination network zone.
- Any of the supplied IPs are private.
- If any IP is the primary IP of a server.

`server.`**`reboot`**`()`

Schedules a reboot for name

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **email_address** (*String, Optional*) – Email address for response to reboot, uses admin contact email if not passed in. Must be a valid email address.

Returns A dictionary as described in `job.status()`.

Raises May raise `ApiErrorAlreadyInProgress` if a reboot is pending already.

`server.`**`reimage_from_snapshot`**`()`

Schedules re-imaging of name.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **storage_name** (*String*) – Name of the storage service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **image_type** (*String*) – Snapshot image type. Acceptable values: ‘raw’, ‘tar’, ‘ntfsclone’.
- **snapshot_path** (*String*) – Path to the snapshot (check notes).

Returns A dictionary as described in `job.status()`.

Raises May raise one of the following exceptions:

- `ApiErrorAlreadyInProgress` if a re-imaging is pending already.
- `ApiErrorBadParameters` if the requested image_type is not available for that server or the snapshot name isn’t valid.
- `ApiErrorMethodNotFound` if the server doesn’t support re-imaging.
- `ApiErrorPreconditionFailed` if server current status isn’t valid for re-imaging.
- `ApiErrorPreconditionFailed` if the server and the cloud storage are in different data zones.

This method is only available on Miniservers.

The snapshot path parameter depends on the image type:

- **raw images**: the path to the .raw.gz image.
- **ntfsclone images**: the path to the snapshot directory (it must contain the .XMBR file and all the .img.gz files).
- **tar images**: the path to the .tar image.

When using a **raw image**, it must contain an instance of the same type of operating system as currently installed on the server. For example: re-imaging a Linux based server with a Microsoft Windows raw image will result in a broken system.

An email will be sent to the admin contact when the process is finished.

See also `server.snapshot_list()` for retrieving a list of valid snapshots from a storage.

`server.`**`reimage_from_stock_image`**`()`

Schedules re-imaging of name.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **os** (*String*) – The Operating System name.
- **os_bits** (*String, Optional*) – Whether to use a 32 or 64-bit system. Note that some operating systems may not be available in both versions. Acceptable values: '32', '64'.
- **pub_ssh_key** (*String, Optional*) – Public SSH key to be installed in the server (*root* user). Only available when reimaging a Linux server.

Returns A dictionary as described in `job.status()`.

Raises May raise one of the following exceptions:

- `ApiErrorAlreadyInProgress` if a re-imaging is pending already.
- `ApiErrorBadParameters` if the requested operating system is not available for that server, the operating system name isn't valid or the provided public SSH key isn't valid.
- `ApiErrorMethodNotFound` if the server doesn't support re-imaging.
- `ApiErrorPreconditionFailed` if server current status isn't valid for re-imaging.

This method is only available on Miniservers.

The *os* and *os_bits* parameters are as described in *Create Methods*.

You can manage public SSH keys on the Manage SSH Keys page in the Memset control panel.

`server.`**`set_intrusion_detection_alert_level`**`()`

Sets the Intrusion Detection email alert level for name.

Refer to the Intrusion Detection page for alert levels.

Raises May raise `ApiErrorPreconditionFailed` if Intrusion Detection is not enabled on this server.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

- **alert_level** (*Integer*) – The required new Intrusion Detection alert level. Ensure this value is greater than or equal to 1. Ensure this value is less than or equal to 15.

server.**slave_ns_add**()

Add a domain to the Memset name server slaves for the primary name server `name`.

It may take up to 30 minutes for the domain to be added.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **domain_name** (*String*) – The slaved domain name. Must be a valid host name.

Returns The added domain name

Raises

May raise one of the following exceptions:

- `ApiErrorDoesNotExist` if `name` cannot be found for the account.
- `ApiErrorPreconditionFailed` if the domain is already slaved.

server.**slave_ns_delete**()

Delete a domain from the Memset name server slaves for the primary name server `name`.

It may take up to 30 minutes for the domain to be deleted.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **domain_name** (*String*) – The slaved domain name. Must be a valid host name.

Returns The deleted domain name.

Raises `ApiErrorDoesNotExist` if `name` cannot be found for the account or slaved `domain_name` not found for `name`.

server.**slave_ns_list**()

List all domain names being slaved by Memset's name server slaves for the primary name server `name`.

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns A list of domain names.

Raises `ApiErrorDoesNotExist` if `name` cannot be found for the account.

server.**snapshot**()

Schedules a snapshot for name.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **storage_name** (*String*) – Name of the storage service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **image_type** (*String*) – Snapshot image type. Acceptable values: 'raw', 'tar', 'ntfsclone'.

Returns A dictionary as described in `job.status()`.

Raises May raise one of the following exceptions:

- ApiErrorAlreadyInProgress if a snapshot is pending already.
- ApiErrorBadParameters if the requested image_type is not available for that server.
- ApiErrorMethodNotFound if the server doesn't support snapshots.
- ApiErrorPreconditionFailed if server current status isn't valid for taking snapshots.
- ApiErrorPreconditionFailed if the server and the cloud storage are in different data zones.

This method is only available on Miniservers.

•"raw" - Supported for Miniservers running Microsoft Windows or a Linux operating system.

•"tar" - A deprecated image type for use with our classic Miniservers running a Linux operating system.

•"ntfsclone" - A deprecated image type for Miniservers running a Microsoft Windows operating system.

Please be advised that classic Miniservers are deprecated and won't be supported in future API releases.

server.**snapshot_delete**()

Deletes a snapshot.

Parameters

- **storage_name** (*String*) – Name of the storage service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **snapshot_path** (*String*) – Path to the snapshot as returned by server.snapshot_list method. Ensure this value has at most 255 characters.

Returns A dictionary as described in job.status().

Raises May raise ApiErrorAlreadyInProgress if there is a pending deletion for that snapshot.

The snapshots are expected to be in a container named 'miniserver-snapshots', so the path to the snapshot is relative to this container.

See also server.snapshot_list() for retrieving a list of valid snapshots from a storage.

server.**snapshot_list**()

Lists snapshots found in the storage identified by storage_name.

Parameters **storage_name** (*String*) – Name of the storage service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns

An array of dictionaries with the following keys:

snapshot_path String: full path to the snapshot.

image_type String: "raw", "ntfsclone" or "tar".

os_type String: "linux", "windows", "freebsd", etc; or "unknown" in case it can't be identified.

os_option String: operating system option to be used to provision a Miniserver using this snapshot.

description String: description of the snapshot.

user_comment String: user provided comment (if any) or none.

last_modified Date: last modification date of the snapshot.

Raises May raise `ApiErrorDoesNotExist` if no snapshots are found.

The snapshots are expected to be in a container named 'miniserver-snapshots'. This function doesn't guarantee that found snapshots are correct.

`server.upgrade()`

Upgrades a Miniserver instance to a higher specification.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **new_sku** (*String*) – The SKU you wish to upgrade to. Acceptable values for Miniserver VM VPS packages (monthly billing) are

sku	description

Acceptable values for Miniserver cloud compute (billed hourly) are

sku	description

- **dry_run** (*Boolean*) – If True, then the service is not provisioned but the information is still returned.
- **upgrade_disk** (*Boolean, Optional*) – If set to False, the disk won't be upgraded. If you don't change your disk you can reduce the time required for the upgrade and later you'll have the option to downgrade your Miniserver if you need to.

Returns

A dictionary with the following keys

new_sku String: The new SKU of the server.

dry_run Boolean: As specified in the API call.

currency String: The currency of the order. You cannot specify the currency in the API request, the account's currency will be used. Currently, the account currency must be changed via the Memset website. Returned value will be one of 'GBP', 'USD', or 'EUR'.

net_setup_price Float: There will be a one-off setup charge to cover the rental increase until the service expiry date, net of VAT. Only for Miniserver VM VPS Packages.

net_increase Float: The increase of the monthly rental for the service, net of VAT. Only for Miniserver VM VPS packages.

net_hourly_inarrears_rental_price Float: The new hourly rate for the service, net of VAT. Only for Miniserver cloud compute.

invoice_ref Integer: The invoice reference for the order. Will not be provided if the method is called with `dry_run=True`.

job Dictionary: Provides the status of the job to set up the requested service. Will not be provided when the method is called with `dry_run=True`. See `job.status()` for further details.

This method is only available on Miniservers.

See the following pages for princing and more information:

•Miniserver VM VPS Packages

•Miniserver cloud compute

1.2.28 Server Monitoring Methods

Server Monitoring API.

server.monitoring.**get_policy**()
Retrieve the current policy for monitoring of named server.

