LevelBlue
Secure What's Next.

MAILMARSHAL

API Handler Listing 12.0.0

Table of Contents

About This Document

This document provides a formatted listing of the API commands as present in MailMarshal 12.0 (as well as later versions unless superseded). There are no changes from version 11.3.0.

For authentication options, see the Authentication section below.

For more information about the API and some examples of usage, see Knowledge Base article Q20867. The latest version of this document is also linked from the article.

Caution: API commands may change with product version. You should confirm the available functions from your installed version of MailMarshal using the API List call described below. Calling applications should always confirm the API version before making other calls.

Authentication

In current MailMarshal versions, the preferred access control method for the API is token based authentication of MailMarshal authorized users (Management Console logins).

- Admin and Superuser logins have full permission in the API.
- Helpdesk logins generally can perform email management tasks and view statistics. Helpdesk logins cannot update configuration of servers or services.

Important: Token based access control only works for MailMarshal authorized users, not Windows credentials. Even if Management Console authentication is set to NTLM, you can and must use MailMarshal users for token based API access.

- You can also use Windows credentials to access the API with Basic Authentication. Members of the Administrators group on the Array Manager are granted this access. This method is deprecated, less secure, and does not allow granular control of access.

Each request to the API must include an Authorization header containing a Bearer token.

For example:

```
Authorization: Bearer eyJhbGciOiJIUzI1N…FsSwlX2U
```

The token grants access to a MailMarshal authorized user.

To get the bearer token, sent a POST request to the MailMarshal Configuration Service site as follows:

`https://[servername]:19007/token` (note no trailing /)

Post data: `?grant_type=password&username=<username>&password=<sha256 hash of password>`

For testing purposes, you can get the hash of a password with a free online or GUI tool.

Tip: In Postman, “post data field” items are entered as individual keys in the x-www-form-urlencoded body data (not the query parameters).

General notes

- In the URLs below, ^ indicates a path within the MailMarshal API website. From the MailMarshal Array Manager you can access this website at `https://localhost:19006/seg/api/`
- Variable substitutions within a path are shown as regular expressions.
- Names in `< >` indicate the type of item expected.
    - `(?<serverid>[0-9]+)` indicates the expected value is an integer server ID.
    - `(?<domain>.+)` indicates the expected value is a text domain name.
    - For example, to get Receiver details for node 1: `https://localhost:19006/seg/api/console/overview/1/receiver/`
    - To find the valid values, take the output of another call or see user interfaces.
- Query Parameters are sent as a http querystring. The parameter names are case sensitive.
- Variable data in querystrings MUST be URL Encoded. For example, a semi-colon ; must be represented as `%3B`
- POST data is generally a JSON formatted body. GET a sample object to see the correct format.
- GUID strings are generally required to be enclosed in curly brackets { }
- Times (integers) are Unix epoch timestamps. The minimum value is 0 and the maximum value is 2147483647. For testing, you can convert times using many free online services. For production, cast the date/time to this format programmatically.
- The parameter processMessageKey is a JSON object with the following items (all required): blockNumber, edition, folderId, messageName, recipient, serverId, timeLogged. These values can be obtained from the result of a Find Message call.
- The parameters Offset and Bytes (where present) allow data such as a MML or unpacked component to be retrieved in chunks.
- The parameter Flags for message handling functions passes access permissions in an integer interpreted as a bitset. The value 32767 signifies full access.

1 API List

This command allows you to get the current version of API commands from the server. Use Firefox or Postman to view the information in a clearly printed format.

Get API list

Method: GET
URL:`^debug/handlers/`

Example: `https://localhost:19006/seg/api/debug/handlers`

2 ACLs

In this version of MailMarshal, access to the REST API features uses Management Console logins. The ACL section of the API is no longer relevant.

3 Autobackup

Return list of automatic config backups

Method: GET
URL:`^services/config/autobackups/$`

Return a specific automatic config backup

Method: PUT
URL:`^services/config/autobackups/restore/$`

- *Put data:* name(str), includeDkim(bool), dkimPassword(string))

Run auto backup now

Method: PUT
URL:`^services/config/autobackups/backup/$`

- *Put data:* includeDkim (bool), dkimPassword(string, optional - defaults to configured password)

4 Azure Information Protection

Test AzureIP client

Method: POST
URL:`^services/azureip/testclient/(?<serverid>[0-9]+)/$`

Test AzureIP connection

Method: POST
URL:`^services/aip/test_connection/(?<serverid>[0-9]+)/$`

- *Post data fields:* bposTenantId(string), appPrincipalId(string), base64Key(string), extranetUrl(string), intranetUrl(string)

Get AzureIP error message

Method: GET
URL:`^ services/azureip/error_message/(?<serverid>[0-9]+)/$`

5 Connectors

These commands allow you to work with MailMarshal Connectors.