See Also:

server.monitoring.set_policy()

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns A dictionary of flags corresponding to those set by server.monitoring.set_policy().

server.monitoring.**rule_create**()
Create a monitoring rule for name.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **service** (*String*) – The monitoring service. Acceptable values

  'http' HTTP / Apache

  'http_string' HTTP string

  'ssh' SSH access

  'mysql' MySQL (external connectivity)

  'mysql_slave' MySQL (slave)

  'smtp' SMTP (email)

  'pop3' POP3 (email)

  'imap' IMAP (email)

  'tcp' Other TCP port

  'disk' Disk space

  'load' Server load

  'ping' Ping

  'ssl_cert' SSL certificate validity
- **alert_type** (*String*) – The alert type. Note: memset-warning, memset-alert only permitted for Server Monitoring Advanced Memset-monitored customers. Acceptable values

  'email' Email to:

  'sms' SMS to:

  'memset-warning' Warn Memset

  'memset-alert' Memset Alert 24x7
- **contact_address** (*String, Optional*) – The email address or telephone number to which alerts should be sent. Required for non Memset-monitored alert types.

- **port** (*Integer, Optional*) – The port number to monitor. Required only for *service* 'tcp' or 'http_string'.
- **host_name** (*String, Optional*) – The host name or URL to check. Required only for *service* 'http_string'.
- **http_check_string** (*String, Optional*) – The string to check for at *host_name*. Required only for *service* 'http_string'.

Returns a dictionary with the new rule information in as described in `server.monitoring.rule_info()`

Raises `ApiErrorBadParameters` if the correct combination of parameters have not been provided.

Raises `ApiErrorPreconditionFailed` when attempting to set up alerts not within the server's Monitoring support level, or rule quota reached.

`server.monitoring.`**`rule_delete`**`()`

Delete the monitoring rule with rule_id for name

Raises `ApiErrorBadParameters` if the *rule_id* is not valid.

Raises `ApiErrorDoesNotExist` if the *rule_id* does not exist for this account.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **rule_id** (*String*) – The id of the monitoring rule

`server.monitoring.`**`rule_info`**`()`

Info about a monitoring rule

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **rule_id** (*String*) – The id of the monitoring rule

Returns

a dictionary with the following keys

rule_id String: The unique identifier for this rule.

service String: The monitoring service for this rule.

alert_type String: The alert type configured for this rule.

contact_address

String: The email address or telephone number to which alerts for this monitoring rule are sent. It will be e
the alert_type is a Memset-managed alert type.

port Integer: It will be empty for a standard *service* e.g. 'http'.

host_name String: The host name or URL being checked. It will be empty if the *service* is not 'http_string'.

http_check_string String: The string being checked. It will be empty if the *service* is not 'http_string'.

Raises `ApiErrorDoesNotExist` if the *rule_id* does not exist for this account.

server.monitoring.**rule_list**()
Provides information on Server Monitoring rules for name

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns list of dictionaries as described in server.monitoring.rule_info().

server.monitoring.**rule_update**()
Update the monitoring rule with rule_id for name

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **rule_id** (*String*) – The id of the monitoring rule
- **service** (*String, Optional*) – The monitoring service. Acceptable values

  'http' HTTP / Apache

  'http_string' HTTP string

  'ssh' SSH access

  'mysql' MySQL (external connectivity)

  'mysql_slave' MySQL (slave)

  'smtp' SMTP (email)

  'pop3' POP3 (email)

  'imap' IMAP (email)

  'tcp' Other TCP port

  'disk' Disk space

  'load' Server load

  'ping' Ping

  'ssl_cert' SSL certificate validity
- **alert_type** (*String, Optional*) – The alert type. Note: memset-warning, memset-alert only permitted for Server Monitoring Advanced Memset-monitored customers. Acceptable values

  'email' Email to:

  'sms' SMS to:

  'memset-warning' Warn Memset

  'memset-alert' Memset Alert 24x7
- **contact_address** (*String, Optional*) – The email address or telephone number to which alerts should be sent. Required for non Memset-monitored alert types.
- **port** (*Integer, Optional*) – The port number to monitor. Required only for *service* 'tcp'.
- **host_name** (*String, Optional*) – The host name or URL to check. Required only for *service* 'http_string'.
- **http_check_string** (*String, Optional*) – The string to check for at *host_name*. Required only for *service* 'http_string'.

Returns a dictionary with the rule information in as described in `server.monitoring.rule_info()`

Raises `ApiErrorBadParameters` if the *rule_id* is invalid or the updated rule is invalid.

`server.monitoring.`**`set_policy`**`()`

Set the monitoring policy for the named server.

See Also:

`server.monitoring.get_policy()`

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **monitoring** (*Boolean*) – Whether or not monitoring is enabled for the named server.
- **auto_reboot** (*Boolean, Optional*) – Whether or not to automatically reboot the named server if monitoring determines that it has becomes unresponsive. This defaults to true when monitoring is *enabled*.

  Note: This flag can only be set when when monitoring is *enabled*, trying to set it to true when this is not the case will result in an API error.

- **reminder_emails** (*Boolean, Optional*) – Whether or not to send a reminder at regular intervals about monitoring being disabled. The reminder is sent to the email address associated with your account. This defaults to true when monitoring is *disabled*.

  Note: This flag can only be set when when monitoring is *disabled*, trying to set it to true when this is not the case will result in an API error.

Returns A dictionary as described in `job.status()`.

Raises Will raise `ApiErrorBadParameters` for incompatible combinations of flags.

Raises Will raise `ApiErrorAlreadyInProgress` if a policy change is already in progress.

1.2.29 Server Status Methods

Server Status API.

`server.status.`**`info`**`()`

Return all monitoring statuses for a server.

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns

A dictionary with the following keys:

name: String: Name of the server, e.g. "testaa1".

service: String: Name of service being monitored.

status: String: Status of the service last time it was checked: "OK", "WARNING", "CRITICAL" or "UNKNOWN".

OK The service was checked successfully and it appeared to be functioning properly.

WARNING The service was checked successfully, but it appeared to be above some "warning" threshold or did not appear to be working properly.

CRITICAL The service was not running or it was above some "critical" threshold.

UNKNOWN The monitoring agent failed or some other low-level error prevented the completion of the status check.

last_checked: Date: Date and time when the service was last checked.

detail: String: Additional information about the status when last checked.

`server.status.list()`

List all monitoring status information for servers in the account.

Parameters

- **names** (*List, Optional*) – Return monitoring status information for these names. If not provided, the status information of all live servers in the account is returned.
- **statuses** (*List, Optional*) – Return monitoring status information in these statuses. Valid statuses are: OK, WARNING, CRITICAL, UNKNOWN.

Returns A list of dictionaries as described in `server.status.info()`.

1.2.30 Service Methods

API for Services

`service.cancel()`

Cancels a service.

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns A dictionary as described in `job.status()`.

Raises May raise `ApiErrorAlreadyInProgress` if the service is pending cancellation.

A cancellation invoice may be generated.

SSL Certificates are not supported and can't be cancelled with this method.

`service.info()`

Describes the service returning a list of dictionaries.

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns

A dictionary with the following keys:

name String: The canonical name of this service. For servers this is also the server name.

status String: The status of the service as a string e.g. "LIVE", "ONHOLD", "CANCELLED".

nickname String: The nickname of the service - set by the customer, defaults to "".

start_date Date: The start date of the service.

expiry_date Date: The end date of the service.

renewal_price_amount Float: The monetary amount for the monthly renewal including VAT.

renewal_price_currency String: The currency for the renewal, e.g. "GBP".

renewal_price_vat Float: The monetary amount for VAT on the monthly renewal.

type String: The type of this service, e.g. "fullserver", "miniserver", etc.

network_zones List of strings: The names of all the network zones associated to this service (if any). Some services are network zone independent.

data_zone String: The name of the data zone this service is in (if applicable). Some services that don't have data at rest are data zone independent and this field will be empty.

`service.list()`
List all the services for an account.

Parameters

- **type** (*String or List, Optional*) – Search only for this type of service, by default all services are returned.
- **status** (*String, Optional*) – Status of the service to look for, default 'LIVE'. Acceptable values: 'LIVE', 'CANCELLED'.

Returns a list of dictionaries where each dictionary is as described in `service.info()`.

`service.set_nickname()`
Updates the service nickname.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **nickname** (*String*) – New nickname for the service.