Get list of connectors

Method: GET
URL:`^connectors/$`

Update an existing connector

- *Put data:* guid(str), type(AD: or LDAP:), startTime (time_t), refreshDelay(int), name(str), description(str),
- *AD specific post data:* adLogonUser(str), adLogonPassword(str), adGcHost(str),
- *LDAP specific post data:* ldapServer(str), ldapPort(int), ldapVersion(int), ldapLogonUser(str), ldapLogonPassword(str), ldapLogonDomain(str) useSsl(bool), useNtlm(bool), timeout(int), groupClasses(str), nameAttribute(str), descriptionAttribute(str), userClasses(str), searchMembers(str), memberAttribute(str), searchRoot(str), addressFilter(array of objects)
    - *addressFilter:* mailAttribute(str), match(str), value(str)

Method: PUT
URL:`^connectors/$`

Create a new connector

- *Post data:* guid(str), type(AD: or LDAP:), startTime (time_t), refreshDelay(int), name(str), description(str),
- *AD specific post data:* adLogonUser(str), adLogonPassword(str), adGcHost(str),
- *LDAP specific post data:* ldapServer(str), ldapPort(int), ldapVersion(int), ldapLogonUser(str), ldapLogonPassword(str), ldapLogonDomain(str) useSsl(bool), useNtlm(bool), timeout(int), groupClasses(str), nameAttribute(str), descriptionAttribute(str), userClasses(str), searchMembers(str), memberAttribute(str), searchRoot(str), addressFilter(array of objects)
    - *addressFilter:* mailAttribute(str), match(str), value(str)

Method: POST
URL:`^connectors/$`

Get connector information

Method: GET
URL:`^connectors/(?<connectorguid>[0-9A-Fa-f]{8}[-]?([0-9A-Fa-f]{4}[-]?){3}[0-9A-Fa-f]{12})/$`

Delete existing connector

Method: DELETE
URL:`^connectors/(?<connectorguid>[0-9A-Fa-f]{8}[-]?([0-9A-Fa-f]{4}[-]?){3}[0-9A-Fa-f]{12})/$`

Get containers of a domain

- *Query parameters:* domain (str, dn - empty for all)

Method: GET
URL:`^connectors/(?<connectorguid>[0-9A-Fa-f]{8}[-]?([0-9A-Fa-f]{4}[-]?){3}[0-9A-Fa-f]{12})/containers/$`

Get list of groups for a connector that have been imported into the GroupManager

Method: GET
URL:`^connectors/(?<connectorguid>[0-9A-Fa-f]{8}[-]?([0-9A-Fa-f]{4}[-]?){3}[0-9A-Fa-f]{12})/importedgroups/$`

Get list of search groups for a container

- *Query parameters:* searchRoot (str), recursive (bool), nameFilter (str), descriptionFilter (str)

Method: GET
URL:`^connectors/(?<connectorguid>[0-9A-Fa-f]{8}[-]?([0-9A-Fa-f]{4}[-]?){3}[0-9A-Fa-f]{12})/searchgroups/$`

Get list of search roots

Method: GET
URL:`^connectors/(?<connectorguid>[0-9A-Fa-f]{8}[-]?([0-9A-Fa-f]{4}[-]?){3}[0-9A-Fa-f]{12})/searchroots/$`

Get list of AD Domains

Method: GET
URL:`^connectors/addomains/$`

Test connector configuration

- *Post data:* guid(str), type(AD: or LDAP:), startTime (time_t), refreshDelay(int), name(str), description(str),
- *AD specific post data:* adLogonUser(str), adLogonPassword(str), adGcHost(str),
- *LDAP specific post data:* ldapServer(str), ldapPort(int), ldapVersion(int), ldapLogonUser(str), ldapLogonPassword(str), ldapLogonDomain(str) useSsl(bool), useNtlm(bool), timeout(int), groupClasses(str), nameAttribute(str), descriptionAttribute(str), userClasses(str), searchMembers(str), memberAttribute(str), searchRoot(str), addressFilter(array of objects)
    - *addressFilter:* mailAttribute(str), match(str), value(str)

  Method: POST
  URL:`^connectors/test/$`

6 Console

These commands allow you to work with the Console settings, statistics, and security, as well as retrying or killing a message.

Get a bitmap of console security settings *(not relevant in this version of MailMarshal)*

Method: GET
URL:`^console/accessmap/`

Get list of current alerts.

- *Query parameters:* activeonly(bool)

  Method: GET
  URL:`^console/alerts/`

Get array statistics.

- *Query parameters:* fromtime(int64), totime(int64)

  Method: GET
  URL:`^console/array/stats/`

Get list of timeseries data available (Console dashboard).

Method: GET
URL:`^console/array/timeseries/list/`

Get array timeseries data.

- *Query parameters:* fromtime(int64), totime(int64), statistic(string), configid(int), granularity(int, optional:0 - Daily;1 - 30 minutes)

Method: GET
URL:`^console/array/timeseries/data/`

Get list of event logs.

- *Query parameters:* servers(string), sources(string), logtype(integer), eventtype(integer), starttime(int64), endtime(int64), allsources(bool).
  - *logtype:* Application(1), Security(2), System(4), UpdateManager(8), All(15).
  - *eventtype:* Information(1), Warning(2), Error(4), Success(8), Failure(16), All(31)
  - *servers:* Semicolon separated list of node id numbers (blank = Array Manager)
  - *sources:* Semicolon separated list of service names.
  - *allsources:* If sources is blank and allsources=1, returns data for all services
  - Semicolons and service names must be URL Encoded (semicolon = `%3B`)

Method: GET
URL:`^console/event/logs/`

Get list of event sources

Method: GET
URL:`^console/event/sources/`

Get list of hold queue details for server

Method: GET
URL:`^console/holdqueue/details/(?<serverid>[0-9]+)/`

Retry the hold queue for server.

- *Post data fields:* ruleName(string)

Method: POST
URL:`^console/holdqueue/retry/(?<serverid>[0-9]+)/`

Get whether it is an appliance *(not used)*

Method: GET
URL:`^console/isappliance/`

Get whether mailbatching is enabled

Method: GET
URL:`^console/mailbatchingenabled/`

Get McAfee update information for server

Method: GET
URL:`^console/mcafeeupdateinfo/(?<serverid>[0-9]+)/`

Get server overview

Method: GET
URL:`^console/overview/(?<serverid>[0-9]+)/$`

Get engine details

Method: GET
URL:`^console/overview/(?<serverid>[0-9]+)/engine/`

Get receiver details

Method: GET
URL:`^console/overview/(?<serverid>[0-9]+)/receiver/`

Get sender details

Method: GET
URL:`^console/overview/(?<serverid>[0-9]+)/sender/`

Get whether quarantine message count on Today page is enabled

Method: GET
URL:`^console/quarantinecountsenabled/`

Kill message currently queued/being delivered by Sender.

- *Post data fields:* messageName(string), bNotify(bool)

Method: POST
URL:`^console/sender/message/kill/(?<serverid>[0-9]+)/`

Kill messages currently queued/being delivered by Sender.

- *Post data fields:* sender(string, regular expression), recipient(string, regular expression), subject(string, regular expression), speCustomerId(int) – at least one of these is required; bNotify(bool, optional)

Method: POST
URL:`^console/sender/message/killex/(?<serverid>[0-9]+)/`

Get sender route details for server.

- *Query parameters:* routename(string, optional), speCustomerId(int, optional) - at least one of routename or speCustomerId is required.

Method: GET
URL:`^console/sender/route/details/(?<serverid>[0-9]+)/`

Get sender route path for server.

- *Query parameters:* routename(string)

Method: GET
URL:`^console/sender/route/path/(?<serverid>[0-9]+)/`

Retry a route now in Sender.

- *Post data fields:* routeName(string)

Method: POST
URL:`^console/sender/route/processnow/(?<serverid>[0-9]+)/`

Get sender routes list for server.

- *Query parameters:* speCustomerId(int, optional)

Method: GET
URL:`^console/sender/routes/(?<serverid>[0-9]+)/`

Get list of disks information for server

Method: GET
URL:`^console/server/disks/(?<serverid>[0-9]+)/`

7 DMARC

Return timeseries graph of dmarc reports

- *Query parameters:* domain (str), fromDate (date), endDate (date), flags (int), renderLegend (bool), chartWidth (int), chartHeight(int)

Method: GET
URL:`^dmarc/reports/byday/graph/$`

Get raw DMARC reports

- *Query parameters:* fromDate (date), endDate (date), ipAddr (str), smtpDomain (str), reporterDomain (str), flags (int)

Method: GET
URL:`^dmarc/reports/raw/$`

Return top DMARC domains graph

- *Query parameters:* fromDate (date), endDate (date), chartWidth (int), chartHeight(int), renderLegend (bool), logScale (false), flags (int)

Method: GET
URL:`^dmarc/reports/topdomains/$`

Return top DMARC reporters graph

- *Query parameters:* fromDate (date), endDate (date), chartWidth (int), chartHeight(int), renderLegend (bool), logScale (false), flags (int)

  Method: GET
  URL:`^dmarc/reports/topreporters/$`

Return list of domains that we have received reports for

Method: GET
URL:`^dmarc/smtpdomains/$`

8 File Transfer

Create new file transfer session

Method: POST
URL:`^filetransfer/$`

Read file data

- *Query parameters:* bytes(int), offset(int: optional)

  Method: GET
  URL:`^filetransfer/(?<context>[0-9]+)/$`

Destroy a file transfer session

Method: DELETE
URL:`^filetransfer/(?<context>[0-9]+)/$`

Write file data

Method: PATCH
URL:`^filetransfer/(?<context>[0-9]+)/$`

9 Geolite (IP to location resolution)

Resolve the ipaddress to its geo location

Method: GET
URL:`^geolite/(?<ipaddr>.*)/$`

10 Header Matching and Rewriting Test

Load and validate a Header Rewrite object

- *Post data fields:* name(string), parseMethod(int), fields(stringarray), preFilterExp(string, optional), searchExp(string), logChanges(bool, optional), enableChanges(boo,l optional), matchCase(bool, optional), substMethod(int), substExpr(string), mapFile(string, required if substMethod = 1), testSource(string)
    - *parseMethod:* 0 – entire line, 1 – email addresses, 2 – user, 3 – domain, 4 – body
    - *substMethod:* 0 – direct substitution, 1 – map file, 2 – delete, 3 – insert

Method: POST
URL:`^headerrewrite/test/$`

Load and validate a Header Match object

- *Post data fields:* name(string), parseMethod(int), fields(stringarray), preFilterExp(string, optional), searchExp(string), matchCase(bool, optional), testSource(string)
    - *parseMethod:* 0 – entire line, 1 – email addresses, 2 – user, 3 – domain, 4 – body

Method: POST
URL:`^headerrewrite/match/$`

11 Groups

Get list of groups

- *Query parameters:* toplevelonly(bool, optional)

Method: GET
URL:`^groups/$`

Get group details

Method: GET
URL:`^groups/(?<groupid>[0-9]+)/$`

Update group from connector

Method: GET
URL:`^groups/(?<groupid>[0-9]+)/connectorupdate/$`

Add membership delta information to group

- *Patch data fields:* add(array of strings), remove(array of strings)

Method: PATCH
URL:`^groups/(?<groupid>[0-9]+)/delta/$`

Find which groups match a user

- *Post data fields:* userEmail(string)

Method: POST
URL:`^groups/(?<groupid>[0-9]+)/find/$`

Get sub-group members of a group

Method: GET
URL:`^groups/(?<groupid>[0-9]+)/groups/$`

Locate user in groups

- *Post data fields:* userEmail(string, may include wildcard)