Returns

a dictionary with the following keys:

nickname String: The new nickname for the service.

This enforces that the nickname must be unique for all the services of the same account.

The nickname must be made up of the characters [A - Z a - z 0 - 9 _ . - ']. Any characters not in this range will be deleted. It must be 255 characters or fewer - any extra will be truncated. Any leading or trailing whitespace will be removed.

It must be unique for the services in the account - if not it returns `ApiErrorNotUnique`.

The nickname set is returned.

1.2.31 VLAN API Methods

API for VLANs

`vlan.add()`
Adds a host to the VLAN.

A host can be added to several VLANs. The first one is considered "primary" as is the VLAN that will used for the default route. The primary VLAN can be replaced by using the *primary* option in this method.

The primary VLAN can't be changed if the host is providing a service in a load balancer using that same VLAN.

When a host is added to a VLAN, the network infrastructure is changed automatically and as consequence the host may lose connectivity if the required configuration changes are not performed.

If needed, a private IP in the VLAN network will be allocated and assigned to the host to be used by Memset's automated tools.

In the case of Miniservers, a reboot using `server.reboot()` is required so any new interface and the VLAN configuration is available to the host.

By default automatic host configuration is supported for all Miniservers using Memset's automated tools. For unmanaged Miniservers and Fullservers, manual configuration changes may be required.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **host** (*String*) – Name of the host service Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **primary** (*Boolean, Optional*) – Replace the primary VLAN on the host. Unless this parameter is set, the default behaviour in Miniservers in case the host is already in a VLAN is to add a new interface to configure the new VLAN.

Returns the output of `vlan.list()` after adding the host.

Raises May raise `ApiErrorPreconditionFailed` if the primary VLAN can't be changed.

`vlan.`**`info`**`()`

Describes the VLAN returning a dictionary.

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns

A dictionary with the following keys:

name String: The canonical name of the VLAN. This is also the service name.

network String: The network address assigned to the VLAN in CIDR format, e.g. "10.1.1.0/24".

hosts List of Strings: The names of all hosts associated to this VLAN.

network_zone String: The network zone this VLAN is in, e.g. "reading".

`vlan.`**`list`**`()`

Lists information about hosts associated to this VLAN returning a list of dictionaries.

Parameters **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns

A list of dictionaries with the following keys:

name String: The canonical name of the service.

type String: The type of host, e.g. "fullserver", "miniserver", etc.

private_ips List of Strings: The private IP addresses from the VLAN network that are associated to the host, e.g. "10.1.1.10". This list is used by Memset's automated tools and is based on the information stored in our database. It can be empty if there are no associated IPs.

tagged Boolean: Whether this VLAN is a tagged VLAN or not.

vlan.**remove**()
Removes a host from the VLAN.

The host will be removed from the VLAN unless one of the following conditions apply:

•The host is a Fullserver and this is the only VLAN it is in.

•The server is providing a service in a load balancer in this VLAN.

When a host is added to a VLAN, the network infrastructure is changed automatically and as consequence the host may lose connectivity if the required configuration changes are not performed.

Once the host is removed from the VLAN it is possible that the private address will not be in use any more. That private IP address won't be removed automatically to avoid possible configuration problems in the host's services. Please contact support if you want a private IP address to be deallocated.

In the case of Miniservers, a reboot using `server.reboot()` is required so any new interface and the VLAN configuration is available to the host.

By default automatic host configuration is supported for all Miniservers using Memset's automated tools. For unmanaged Miniservers and Fullservers, manual configuration changes may be required.

Parameters

- **name** (*String*) – Name of the service. Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.
- **host** (*String*) – Name of the host service Service names are 1-64 characters A-Z, a-z, 0-9, -, . and _. Ensure this value has at most 64 characters.

Returns the output of `vlan.list()` after removing the host.

Raises May raise `ApiErrorPreconditionFailed` if the host can't be removed.

1.2.32 Website Firewall Methods

Api for managing a Sucuri Website Firewall

website_firewall.**clear_cache**()
Clear the cache of a list of paths for a site protected by a Website Firewall. If 'paths' isnt given, clear the entire cache.

Parameters

- **domain** (*String*) – The domain of the Website Firewall Platform. Must be a valid domain/subdomain
- **paths** (*List, Optional*) – The paths to be cleared from the cache, eg ["/path1.html", "logo.png"]. You can only clear a maximum of 500 pages in one request.

Returns A dictionary as described in `job.status()`.

1.3 Errors

exception ApiError
Non Specified General API Error

•Error code: 1

•HTTP Status: 400

exception `ApiErrorInternalError`
Internal error - something bad happened

- Error code: 2
- HTTP Status: 500

exception `ApiErrorDoesNotExist`
Object does not exist

- Error code: 3
- HTTP Status: 404

exception `ApiErrorAccountDoesNotExist`
No account exists with the given constraints

- Error code: 14
- HTTP Status: 404

exception `ApiErrorServiceDoesNotExist`
No service exists with the given constraints

- Error code: 15
- HTTP Status: 404

exception `ApiErrorForbidden`
Forbidden - Authentication is incorrect

- Error code: 4
- HTTP Status: 403

exception `ApiErrorMethodNotFound`
Method not found

- Error code: 5
- HTTP Status: 400

exception `ApiErrorAlreadyInProgress`
Request is already in progress

- Error code: 6
- HTTP Status: 400

exception `ApiErrorBadParameters`
Bad parameters

- Error code: 7
- HTTP Status: 400

exception `ApiErrorNotUnique`
Setting is not unique

- Error code: 8
- HTTP Status: 400

exception `ApiErrorPreconditionFailed`
A precondition failed

- Error code: 9

- HTTP Status: 412

exception ApiErrorNotAuthorized
Not Authorized - Authentication is required

- Error code: 10
- HTTP Status: 401

exception ApiErrorPartnerOnly
This method is only available to Memset partners

- Error code: 16
- HTTP Status: 401

exception ApiErrorPartnerRestriction
This method is not available for partners on their child account

- Error code: 17
- HTTP Status: 401

exception ApiErrorServiceTemporarilyUnavailable
Service Temporarily Unavailable

- Error code: 11
- HTTP Status: 503

exception ApiErrorThrottled
Throttled

- Error code: 12
- HTTP Status: 403

exception ApiErrorOverquota
Overquota

- Error code: 13
- HTTP Status: 403

1.4 Memset API Shell

The *Memset API interactive shell* (**ma-shell** for short) is a full featured XMLRPC client that enables remote command invocation from a text terminal.

It can run a single command or an interactive session, supports JSON and XML output as well as output redirection into a file.

1.4.1 ma-shell Invocation

ma-shell can be invoked with the **-h** flag to get an online help screen:

```
Usage: ma-shell [options] [command [param_name param_value ...]]

Memset API interactive shell

Options:
```

```
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -u URL, --url=URL     API URL (default:
                                          https://%s:@api.memset.com/v1/xmlrpc)
  -k KEY, --api-key=KEY
                                          API KEY (required)
  -x, --xml             Use XML as output format (default: JSON)
```

The only required parameter is **-k KEY** to provide a valid API key. In all the following examples API_KEY_HEX is used instead of a real API key.

If no command is provided, **ma-shell** will enter the interactive mode advertising the version number and displaying a command prompt that shows the tail of the used key:

```
(Memset API shell v0.2)
[...a32a1fe9]>
```

The interactive session can be closed with the **quit** command or with **^D** (end of file).

If a command is provided, **ma-shell** executes it and writes the result to standard output before exiting:

```
$ ./ma-shell.py -k API_KEY_HEX help system.listMethods
List all the public methods
```

The output can be redirected to a file using standard shell redirection.

The method invocation will use JSON by default as the output format for structured responses. XML output can be obtained using **-x** flag:

```
./ma-shell.py -k API_KEY_HEX -x service.info name myserver1
<result>
  <status>LIVE</status>
  <renewal_price_currency>GBP</renewal_price_currency>
  <type>miniserver</type>
  <name>myserver1</name>
  <expiry_date>20110917T00:00:00</expiry_date>
  <renewal_price_amount>21.60</renewal_price_amount>
  <nickname>www</nickname>
  <start_date>20110617T00:00:00</start_date>
  <renewal_price_vat>4.34</renewal_price_vat>
</result>
```

Note that when an interactive session has been started, the output format can't be changed.

1.4.2 Command Invocation

There are two local commands:

- **quit**: exit the interactive session.
- **help [command]**: used to describe commands.

The **help** command without argument will display a general help screen. Use **help remote** to get the list of remote commands.