Method: POST
URL:`^groups/(?<groupid>[0-9]+)/locate/$`

Get group members

Method: GET
URL:`^groups/(?<groupid>[0-9]+)/members/$`

Import a group from a connector

- *Post data fields:* connectorGuid(string), dn(string)
- Method: POST
  URL:`^groups/import/$`

Create a new user-maintained group

- *Post data fields:* friendlyName(string), description(string), guid(string, optional)

Method: POST
URL:`^groups/usermaintained/$`

Update group

- *Put data fields:* friendlyName(string), description(string), pruningDays(int), pruningCount(int)

Method: PUT
URL:`^groups/usermaintained/(?<groupid>[0-9]+)/$`

Bulk update group

Method: PUT
URL:`^groups/usermaintained/bulkupdate/$`

- *Put data fields:* guid(string; if missing, a new group is created), friendlyName(string), description(string), members(array of objects)
  - memberType(string), value(string)
  - *Valid member types:*
    - guid (value must be the GUID of an existing group)
    - email (value must be an email or wildcard pattern)

Bulk clean groups

Caution: This function deletes all user maintained groups **except** the ones named.

- *Delete data fields:* guids(array of strings)
  - *Guids of groups to be **retained***

Method: DELETE
URL:`^groups/usermaintained/bulkclean/$`

Duplicate group

Method: POST
URL:`^groups/usermaintained/(?<groupid>[0-9]+)/$`

Delete group

Method: DELETE
URL:`^groups/usermaintained/(?<groupid>[0-9]+)/$`

Bulk set group members

- *Put data fields:* array of strings

Method: PUT
URL:`^groups/usermaintained/(?<groupid>[0-9]+)/members/$`

Add member to a group

- *Post data fields:* memberType(int), memberGroupId(int), memberData(string), touchDate(int)
  - *GroupMemberType:* 0 - group, 1 - email, 2 - wildcard

Method: POST
URL:`^groups/usermaintained/(?<groupid>[0-9]+)/members/$`

Remove member from a group

- *Delete data fields:* memberType(int), memberGroupdId(int), memberData(string)

- *GroupMemberType:* 0 - group, 1 - email, 2 - wildcard

Method: DELETE
URL:`^groups/usermaintained/(?<groupid>[0-9]+)/members/$`

Get group USN

- *Query parameters:* fqn(string)

Method: GET
URL:`^groups/usn/`

12 IP Groups

Get list of IP groups

Method: GET
URL:`^ipgroups/$`

Create a new IP group

- *Post data fields:* friendlyName(string), description(string), guid(string, optional)

Method: POST
URL:`^ipgroups/$`

Get IP group details

Method: GET
URL:`^ipgroups/(?<groupid>[0-9]+)/$`

Update an IP group

- *Put data fields:* friendlyName(string, optional), description(string, optional)

Method: PUT
URL:`^ipgroups/(?<groupid>[0-9]+)/$`

Delete an IP group

Method: DELETE
URL:`^ipgroups/(?<groupid>[0-9]+)/$`

Get child groups of an IP group

Method: GET
URL:`^ipgroups/(?<groupid>[0-9]+)/childgroups/$`

Add membership delta information to an IP group

- *Patch data fields:* add(array of objects with fields: ipAddressStart(string), cidr(int), ipAddressEnd(string, required only when cidr is 0), name(string), description(string)), remove(array of objects with fields: ipAddressStart(string), cidr(int), ipAddressEnd(string))

  Method: PATCH
  URL:`^ipgroups/(?<groupid>[0-9]+)/delta/$`

Duplicate an IP group

Method: POST
URL:`^ipgroups/(?<groupid>[0-9]+)/duplicate/$`

Find which IP group members match an IP address

- *Post data fields:* ip(string)

  Method: POST
  URL:`^ipgroups/(?<groupid>[0-9]+)/find/$`

Update an IP group member

- *Put data fields:* memberType(GroupMemberType, optional), oldName(string), newName(string), description(string, optional)
  - *GroupMemberType:* 3 - IPCIDR, 4 - IPRange

  Method: PUT
  URL:`^ipgroups/(?<groupid>[0-9]+)/member/$`

Add a member to an IP group

- *Post data fields:* To insert Group: memberType(GroupMemberType), memberGroupdId(int); To add IPCIDR or IPRange: memberType(GroupMemberType), ipAddressStart(string), cidr(int), ipAddressEnd(string, required only when cidr is 0), name(string), description(string), touchDate(int64)
  - *GroupMemberType:* 0 - Group, 3 - IPCIDR, 4 - IPRange

  Method: POST
  URL:`^ipgroups/(?<groupid>[0-9]+)/member/$`

Remove a member from an IP group

- *Delete data fields:* To remove Group: memberType(GroupMemberType), memberGroupdId(int); To remove IPCIDR or IPRange: memberType(GroupMemberType), ipAddressStart(string), cidr(int), ipAddressEnd(string)
  - *GroupMemberType:* 0 - Group, 3 - IPCIDR, 4 - IPRange

Method: DELETE
URL:`^ipgroups/(?<groupid>[0-9]+)/member/$`

Get IP group members

Method: GET
URL:`^ipgroups/(?<groupid>[0-9]+)/members/$`

Bulk set IP group members

- *Put data fields:* array of objects with fields: ipAddressStart(string), cidr(int), ipAddressEnd(string, required only when cidr is 0), name(string), description(string)

Method: PUT
URL:`^ipgroups/(?<groupid>[0-9]+)/members/$`

Get IP group USN

- *Query parameters:* fqn(string)