The remote command list is retrieved from the API service at session start.

When **help** is used with a remote command, the information is obtained directly from the API service.

Remote Command Invocation

Remote commands without parameters can be executed as-is using the command name, for example:

```
[...a32a1fe9]> service.list
(service.list result)
[
  {
        "status": "LIVE",
        "renewal_price_currency": "GBP",
        "type": "cloudstorage",
        "name": "mystorage1",
        "expiry_date": "20111112T00:00:00",
        "renewal_price_amount": 24.0,
        "nickname": "",
        "start_date": "20110617T00:00:00",
        "renewal_price_vat": 4.0
  },
  {
        "status": "LIVE",
        "renewal_price_currency": "GBP",
        "type": "miniserver",
        "name": "myserver1",
        "expiry_date": "20110917T00:00:00",
        "renewal_price_amount": 21.60,
        "nickname": "www",
        "start_date": "20110617T00:00:00",
        "renewal_price_vat": 4.34
  }
]
```

In the case of remote commands that require parameters, the parameters must be provided in name/value pairs. The number and name of the parameters depends on the remote command and is specified on its documentation.

For example, `service.info()` requires a single parameter named **name**. Its invocation is as follows:

```
./ma-shell.py -k API_KEY_HEX service.info name myserver1
{
  "status": "LIVE",
  "renewal_price_currency": "GBP",
  "type": "miniserver",
  "name": "myserver1",
  "expiry_date": "20110917T00:00:00",
  "renewal_price_amount": 21.60,
  "nickname": "www",
  "start_date": "20110617T00:00:00",
  "renewal_price_vat": 4.34
}
```

With commands that require more than one parameter, the order is not important as long as the name/value pairs are correct.

For example, `service.set_nickname()` requires two different parameters:

```
[...a32a1fe9]> service.set_nickname name myserver1 nickname webserver
(service.set_nickname result)
{
  "nickname": "webserver"
}
```

The result would be the same if the **nickname** parameter were first.

By default all parameter values are treated as strings, and a **(type)** prefix can be used to indicate the type of a specific value. The following types are supported:

- **(string)**: a string value (default).
- **(int)**: an integer value.
- **(float)**: a decimal number value.
- **(boolean)**: a boolean value (True or False, 1 or 0).
- **(list)**: a comma separated list of strings.

For example, using a list in the **type** parameter of `service.list()`:

```
[...a32a1fe9]> service.list type (list) cloudstorage,miniserver
(service.list result)
[
  {
        "status": "LIVE",
        "renewal_price_currency": "GBP",
        "type": "miniserver",
        "name": "myserver1",
        "expiry_date": "20110917T00:00:00",
        "renewal_price_amount": 21.60,
        "nickname": "www",
        "start_date": "20110617T00:00:00",
        "renewal_price_vat": 4.34
  },
  {
        "status": "LIVE",
        "renewal_price_currency": "GBP",
        "type": "cloudstorage",
        "name": "mystorage1",
        "expiry_date": "20111112T00:00:00",
        "renewal_price_amount": 24.0,
        "nickname": "",
        "start_date": "20110617T00:00:00",
        "renewal_price_vat": 4.0
  }
]
```

The output of any specific remote command invocation can be stored in a file using the pipe (|) operator followed by the desired file name:

```
[...a32a1fe9]> service.list | out.json
(service.list result)
[
  {
        "status": "LIVE",
        "renewal_price_currency": "GBP",
        "type": "cloudstorage",
        "name": "mystorage1",
        "expiry_date": "20111112T00:00:00",
        "renewal_price_amount": 24.0,
        "nickname": "",
        "start_date": "20110617T00:00:00",
        "renewal_price_vat": 4.0
  },
```

```
  {
        "status": "LIVE",
        "renewal_price_currency": "GBP",
        "type": "miniserver",
        "name": "myserver1",
        "expiry_date": "20110917T00:00:00",
        "renewal_price_amount": 21.60,
        "nickname": "www",
        "start_date": "20110617T00:00:00",
        "renewal_price_vat": 4.34
  }
]
(result piped to out.json)
```

The pipe is not needed when running one single command, and the shell redirection can be used:

```
./ma-shell.py -k API_KEY_HEX service.list > out.json
```

1.4.3 Code and License

ma-shell is open source in terms of **MIT** license and can be obtained for free: https://github.com/memset/ma-shell.

1.5 Changelog

1.5.1 0.20.1 - 2019-10-02

- **Changes**
  - The *path* parameter of `website_firewall.clear_cache()` has been replaced with a *paths* parameter which now takes a list of up to 500 paths.

1.5.2 0.20.0 - 2019-07-22

- **Changes**
  - Added *website_firewall* to API
  - **New methods**
    - `website_firewall.clear_cache()`

1.5.3 0.19.0 - 2019-05-28

- **Changes**
  - All *squirrelsave* methods have been removed and are no longer accessible
  - **Removed methods**
    - `squirrelsave.info()`
    - `squirrelsave.keys()`
    - `squirrelsave.list()`

1.5.4 0.18.0 - 2019-02-04

- **Changes**
  - For the *support_level* parameter of `create.monthly_miniserver()` and `create.monthly_fullserver()`, the options *managed_infrastructure* and *managed_platform* have been superceded by *standard* and *premium* respectively. The old values are still accepted.

1.5.5 0.17.0 - 2019-01-27

- **Changes**
  - The *penetration_patrol* parameter for `create.monthly_miniserver()` and `create.monthly_fullserver()` has been changed to *intrusion_detection*.
  - The keys *penetration_patrol* and *penetration_patrol_alert_level* returned from `server.info()` have been changed to *intrusion_detection* and *intrusion_detection_alert_level*.
  - `server.set_penetration_patrol_alert_level()` been deprecated and changed to `server.set_intrusion_detection_alert_level()`. It can still be used as an alias to `server.set_intrusion_detection_alert_level()`. It will be removed in a future release.

1.5.6 0.16.0 - 2018-04-03

- **Changes**
  - The following parameters in `create.monthly_miniserver()` and `create.monthly_fullserver()` are no longer deprecated: *firewall*, *firewall_rule_group*, *monitoring_level*, *penetration_patrol*, *vulnscan*.

1.5.7 0.15.0 - 2018-02-13

- **New methods**
  - `openstack.list_projects()` to list live projects

1.5.8 0.14.0 - 2018-01-06

- **New methods relating to Cloud IaaS services**
  - `openstack.sync_users()` to synchronise account users to the cloud
  - `openstack.user.create()` to create a new user
  - `openstack.user.set_password()` to set the password for a user
  - `openstack.user.disable()` to disable a user in the cloud
  - `openstack.user.enable()` to enable a user in the cloud
  - `openstack.user.delete()` to delete a user
  - `openstack.user.info()` to retrieve information about a user
  - `openstack.user.list()` to list all users
  - `openstack.project.add_user()` to add a user to a project

  - `openstack.project.remove_user()` to remove a user from a project
  - `openstack.project.list_users()` to list users associated with a project

1.5.9 0.13.0 - 2017-11-13

- **New methods**
  - `create.openstack_project()`

1.5.10 0.12.0 - 2017-08-15

- **New methods**
  - `memstore.container.create()`

1.5.11 0.11.0 - 2017-05-22

- **New methods**
  - `partner.account.create()`
  - `partner.account.list()`
  - `partner.apikey.create()`
  - `partner.apikey.list()`
  - `partner.service.list()`

1.5.12 0.10.1 - 2017-03-13

- **Changes**
  - The '4x500gb' disk option has been replaced with '2x1000gb'. 500GB drives are no longer provided as an option in `create.monthly_fullserver()` for the UFS29 product spec.

1.5.13 0.10.0 - 2016-11-30

- **Changes**
  - The *sku* parameter of `create.monthly_miniserver()` now supports our current Miniserver SKUs.
  - `create.hourly_miniserver()` has now been deprecated and will not be supported in future API releases.
  - The following parameters in `create.monthly_miniserver()` and `create.monthly_fullserver()` are deprecated and only available for use with our classic Miniserver and dedicated server SKUs: *bandwidth_type*, *connection*, *disk_type*, *firewall*, *firewall_rule_group*, *monitoring_level*, *monthly_transfer_gb*, *os_bits*, *penetration_patrol*, *vulnscan*.

1.5.14 0.9.30 - 2016-11-17

- **Changes**
  - The "tar" *image_type* in `server.snapshot()` has been deprecated and only available for use with our classic Miniservers running a Linux operating system.