Method: GET
URL:`^ipgroups/usn/`

13 Quarantine (message handling)

Get list of folders with day info

- *Query parameters:* utcOffset (int - in mins), maxRetentionDays(int,360)

Method: GET
URL:`^quarantine/folderswithdayinfo/$`

Get folder day info

- *Query parameters:* utcOffset (int - in mins), maxRetentionDays(int,360)

Method: GET
URL:`^quarantine/(?<folderid>[0-9]+)/(?<retention>[0-9]+)/$`

Get day info for a folder

- *Query parameters:* utcOffset (int - in mins), maxRetentionDays(int,360)

Method: GET
URL:`^quarantine/folders/(?<folderid>[0-9]+)/dayinfo/$`

Get bulk folder day info

- *Query parameters:* utcOffset (int - in mins), maxRetentionDays(int,360)
- *Post parameters:* map of "folder": retentionPeriod

Method: POST
URL:^`quarantine/folders/dayinfo/$`

Get list of classifications

Method: GET
URL:`^quarantine/classifications/$`

Get MML

- *Post data:* message(ProcessMessageKey), offset, bytes

Method: POST
URL:`^quarantine/mml/$`

Find message by specified parameters

- *Post data:* startTime(int, required), endTime(int, required), folderId(int), messageName(str), actionType(int), classification(int), fromUser(str), fromDomain(str), toOrFrom(bool), toUser(str), toDomain(str), minSize(int), maxSize(int), subject(str), searchHistory(bool), forwards(bool), blockNumber(int64), searchBlankSubject(bool), direction(int)
- *Query Parameters:* maxRows(int, required) *(if not set, no rows are returned)*

Method: POST
URL:`^quarantine/findmessage/$`

Find Receiver rejected message by specified parameters

- *Post data:* startTime(int), endTime(int), fromUser(str, optional), fromDomain(str, optional), toUser(str, optional), toDomain(str, optional), toOrFrom(bool, optional), andOperator(bool, optional), fromIPStart(str, optional), fromIPEnd(str, optional), speCustomerId(int, optional), forwards(bool, optional)
- *Query Parameters:* maxRows(int, required) *(if not set, no rows are returned)*

Method: POST
URL:`^quarantine/findrejectedmessage/$`

Get list of folders

- *Query parameters:* checkAccess (bool)

Method: GET
URL:`^quarantine/folders/$`

Get files for folderid

- *Query parameters:* filter (str), startTime (time_t), endTime (time_t), blockNumber (uint64), forwards (bool), maxMsgs (int)

  Method: GET
  URL:`^quarantine/folders/(?<folderid>[0-9]+)/$`

Check access for specific folder

Method: GET
URL:`^quarantine/folders/(?<folderid>[0-9]+)/checkaccess/$`

Get message count for a published folder

- *Query parameters:* user(recipient for published inbound, sender for published outbound: string)

  Method: GET
  URL:`^quarantine/folders/(?<folderid>[0-9]+)/messagecount/$`

Set folder properties

- *Post data:* ruleFolderItem (RuleFolderItem)

  Method: PUT
  URL:`^quarantine/folders/properties/$`

Get system folders

Method: GET
URL:`^quarantine/folders/system/$`

Forward a message to spiderlabs as spam

- *Post data:* messages (array of ProcessMessageKey), isSpam (bool), spamReportNotificationFromAddress (str)

  Method: POST
  URL:`^quarantine/forwardspam/$`

Delete specified messages

- *Post data:* messages (array of ProcessMessageKey), username (str), deleteType (int)

  Method: DELETE
  URL:`^quarantine/message/$`

Forward copy of message

- *Post data:* messages (array of ProcessMessageKey), deleteMessage (bool), recipients (str)
- *Optional post data:* speForwardCopyInfo (replyTo (str), fromAddr (str), speCustomerId (int), msgDirection (int))

  Method: POST
  URL:`^quarantine/message/forwardcopy/$`

Pass through message

- *Post data:* messages (array of ProcessMessageKey), userName (str), spamReportNotificationFromAddress (str), flags (int, required)

  Method: POST
  URL:`^quarantine/message/passthrough/$`

Get message properties

- *Post data:* messageKey field with json object with fields: blockNumber, edition, folderId, messageName, recipient, serverId, timeLogged

  Method: POST
  URL:`^quarantine/message/properties/$`

Restore specified messages from trash

- *Post data:* messages (array of ProcessMessageKey)

  Method: POST
  URL:`^quarantine/message/restore/$`

Trash specified messages

- *Post data:* messages (array of ProcessMessageKey)

  Method: POST
  URL:`^quarantine/message/trash/$`

Check user rights to specific messages

- *Post data:* messages (array of ProcessMessageKey), username (str), flags (int, required)

  Method: PUT
  URL:`^quarantine/message/validaterights/$`

Unpack message

- *Post data:* message (ProcessMessageKey), useContext (bool)

  Method: POST
  URL:`^quarantine/messagedetails/$`

Close unpacked message context

Method: DELETE
URL:`^quarantine/messagedetails/(?<contextid>[0-9]+)/$`

Retrieve unpacked message component

- *Post parameters:* contextId, componentName, offset, bytes

  Method: POST
  URL:`^quarantine/messagedetails/getunpackedcomponent/$`

Retrieve unpacked message component *(deprecated – Use the POST call above)*

- *Query parameters:* offset, bytes

  Method: GET
  URL:`^quarantine/messagedetails/(?<contextid>[0-9]+)/(?<component>.*)/$`