1.5.15 0.9.29 - 2016-07-11

- **Changes**
  - Optional partitioning parameter for `create.monthly_fullserver()` now uses 'two_volumes' instead of 'two_drives' for Windows servers with 4 disks.

1.5.16 0.9.28 - 2016-06-21

- **Changes**
  - `create.available()` now returns the full Data Zone name.
  - New optional *pub_ssh_key* parameter for `server.reimage_from_stock_image()`.

1.5.17 0.9.27 - 2016-01-14

- **Changes**
  - Added *SSL_URL* to the output of `memstore.container.cdn()`.

1.5.18 0.9.26 - 2015-11-23

- **Changes**
  - Support for data zones in `create.monthly_miniserver()`, `create.hourly_miniserver()`, `create.monthly_fullserver()`, `create.memstore()` and `create.available()`.

1.5.19 0.9.25 - 2015-09-17

- **New methods**
  - `loadbalancer.service.list()` to list a load balancer's services.
  - `loadbalancer.service.info()` to fetch information about a load balancer service.
  - `loadbalancer.service.add()` to add a service to a load balancer.
  - `loadbalancer.service.update()` to update a load balancer service's properties.
  - `loadbalancer.service.remove()` to remove a service from a load balancer.
  - `loadbalancer.server.info()` to fetch information about a server attached to a load balancer.
  - `loadbalancer.server.add()` to add a server to a load balancer service.
  - `loadbalancer.server.update()` to update the properties of a server attached to a load balancer.

  - `loadbalancer.server.remove()` to remove a server from a load balancer.

1.5.20 0.9.24 - 2015-08-18

- **Changes**
  - `server.slave_ns_list()`, `server.slave_ns_add()` and `server.slave_ns_delete()` methods now only work on non-cancelled servers.
  - `server.slave_ns_add()` returns an error when trying to add a domain name that is already slaved to the given server.
  - Domain names are now sorted in `server.slave_ns_list()`.

1.5.21 0.9.23 - 2015-07-23

- **Changes**
  - Optional *cors* parameter was added to `memstore.container.set_public_cdn()` method and included in `memstore.container.cdn()` output.

1.5.22 0.9.22 - 2015-04-23

- **New methods**
  - `vlan.add()` to add hosts to a vLAN.
  - `vlan.info()` to retrieve vLAN information.
  - `vlan.list()` to list informatio about hosts associated to a vLAN.
  - `vlan.remove()` to remove a host from a vLAN.
- **Changes**
  - vLANs are now listed in `server.info()` method.
  - Network zones information is now exposed in `service.info()`.
  - Added *add_to_next_bill* parameters to create methods.

1.5.23 0.9.21 - 2015-03-16

- **Changes**
  - New optional *network_zone* parameter for `create.monthly_fullserver()`.

1.5.24 0.9.20 - 2015-02-13

- **New method**
  - `firewalling.rule_group_status()` to retrieve the status of a rule group for a server.

1.5.25 0.9.19 - 2014-10-28

- **New method**
  - `server.move_ips()` to move public IP addresses between servers.

1.5.26 0.9.18 - 2014-10-17

- **New methods**
  - `server.monitoring.set_policy()` to enable/disable monitoring features
  - `server.monitoring.get_policy()` to get enabled/disabled monitoring features
- **Changes**
  - New optional *vlan* parameter for `create.hourly_miniserver()`
  - New optional *pub_ssh_key* parameter for *create* methods that allows installing a public SSH key during the setup process (Linux only).

1.5.27 0.9.17 - 2014-09-11

- **New method**
  - `server.reimage_from_stock_image()` for re-imaging Miniservers using a stock OS image.
- **Changes**
  - Fixed a typo in `apikey.info()` documentation.

1.5.28 0.9.16 - 2014-03-19

- Changes
  - The optional *vlan* parameter in `create.monthly_miniserver()` has changed. Now it can be used to provide the name of a vLAN product to put the server in when it is created. If the parameter is not provided, the server won't join any vLAN.

1.5.29 0.9.15 - 2014-02-14

- Changes
  - Support for `self_managed` firewalling level
  - New optional parameter `firewall_rule_group` for `create.hourly_miniserver()`, `create.monthly_miniserver()` and `create.monthly_fullserver()`

1.5.30 0.9.14 - 2014-01-27

- **Changes**
  - Added new *JSON-RPC* interface to the API.

1.5.31 0.9.13 - 2014-01-24

- **New methods**
  - `firewalling.rule_create()` for creating a private firewall rule
  - `firewalling.rule_delete()` for deleting a private firewall rule
  - `firewalling.rule_info()` for retrieving information for a firewall rule
  - `firewalling.rule_update()` for changing a firewall rule definition
  - `firewalling.rule_group_create()` for creating a private firewall rule group
  - `firewalling.rule_group_delete()` for deleting a private firewall rule group
  - `firewalling.rule_group_info()` for retrieving information for a firewall rule group
  - `firewalling.rule_group_list()` for retrieving information for multiple firewall rule groups
  - `firewalling.update()` to update the firewall rule group in use for a server ('managed' firewalling service only)
- **Changes**
  - Dictionary returned by `server.info()` now includes a `firewall_rule_group` key giving details of the firewall rule group applied to the server

1.5.32 0.9.12 - 2014-01-08

- **New methods**
  - `invoice.info()` to retrieve information about an invoice
  - `invoice.list()` to retrieve information about multiple invoices
  - `server.slave_ns_list()` to list all slaved domains for a server
  - `server.slave_ns_add()` to add a slaved domain to Memset's slave name servers
  - `server.slave_ns_delete()` to delete a slaved domain from Memset's slave name servers

1.5.33 0.9.11 - 2013-11-22

- **Changes**
  - The *record* parameter for `dns.zone_record_create()` is now optional, and will default to the empty string *''*. This allows creation of zone records for the domain itself, e.g. *example.com*.

1.5.34 0.9.10 - 2013-10-03

- **Changes**
  - Create methods now take the full name of the operating system (example: *debian_wheezy_64*). This is consistent with other api methods, for example `server.info()`. The *os_bits* parameter is now optional.
- **New methods**
  - `server.upgrade()` to upgrade Miniserver instances to a higher specification.

1.5.35 0.9.9 - 2013-05-22

- **Changes**
  - `create.monthly_miniserver()` and `create.monthly_fullserver()` *vulnscan* parameter type changed from Integer to String
- **New method parameters**
  - New optional *vulnscan* parameter for `create.hourly_miniserver()`
- **New methods**
  - `server.monitoring.rule_list()` to list Port Patrol monitoring rules for server
  - `server.monitoring.rule_info()` to retrieve details of a single monitoring rule
  - `server.monitoring.rule_create()` to create a monitoring rule
  - `server.monitoring.rule_delete()` to delete a monitoring rule
  - `server.monitoring.rule_update()` to update a monitoring rule

1.5.36 0.9.8 - 2013-05-03

- **Changes**
  - `create.monthly_miniserver()` and `create.monthly_fullserver()` *penetration_patrol* parameter type changed from Integer to String
  - `server.info()` now returns Penetration Patrol information
- **New method parameters**
  - `create.hourly_miniserver()` now accepts an optional *penetration_patrol* parameter
- **New methods**
  - `server.set_penetration_patrol_alert_level()`

1.5.37 0.9.7 - 2013-02-19

- **New methods**
  - `create.extra_bandwidth()` for purchasing extra bandwidth bank
- **New method parameters**
  - Added Penetration Patrol parameter to `create.monthly_miniserver()` and `create.monthly_fullserver()`
  - New network zone parameter for `create.monthly_miniserver()`, `create.hourly_miniserver()` and `create.memstore()`
  - New *notfound* parameter for `memstore.container.set_public_cdn()`
- **Changes**
  - `memstore.container.cdn()` returns an extra *notfound* dictionary key

1.5.38 0.9.6 - 2012-11-20

- **New methods**
  - `create.monthly_miniserver()` for provisioning Miniserver VM Virtual Servers on a monthly basis
  - `create.monthly_fullserver()` for provisioning Fully Dedicated Servers on a monthly basis

1.5.39 0.9.5 - 2012-09-20

- **Changes**
  - Don't apply container name restrictions on ACL methods

1.5.40 0.9.4 - 2012-07-12

- **New methods**
  - `server.snapshot_delete()` for deleting snapshots
- **Changes**
  - Documentation error fixed in `service.info()`
  - Corrected wrong references to `server.snapshot_list()`

1.5.41 0.9.3 - 2012-06-29

- **Changes**
  - Changed bytes field to Float in `memstore.usage()` to support big numbers

1.5.42 0.9.2 - 2012-06-08

- **Changes**
  - Support SSD as an option in `create.hourly_miniserver()`
  - Correct error return in `server.snapshot_list()`

1.5.43 0.9.1 - 2012-06-01

- **New methods**
  - `create.memstore()` for creating Memstore instances
  - `dns.reload()` to push DNS changes out
  - `memshell.info()` for info about your Memshell
- **Changes**
  - Document default flag in `payment_method.info()`
  - Clarify docs in `payment_method.remove()`

1.5.44 0.9.0 - 2012-04-27

- First public release

CHAPTER

TWO

EXAMPLES

2.1 Python Example

This is an example of how to use the Memset API with Python and the standard module xmlrpclib for XML-RPC client access.