Convert component html/rtf/text into form safe to display

- *Post parameters:* bodyType, body

  Accepted values for bodyType:
  - mdtHTML
  - mdtRTF
  - mdtText

  Method: POST
  URL:`^quarantine/messagedetails/makesafehtml/$`

Get latest messages

- *Query parameters:* user(recipient for published inbound, sender for published outbound: string), messageCount(int)

  Method: GET
  URL:`^quarantine/messages/latest/$`

Get list of published folders

Method: GET
URL:`^quarantine/publishedfolders/$`

Get list of files for published folder

- *Query parameters:* user(recipient for published inbound, sender for published outbound: string), maxMsgs(messages limit: int)

  Method: GET
  URL:`^quarantine/publishedfolders/(?<folderid>[0-9]+)/$`

Register SSM url

- *Post data:* url

  Method: POST
  URL:`^quarantine/registerssm/$`

Unpack message in sender queue

- *Post data:* messageName (string), useContext (bool)

  Method: POST
  URL:`^quarantine/sendermessagedetails/(?<serverid>[0-9]+)/$`

Delete quarantine trashed messages permanently

Method: DELETE
URL:`^quarantine/trash/$`

Find Quarantine Audit entries

- *Query parameters:* fromTime(int), toTime(int), messageName(str), userName(str), recipientUser(str), recipientDomain(str), top(int, default 250), skip (int, default 0),sort(str, sort field),order(str, asc or desc)

  Method: GET
  URL:`^quarantine/findaudit/$`

14 Services

Get a list of the AMAX script files

Method: GET
URL:`^services/amaxscriptfiles/$`

Check if the product is licensed for BTM

Method: GET
URL:`^services/btmlicensed/$`

Get current config commit set ID

Method: GET
URL:`^services/config/commitset/current/$`

Get current config timestamp

Method: GET
URL:`^services/config/current/$`

Get latest SpamCensor updates

Method: GET
URL:`^services/spamcensorupdateresults/$`

Export the configuration

Caution: This function exports the latest configuration from the Array Manager (not the Configuration Service) to a zip file. It is not compatible with configuration exports from earlier versions of MailMarshal. If `policyOnly` is false, field-replaceable DLL files will be included in the output and the size of the file could be up to 100 MB. In this case use a client such as Curl that efficiently saves large file streams.

- *Query parameters:* policyOnly(bool), dkimPassword(string)

Method: GET
URL:`^services/config/export/$`

Import configuration

Caution: This function expects a zip file. It is only compatible with configuration files exported from version 10.X and above. It is not compatible with configuration exports from earlier versions of MailMarshal.

- *Query parameters:* merge(bool), replacedll(bool), dkimPassword(string)

Method: PUT
URL:`^services/config/import/$`

Import default configuration

Note: This function imports the DefaultRules.zip file from the product installation.

Method: GET
URL:`^services/config/importdefault/$`

Reload configuration for all servers.

- *Post data fields:* commitSetId(int), startStoppedServices(bool), forceOutOfSchedule(bool, false), needsRestart(bool, false)

  Method: POST
  URL:`^services/config/reload/$`

Reload server configuration.

- *Post data fields:* startStoppedServices(bool), forceOutOfSchedule(bool, false), needsRestart(bool, false)

  Method: POST
  URL:`^services/config/reload/(?<serverid>[0-9]+)/$`

Get database connection string used by ArrayManager to connect to database

Method: GET
URL:`^services/connectionstring/$`

Get the current admin user

Method: GET
URL:`^services/currentadminuser/$`

Check if delivery server is reachable or not.

- *Post data fields:* host(string), port(int)

  Method: POST
  URL:`^services/deliveryserver/check/(?<serverid>[0-9]+)/$`

Export password protected DKIM Private Key.

- *Query parameters:* passphrase(string)

  Method: GET
  URL:`^services/dkimkey/(?<domainname>((?!/).)+)/(?<selector>((?!/).)+)/$`

Import the DKIM key for a domain

Caution: This function places the key in the Windows CNG service, but it does not change configuration settings or expose the new key in the user interface. For information about how to accomplish these tasks, contact Support.

- *Put data fields:* key(base64 string), password(base64 string)

  Method: PUT
  URL:`^services/dkimkey/(?<domainname>((?!/).)+)/(?<selector>((?!/).)+)/$`

Delete the DKIM key for a domain

Caution: This function removes the key from the Windows CNG service, but it does not change configuration settings. If the key is marked for use in signing, signing will fail. Before deleting the key, you must ensure it is not in use. For information about how to change configuration settings through the API, contact Support.

Method: DELETE
URL:`^services/dkimkey/(?<domainname>((?!/).)+)/(?<selector>((?!/).)+)/$`

Lookup DNS record and check status of DKIM public key

Method: GET
URL:`^services/dkimkey/dnsrecord/(?<domainname>((?!/).)+)/(?<selector>((?!/).)+)/$`

Generate the DKIM key of specified keylength for a domain and selector

Caution: This function places the key in the Windows CNG service, but it does not change configuration settings or expose the new key in the user interface. For information about how to accomplish these tasks, contact Support.