Substitute **API_KEY_HEX** with a valid API key and it can be run with a Python 2 interpreter.

```
#!/usr/bin/env python
"""
    Memset API example with Python.
"""

uri = "https://API_KEY_HEX:@api.memset.com/v1/xmlrpc/"

from xmlrpclib import ServerProxy
from pprint import pprint

def main():
    s = ServerProxy(uri)

    # list available methods
    print s.system.listMethods()

    # display help for service.list method
    print s.system.methodHelp("service.list")

    # get the product list
    r = s.service.list()
    pprint(r)

    # get the server list
    r = s.server.list()
    pprint(r)

    # get server information (one parameter)
    r = s.server.info(dict(name="myserver1"))
    pprint(r)

    # set a new nickname for a product (two parameters)
    r = s.service.set_nickname(dict(name="myserver1", nickname="www"))
    pprint(r)

    # reboot a server
```

```
    r = s.server.reboot(dict(name="myserver1"))
    pprint(r)

if __name__ == "__main__":
    main()
```

2.2 Python JSON-RPC Example

This is an example of how to use the Memset API with Python and the 3rd party module jsonrpclib for JSON-RPC client access.

Substitute **API_KEY_HEX** with a valid API key and it can be run with a Python 2 interpreter.

```
#!/usr/bin/env python
"""
    Memset API example with Python and JSON-RPC

    http://www.jsonrpc.org/specification

    Install the JSON-RPC library from

    https://pypi.python.org/pypi/jsonrpclib
"""

uri = "https://API_KEY_HEX:@api.memset.com/v1/jsonrpc/"

from jsonrpclib import Server, MultiCall
from pprint import pprint

def main():
    s = Server(uri)

    # get the product list
    r = s.service.list()
    pprint(r)

    # get the server list
    r = s.server.list()
    pprint(r)

    # get server information (one parameter)
    r = s.server.info(name="myserver1")
    pprint(r)

    # set a new nickname for a product (two parameters)
    r = s.service.set_nickname(name="myserver1", nickname="www")
    pprint(r)

    # reboot a server
    r = s.server.reboot(name="myserver1")
    pprint(r)

    # batch mode
    batch = MultiCall(s)
    batch.internal.echo(string="Hello World")
    batch.internal.version()
```

```
    batch.internal.noop()
    batch.internal.echo(string="Goodbye!")
    # run the batch - this returns an iterator which gives the results
    # in order that they were added
    r = batch()
    pprint(list(r))

if __name__ == "__main__":
    main()
```

2.3 Python Firewalling Example

This is an example of how to use the Memset Firewalling API with Python and the standard module xmlrpclib for XML-RPC client access.

Substitute **API_KEY_HEX** with a valid API key and **SERVER_NAME** with a valid server name and it can be run with a Python 2 interpreter.

```
#!/usr/bin/env python
"""
    Memset firewalling API example with Python.
"""

uri = "https://API_KEY_HEX:@api.memset.com/v1/xmlrpc/"
server_name = 'SERVER_NAME'

from xmlrpclib import ServerProxy
from pprint import pprint

def main():
    s = ServerProxy(uri)

    # create a rule group with some rules
    rule_group_nickname = "test-rule-group"
    rules = [
        {'action': 'ACCEPT',
        'dest_ports': '80,8000,8080',
        'protocols': 'tcp',
        'ordering': 1,
        'comment': 'Extended HTTP traffic',
        },
        {'action': 'REJECT',
        'ip_version': 'ipv4',
        'source_ips': '222.111.111.10/24',
        'protocols': 'tcp,udp',
        'ordering': 2,
        'comment': 'Dossers',
        },
    ]
    r = s.firewalling.rule_group_create(dict(nickname=rule_group_nickname, notes="Standard firewall r
    rule_group_name = r['name']
    pprint(r)

    # Apply this firewall rule group
    r = s.firewalling.update(dict(name=server_name, rule_group_name=rule_group_name))
    pprint(r)
```

```
    # add another rule to the rule group
    rule = {'action': 'ACCEPT',
            'dest_ports': '9999',
            'protocols': 'udp',
            'ordering': 3,
            'comment': 'Video conferencing',
            }

    r = s.firewalling.rule_create(dict(rule_group_name=rule_group_name, **rule))
    rule_id = r['rule_id']
    pprint(r)

    # update this firewall rule
    r = s.firewalling.rule_update(dict(rule_id=rule_id, ip_version='ipv4', dest_ports='9000,9001,999
    pprint(r)

    # switch back to one of the default public rules
    r = s.firewalling.update(dict(name=server_name, rule_group_name="managed-linux"))
    pprint(r)

    # clean up
    r = s.firewalling.rule_group_delete(dict(rule_group_name=rule_group_name))
    pprint(r)

    # 'clone' the managed-linux public group
    r = s.firewalling.rule_group_info(dict(rule_group_name='managed-linux'))
    cloned_rules = []
    for rule_id, rule_def in r['rules'].items():
        del(rule_def['rule_group_name'])
        cloned_rules.append(rule_def)

    cloned_group_nickname = 'cloned-group'
    r = s.firewalling.rule_group_create(dict(nickname=cloned_group_nickname, notes='Test to clone a
    cloned_group_name = r['name']
    pprint(r)

    # clean up
    r = s.firewalling.rule_group_delete(dict(rule_group_name=cloned_group_name))
    pprint(r)

if __name__ == "__main__":
    main()
```

2.4 Python Partner Example

This is an example of how to use the Memset Firewalling API with Python and the standard module xmlrpclib for XML-RPC client access.

Once the partner API key has been entered, and the discount line filled in or removed, the code will run with Python 2 or Python 3.

```
#!/usr/bin/env python
"""
Use a partner API key to create an associated account, and then a
free cloud storage service on the new account.
"""
```

```
from pprint import pprint
from time import sleep
try:
    from xmlrpclib import ServerProxy
except ImportError:
    # Python3
    from xmlrpc.client import ServerProxy


def get_connection(api_key):
    """Return an XMLRPC proxy when given an API key"""
    return ServerProxy('https://%s:@api.memset.com/v1/xmlrpc/' % api_key)

# key which has access to partner only methods
partner_key = '<PUT PARTNER API KEY HERE>'

# Create a new account
partner_api = get_connection(partner_key)
result = partner_api.partner.account.create({
    'user_title': "Ms",
    'user_forename': "Jane",
    'user_surname': "Doe",
    'user_company': "ACME",
    'user_email': "jane.doe@example.com",
    'user_phone': "01111111111",
    'user_address': "10 Downing Street",
    'user_town': "Westminster",
    'user_statecounty': "London",
    'user_postcode': "SW1A 2AA",
    'user_country': "GB"})
pprint(result)
account_name = result['account']
account_key = result['api_key']['key']

# Call api methods on behalf of the newly generated account
# using the returned key.

# The partner account will have defined a list of methods
# it is able to call on behalf of the newly created account
# e.g. create.memstore, memstore.user.create, memstore.usage,
# job.status

account_api = get_connection(account_key)
result = account_api.create.memstore({
    'sku': 'CLOUDF5',
    'discount_code': '<PUT DISCOUNT CODE HERE>',  # this line is optional
    'dry_run': False})
job = result['job']
pprint(result)

# wait for the job to finish
print("Waiting for setup")
while job['status'] != 'DONE':
    sleep(10)
    job = account_api.job.status({'id': job['id']})
    pprint(job)
print("Done!")
service_name = result['service']
```

```
# create a new API key for the product, and create a method proxy for it
result = partner_api.partner.apikey.create({
    'account_name': account_name,
    'service': service_name,
    'methods': ['memstore.usage'],
    'comment': "Usage only"})
pprint(result)

service_api = get_connection(result['key'])

# make a new user for the memstore
pprint(service_api.memstore.user.create({'name': service_name, 'username': 'a_username', 'password':

# display the usage information about the new store
pprint(service_api.memstore.usage({'name': service_name}))

# create a new Memstore container
service_api.memstore.container.create({'name': service_name, 'container': 'a_container_name'})

# set an ACL on the newly created container granting read/write access to the previously created use
service_api.memstore.container.set_acl({'name': service_name, 'container': 'a_container_name', 'read_

# view the ACL of a container
pprint(service_api.memstore.container.acl({'name': service_name, 'container': 'a_container_name'}))
{'read_acl': ['a_username'], 'write_acl': ['a_username']}
```

2.5 Java Example

This is an example of how to use the Memset API with Java and Apache XML-RPC.

Substitute **API_KEY_HEX** with a valid API key.

```
/*
* Memset API example using Java and Apache XML-RPC.
*/

import static java.lang.System.out;
import java.net.*;

import org.apache.xmlrpc.client.XmlRpcClient;
import org.apache.xmlrpc.client.XmlRpcClientConfigImpl;
import org.apache.xmlrpc.XmlRpcException;

import java.util.HashMap;

class xmlrpctest {
    private static final String API_URL = "https://api.memset.com/v1/xmlrpc";

    public static void main(String[] args) {

        XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();

        try {
            config.setServerURL(new URL(API_URL));
        } catch (MalformedURLException ex) {
            out.println("Wrong API_URL");
```

```
            return;
        }

        config.setBasicUserName("API_KEY_HEX");

        XmlRpcClient client = new XmlRpcClient();
        client.setConfig(config);

        try {
            Object[] methods = (Object[]) client.execute("system.listMethods",
                                                         new Object[] {});

            out.println("Method list:");
            for(int i=0; i<methods.length; i++) {
                out.println(methods[i]);
            }

            // we use named parameters
            HashMap params = new HashMap<String, Object>();
            params.put("name", "myserver1");

            HashMap service_info = (HashMap) client.execute("service.info",
                                                            new Object[] { params });

            out.println("service_info:");
            out.println(service_info.toString());

            // if a reboot has already been initiated/requested, XmlRpcException will be raised
            // otherwise we get information about the created job
            HashMap result = (HashMap) client.execute("server.reboot",
                                                      new Object[] { params });

            out.println("server reboot:");
            out.println(result.toString());

        } catch (XmlRpcException ex) {
            out.println(ex);
            return;
        }
    }
}
```

2.6 C# Example

This is an example of how to use the Memset API with C# and XML-RPC.NET.

Please note that float point numbers are represented as **double** in C#.

Substitute **API_KEY_HEX** with a valid API key.

```
/*
* Memset API example with C# and XML-RPC.NET library.
*
* Tested wit Mono:
*   gmcs -r:CookComputing.XmlRpcV2.dll xmlrpctest.c
*   mono xmlrpctest.exe
*/
```

```
using System;
using CookComputing.XmlRpc;

// structure returned by service.info
public struct ProductInfo
{
    public string name;
    public string status;
    public string nickname;
    public DateTime start_date;
    public DateTime expiry_date;
    public double renewal_price_amount;
    public string renewal_price_currency;
    public double renewal_price_vat;
    public string product_type;
}

// structure defining a job status
public struct JobStatus
{
    public string id;
    public string type;
    public string status;
    public string product; // optional
    public bool finished;
    public bool error;
}

// Memset API XML-RPC interface (only some examples)
[XmlRpcUrl("https://api.memset.com/v1/xmlrpc/")]
public interface IMemsetAPI : IXmlRpcProxy
{
    [XmlRpcMethod("system.listMethods")]
    string[] listMethods();

    // all methods use named parameters, so StructParams is required
    [XmlRpcMethod("service.info", StructParams = true)]
    ProductInfo product_info(string name);

    [XmlRpcMethod("server.reboot", StructParams = true)]
    JobStatus server_reboot(string name);

    // ...
}

class xmlrpctest
{
    static void Main(string[] args)
    {
        try
        {
            IMemsetAPI proxy = XmlRpcProxyGen.Create<IMemsetAPI>();
            proxy.Credentials = new System.Net.NetworkCredential("API_KEY_HEX", "*");

            string[] methods = proxy.listMethods();
            Console.WriteLine("Current method list:");
            foreach (string method in methods)
            {
```

```
                Console.WriteLine(method);
            }

            ProductInfo pi = proxy.product_info("myserver1");
            Console.WriteLine("Product info: {0} status is {1}", pi.name, pi.status);

            Console.WriteLine("Trying to reboot the server...");
            JobStatus js = proxy.server_reboot("myserver1");
            Console.WriteLine("Error {0}: id {1}, status {2} ({3} for {4})",
                              js.error, js.id, js.status, js.type, js.product);
        }
        catch (XmlRpcFaultException fex)
        {
            Console.WriteLine("XMLRPC FAULT: {0}", fex.FaultString);
        }
        catch (Exception ex)
        {
            Console.WriteLine("EXCEPTION: {0}", ex.Message);
        }
    }
}
```

2.7 Ruby Example

This is an example of how to use the Memset API with Ruby and the XMLRPC library.

Substitute **API_KEY_HEX** with a valid API key.

```
#!/usr/bin/env ruby
#
# Memset API example with Ruby.
#
# Requires XMLRPC library.
#

require 'xmlrpc/client'
require 'pp'

API_URL = "https://API_KEY_HEX:@api.memset.com/v1/xmlrpc"

server = XMLRPC::Client.new2(API_URL)

result = server.call("system.listMethods")
pp result

# methods must use named parameters
result = server.call("service.info", { :name => "myserver1" })
pp result

begin
        result = server.call("service.info", { :name => "doesnotexist" })
rescue XMLRPC::FaultException => ex
        print "Remote call failed (method doesnotexist): " + ex + "\n"
end

result = server.call("server.reboot", { :name => "myserver1" })
```

```
pp result
```

2.8 PHP 5 Example

This is an example of how to use the Memset API with PHP 5 and the standard modules **xmlrpc** and **curl**.

Substitute **API_KEY_HEX** with a valid API key and it can be run with *php-cli*.

```
<?php
/*
 * Memset API example with PHP5.
 *
 * Requires PHP modules: xmlrpc and curl.
 */

// the API URL includes a valid API key
define('API_URL', 'https://API_KEY_HEX:@api.memset.com/v1/xmlrpc/');

// set to TRUE to get extra CURL info
define('DEBUG', FALSE);

// helper class, wraps xmlrpc and curl calls
class xmlrpc_cli {
        private $url;
        private $curl;

        function __construct($url) {
                $this->url = $url;
                $this->curl = curl_init();
        }

        function call($method, $params) {
                if ( $this->url == FALSE ) {
                        throw new Exception('Connection already closed');
                }

                $data = xmlrpc_encode_request($method, $params);
                if ( DEBUG ) {
                        curl_setopt($this->curl, CURLOPT_VERBOSE, TRUE);
                        curl_setopt($this->curl, CURLOPT_SSL_VERIFYPEER, FALSE);
                }
                curl_setopt($this->curl, CURLOPT_URL, $this->url);
                curl_setopt($this->curl, CURLOPT_RETURNTRANSFER, TRUE);
                curl_setopt($this->curl, CURLOPT_POST, 1);
                curl_setopt($this->curl, CURLOPT_POSTFIELDS, $data);
                $result = curl_exec($this->curl);
                $errno = curl_errno($this->curl);
                if ( $errno || $result == FALSE ) {
                        throw new Exception("Method call failed:" .$result, $errno);
                }
                return xmlrpc_decode($result);
        }
        function close() {
                curl_close($this->curl);
                $this->url = FALSE;
        }
```

```
}

// MAIN

$client = new xmlrpc_cli(API_URL);

// get method list
$methods = $client->call('system.listMethods', array());
echo "Available methods:\n";
print_r($methods);

echo "The request failed:\n";
try {
        // API call fail will return a fault code
        $fail = $client->call('notfound', array());
        print_r($fail);
} catch (Exception $e)  {
        // the helper class will throw an exception when there's a
        // problem in the transport layer (HTTPS, ie. bad URL)
        print_r($e);
}

// example of method call with one parameter
$prod_info = $client->call('service.info', array('name'=>'myserver'));
echo "Product info:\n";
print_r($prod_info);

$client->close();
```

2.9 node.js Example

This is an example of how to use the Memset API with node.js and node-xmlrpc module.