Method: POST
URL:`^services/dkimkey/generate/(?<domainname>((?!/).)+)/(?<selector>((?!/).)+)/(?<keylength>[0-9]+)/$`

Get DKIM Public Key

Method: GET
URL:`^services/dkimkey/pubkey/(?<domainname>((?!/).)+)/(?<selector>((?!/).)+)/$`

Get the delivery routes for a domain

- *Query parameters:* specustomerid (int default=0), messagedirection (int default=0)

Method: GET
URL:`^services/domainroutes/(?<domainname>((?!/).)+)/$`

Run file update now

Method: POST
URL:`^services/fileupdate/run/$`

Get the license for MailMarshal

Method: GET
URL:`^services/license/$`

Set the license for MailMarshal

- *Put data fields:* licensekey(string)

Method: PUT
URL:`^services/license/$`

Create the initial license for MailMarshal

Method: PUT
URL:`^services/license/createinitial/$`

Create a temporary license for MailMarshal

Method: PUT
URL:`^services/license/createtemporary/$`

Request a license key

- *Post data fields:* reselleremail(string), contactname(string), contactemail(string), contactphone(string), contactcompany(string), contactaddress(string), customerreference(string), comments(string)

Method: POST
URL:`^services/license/request/$`

Get license key seeds

Method: GET
URL:`^services/license/seeds/$`

Get a list of MailMarshal Node Ids

Method: GET
URL:`^services/nodeids/$`

Test the Marshal RBL service

- *Post data fields:* customernumber(string), rblkey(string)

Method: POST
URL:`^services/rbl/test/$`

Validate a regex against the built in regex library

- *Post data fields:* regex(string), ignorecase(bool)

Method: POST
URL:`^services/regex/$`

Get Rule profile details

- *Query parameters:* speCustomerId(int, optional)

Method: GET
URL:`^services/ruleprofile/(?<serverid>[0-9]+)/$`

Get list of file types and groups

Method: GET
URL:`^services/filetypes/`

Get configuration service location

Method: GET
URL:`^services/configservice/`

Run script

Method: POST
URL:`^services/script/$`

Get security descriptor

Method: GET
URL:`^services/securitydescriptor/(?<oid>.+)/$`

Get security rights

Method: GET
URL:`^services/securityrights/$`

Get list of servers

Method: GET
URL:`^services/servers/$`

Get server details

Method: GET
URL:`^services/servers/(?<serverid>[0-9]+)/$`

Remove server from the array

- *Delete data fields:* force(bool, optional)

Method: DELETE
URL:`^services/servers/(?<serverid>[0-9]+)/$`

Get service status for specified service on server

Method: GET
URL:`^services/servers/(?<serverid>[0-9]+)/(?<serviceid>[0-9]+)/$`

Start or stop service for specified service on server

- *Put data fields:* control(string) = "start" or "stop"

Method: PUT
URL:`^services/servers/(?<serverid>[0-9]+)/(?<serviceid>[0-9]+)/$`

Set server description

- *Put data fields:* description(string)

Method: PUT
URL:`^services/servers/(?<serverid>[0-9]+)/description/$`

Set server location

- *Put data fields:* location(string)

Method: PUT
URL:`^services/servers/(?<serverid>[0-9]+)/location/$`

Rename server

- *Put data fields:* name(string), force(bool, optional)

Method: PUT
URL:`^services/servers/(?<serverid>[0-9]+)/name/$`

Get a list of the server ip addresses

Method: GET
URL:`^services/servers/ips/(?<serverid>[0-9]+)/$`

Get the number of processors available on a server

Method: GET
URL:`^services/servers/processors/(?<serverid>[0-9]+)/$`

Get a list of SpamCensor types for a SpamCensor script

Method: GET
URL:`^services/spamcensortypes/(?<filename>.+)/$`

Get a list of SpamCensor files

Method: GET
URL:`^services/spamconfigfiles/$`

Convert a Text Censor line

- *Post data fields:* line(string), score(number), style(number)

Method: POST
URL:`^services/textcensor/convert/$`

Get user count

Method: GET
URL:`^services/usercount/$`

Get list of virus scanners

Method: GET
URL:`^services/virusscanners/$`

Test that a command line scanner works properly

- *Post data fields:* command(object), data(base64 string)
- *Command object data fields:* name(string), exefile(string), params(string), trigger(string), notrigger(string), timeout(number), timeoutmb(number)

Method: POST
URL:`^services/virusscanners/commandtest/(?<serverid>[0-9]+)/$`

Test that a command line or DLL virus scanner works properly

- *Post data fields:* dll(string), options(string), data(base64 string)

Method: POST
URL:`^services/virusscanners/dlltest/(?<serverid>[0-9]+)/$`

Get McAfee DAT file information

Method: GET
URL:`^services/virusscanners/mcafee/datinfo/(?<serverid>[0-9]+)/$`

Update McAfee DAT files

Method: GET
URL:`^services/virusscanners/mcafee/datupdate/(?<serverid>[0-9]+)/$`

Get McAfee status

Method: GET
URL:`^services/virusscanners/mcafee/status/(?<serverid>[0-9]+)/$`

Get list of file types and groups

Method: GET
URL:`^services/filetypes/$`

Get configuration service location

Method: GET
URL:`^services/configservice/$`

Get DMARC record

- *Query parameters:* domain (str: domain)

Method: GET
URL:`^services/dmarc/record/$`

Get general array properties

Method: GET
URL:`^services/array-properties/general/$`

15 Spam Quarantine Manager Site

Get authentication mode

Method: GET
URL:`^spam/authenticationmode/$`

Set authentication mode

- *Put data fields:* authMode(int: 0 - None, 1 - Forms, 2 - NTLM), enableADIntegration(boolean)

Method: PUT
URL:`^spam/authenticationmode/$`

Get whether global blacklist is enabled

Method: GET
URL:`^spam/globalblacklist/$`

Enable or disable global blacklist

- *Put data fields:* enabled(boolean)

Method: PUT
URL:`^spam/globalblacklist/$`

Get whether global per user digest is enabled

Method: GET
URL:`^spam/globalperuserdigest/$`

Enable or disable global per user digest

- *Put data fields:* enabled(boolean)