Substitute **API_KEY_HEX** with a valid API key.

```
//
// Memset API example using node.js and node-xmlrpc.
//

var xmlrpc = require('xmlrpc');

var cliOps = {
                host: 'api.memset.com',
                port: 443,
                path: '/v1/xmlrpc',
                basic_auth: {
                    user: 'API_KEY_HEX',
                    pass: '',
                }
    };

var client = xmlrpc.createSecureClient(cliOps);

client.methodCall('system.listMethods', null, function(error, value) {
    if(error) {
        console.log("Error: " + error);
```

```
        return;
    }
    console.log('Method list: ' + value);
});

client.methodCall('service.info', [ { name: 'myserver1', } ], function(error, value) {
    if(error) {
        console.log("Error: " + error);
        return;
    }
    console.log('service.info: ');
    for(prop in value) {
        console.log(prop + " = " + value[prop]);
    }
});

client.methodCall('server.reboot', [ { name: 'myserver1', } ], function(error, value) {
    if(error) {
        console.log("Error: " + error);
        return;
    }
    console.log('service.reboot result (new job): ');
    for(prop in value) {
        console.log(prop + " = " + value[prop]);
    }
});
```

2.10 Perl 5 Example

This is an example of how to use the Memset API with Perl 5 and XML::RPC::Client.

Substitute **API_KEY_HEX** with a valid API key.

```
#!/usr/bin/env perl
#
# Memset API example with Perl 5.
#

require RPC::XML;
require RPC::XML::Client;

use strict;
use warnings;

my $API_URL = 'https://API_KEY_HEX:@api.memset.com/v1/xmlrpc';

my $c = RPC::XML::Client->new($API_URL);

my $r;

# method invocation, no parameters
$r = $c->send_request('system.listMethods');
die("Error: $r") unless ref $r;

print "Method list:\n" .join("\n", @{$r->value}) ."\n\n";
```

```
# method invocation, named parameters
$r = $c->send_request('service.info', { name=>'myserver1' });
die("Error: $r") unless ref $r;

print "service.info:\n";

foreach (keys %{$r->value}) {
        print $_ ." = " .$r->value->{$_} ."\n";
}
```

2.11 Go Example

This is an example of how to use the Memset API with Go (golang) and the standard library JSON package.

Substitute **API_KEY_HEX** with a valid API key.

```
// Memset API example with Go
//
// This was written with Go 1 and only uses standard library components
//
// It uses the JSON interface to the Memset API and it uses the
// generic JSON tools in Go.  This means that it is necessary to use
// type assertions to read data from the API.  See below for some
// examples.
//
// The Rpc* functions below can be used as helper functions to make
// the dynamically typed API more compatible with a statically typed
// language like Go

package main

import (
        "encoding/json"
        "fmt"
        "io/ioutil"
        "log"
        "net/http"
        "net/url"
        "time"
)

var uri = "https://API_KEY_HEX:@api.memset.com/v1/json/"

type Param map[string]interface{}

// An RPC function for the Memset API
// Call with a method name and a Param map returns an interface{}
// which is most likely a map[string]interface{} of results
func Rpc(method string, params Param) (interface{}, error) {
        encoded_params, err := json.Marshal(params)
        if err != nil {
                return nil, err
        }
        resp, err := http.PostForm(uri+method, url.Values{"parameters": {string(encoded_params)}})
        if err != nil {
                return nil, err
```

```
        }
        defer resp.Body.Close()
        if resp.StatusCode != http.StatusOK {
                return nil, fmt.Errorf("Bad response %s", resp.Status)
        }
        body, err := ioutil.ReadAll(resp.Body)
        if err != nil {
                return nil, err
        }
        var result interface{}
        err = json.Unmarshal(body, &result)
        return result, err
}

// As Rpc but will log.Fatal any error messages
func RpcChecked(method string, params Param) interface{} {
        result, err := Rpc(method, params)
        if err != nil {
                log.Fatal(err)
        }
        return result
}

// As RpcChecked but returns the result as a map
func RpcCheckedMap(method string, params Param) Param {
        return RpcChecked(method, params).(map[string]interface{})
}

// As RpcChecked but returns the result as an array of maps
func RpcCheckedMapArray(method string, params Param) []Param {
        result := RpcChecked(method, params).([]interface{})
        new_result := make([]Param, len(result))
        for i := range result {
                new_result[i] = result[i].(map[string]interface{})
        }
        return new_result
}

func main() {
        fmt.Println("get the service list")
        r := RpcCheckedMapArray("service.list", Param{})
        fmt.Println(r)
        fmt.Println("First service", r[0])
        fmt.Println("First service nickname", r[0]["nickname"])

        fmt.Println("get the server list")
        r = RpcCheckedMapArray("server.list", Param{})
        fmt.Println(r)
        fmt.Println("First server", r[0])
        fmt.Println("First server nickname", r[0]["nickname"].(string))

        fmt.Println("get server information (one parameter)")
        result := RpcCheckedMap("server.info", Param{"name": "myserver1"})
        for key, value := range result {
                fmt.Printf("  %s = %v\n", key, value)
        }
        // use type assertions to get values out and use them
        nickname := result["nickname"].(string)
```

```
        fmt.Printf("Nickname is %s\n", nickname)
        monitor := result["monitor"].(bool)
        if monitor {
                fmt.Println("Monitoring is On")
        } else {
                fmt.Println("Monitoring is Off")
        }

        fmt.Println("set a new nickname for a product (two parameters)")
        result = RpcCheckedMap("service.set_nickname", Param{"name": "myserver1", "nickname": "www"})
        fmt.Println(result)

        fmt.Println("reboot a server")
        job := RpcCheckedMap("server.reboot", Param{"name": "myserver1"})
        for !job["finished"].(bool) {
                fmt.Println("Waiting for reboot to finish...")
                time.Sleep(5 * time.Second)
                job = RpcCheckedMap("job.status", Param{"id": job["id"]})
        }
        if !job["error"].(bool) {
                fmt.Println("Reboot OK")
        } else {
                fmt.Println("Reboot FAILED")
        }
}
```

2.12 cURL Example

The API can be used easily with cURL or any other command line based HTTP client thanks to the REST interface.

The request URL is built using the following steps:

- The **base URL** for the API: `https://api.memset.com/v1/json/`.
- The **method name**: i.e. service.info.
- Optionally a `name` **parameter**: ie. myserver1.

According to the previous example, our request URL for the `service.info()` method (that requires just one `name` parameter) would be:

```
https://api.memset.com/v1/json/service.info/myserver/
```

The API key required to authenticate can be provided in two ways:

- Using the **api_key** GET parameter.
- Using basic HTTP authentication mechanism with the api_key as the username and the password is ignored.

Following our previous example (substitute **API_KEY_HEX** with a valid API key), the cURL invocation including the API key in a GET parameter would be:

```
curl 'https://api.memset.com/v1/json/service.info/myserver1/?api_key=API_KEY_HEX'
```

Including the API key as part of the basic HTTP authentication mechanism, the cURL invocation would be:

```
curl --user API_KEY_HEX:x https://api.memset.com/v1/json/service.info/myserver1/
```

Responses will be delivered as an HTTP document with JSON encoded data with content type `application/json`. An example of a valid response would be:

```
{
    "status": "LIVE",
    "renewal_price_currency": "GBP",
    "type": "miniserver",
    "name": "myserver1",
    "expiry_date": "2012-12-01",
    "renewal_price_amount": 21.60,
    "nickname": "www",
    "start_date": "2010-12-01",
    "renewal_price_vat": 4.34
}
```

Invalid responses will cause one of the HTTP status codes listed in *Errors* to be returned along with a JSON encoded dictionary with a bit more information in.

For example this request for a nonexistent server:

```
curl --user API_KEY_HEX:x https://api.memset.com/v1/json/service.info/BADSERVERNAME/
```

Returns this JSON dictionary with a `404` HTTP status code:

```
{
    "error_type": "ApiErrorDoesNotExist",
    "error_code": 3,
    "error": "Couldn't find service with name 'BADSERVERNAME'"
}
```

To pass more complicated parameters it is necessary to JSON encode them in the `parameters` argument:

```
curl --data-urlencode 'parameters={"name":"myserver","api_key":"API_HEX_KEY"}' https://api.memset.cor
```

This supplies a JSON encoded dictionary with a `name` parameter and an `api_key` parameter. The parameters argument can be used to pass all the parameters and is the only way to pass complex data structures.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *search*

INDEX

E

F

G

I

J

L

M

N

O

P

R

S

V

W

X