Method: PUT
URL:`^spam/globalperuserdigest/$`

Get whether global whitelist is enabled

Method: GET
URL:`^spam/globalwhitelist/$`

Enable or disable global whitelist

- *Put data fields:* enabled(boolean)

Method: PUT
URL:`^spam/globalwhitelist/$`

Get subscribable digest of a user. If user is null, get all digests and their default subscription

- *Query parameters:* user(string: optional)

Method: GET
URL:`^spam/subscribabledigests/$`

Set subscribable digest of a user. If user is null, the default subscription for the digest

- *Put data fields:* user(string: optional), digestName(string), isSubscribed(boolean)

Method: PUT
URL:`^spam/subscribabledigests/$`

Get a user's information by alias

- *Query parameters:* alias(string)

Method: GET
URL:`^spam/user/$`

Create a new user

- *Post data fields:* userName(string), fullName(string), updateTime(int64), sendDigest(boolean), languageCode(string), chartsEnabled(boolean), isAdministrator(boolean), createPassword(boolean), customPassword(string)

Method: POST
URL:`^spam/user/$`

Delete a user by userid

- *Query parameters:* userid(int)

Method: DELETE
URL:`^spam/user/$`

Get a user's information

Method: GET
URL:`^spam/user/(?<user>((?!/).)+)/$`

Update a user's information

- *Put data fields:* password(string), fullName(string), lastLoggedOn(int64), updateTime(int64), sendDigest(boolean), userRole(int), chartsEnabled(boolean), languageCode(string), lastMessageViewed(int64), theme(string)

Method: PUT
URL:`^spam/user/(?<user>((?!/).)+)/$`

Create a new user

Method: POST
URL:`^spam/user/(?<user>((?!/).)+)/$`

Delete a user and associated aliases

Method: DELETE
URL:`^spam/user/(?<user>((?!/).)+)/$`

Delete an associated alias from a user

Method: DELETE
URL:`^spam/user/(?<user>((?!/).)+)/alias/(?<alias>((?!/).)+)/$`

Request to associate an alias to a user

- *Post data fields:* verificationUrl(string)

Method: POST
URL:`^spam/user/(?<user>((?!/).)+)/alias/(?<alias>((?!/).)+)/requestassociation/$`

Verify the association of an alias and a user

- *Post data fields:* verificationCode(string)

Method: POST
URL:`^spam/user/(?<user>((?!/).)+)/alias/(?<alias>((?!/).)+)/verifyassociation/$`

Get the list of aliases set up for this user

Method: GET
URL:`^spam/user/(?<user>((?!/).)+)/aliases/$`

Authenticate a user

- *Post data fields:* password(string)

Method: POST
URL:`^spam/user/(?<user>((?!/).)+)/authenticate/$`

Delete an alias from blocked list

Method: DELETE
URL:`^spam/user/(?<user>((?!/).)+)/blockedlist/alias/(?<alias>((?!/).)+)/$`

Allow another user to manage this user

Method: POST
URL:`^spam/user/(?<user>((?!/).)+)/delegate/(?<delegate>((?!/).)+)/$`

Delete an delegate from a user

Method: DELETE
URL:`^spam/user/(?<user>((?!/).)+)/delegate/(?<delegate>((?!/).)+)/$`

Get list of users who are allowed to manage this user

Method: GET
URL:`^spam/user/(?<user>((?!/).)+)/delegates/$`

Get list of users who are managed by this user

Method: GET
URL:`^spam/user/(?<user>((?!/).)+)/delegators/$`

Tell Array Manager that a user has been authenticated using NTLM

Method: PUT
URL:`^spam/user/(?<user>((?!/).)+)/ntlm/$`

Change a user's password

- *Put data fields:* oldPassword(string), newPassword(string)

Method: PUT
URL:`^spam/user/(?<user>((?!/).)+)/password/$`

Search user published messages by specific parameters

- *Post data fields:* fromDate(int64), toDate(int64), folderId(int), searchText(string), maxMsgs(int)

Method: POST
URL:`^spam/user/(?<user>((?!/).)+)/searchmail/$`

Get statistics of a user

- *Query parameters:* startTime(int64), stopTime(int64)

Method: GET
URL:`^spam/user/(?<user>((?!/).)+)/statistics/$`

Get whitelist or blacklist of a user

- *Query parameters:* isWhitelist(int: 1 - whitelist, 0 - blacklist)

Method: GET
URL:`^spam/user/(?<user>((?!/).)+)/whitelist/$`

Bulk set whitelist or blacklist of a user

- *Query parameters:* isWhitelist(int: 1 - whitelist, 0 - blacklist)
  - *Put data fields:* entries(array of strings)

Method: PUT
URL:`^spam/user/(?<user>((?!/).)+)/whitelist/$`

Delete whitelist or blacklist of a user

- *Query parameters:* isWhitelist(int: 1 - whitelist, 0 - blacklist)

Method: DELETE
URL:`^spam/user/(?<user>((?!/).)+)/whitelist/$`

Add an alias to whitelist or blacklist

- *Query parameters:* isWhitelist(int: 1 - whitelist, 0 - blacklist)

Method: POST
URL:`^spam/user/(?<user>((?!/).)+)/whitelist/alias/(?<alias>((?!/).)+)/$`

Delete an alias from whitelist

Method: DELETE
URL:`^spam/user/(?<user>((?!/).)+)/whitelist/alias/(?<alias>((?!/).)+)/$`

Get users information by key index

- *Query parameters:* keyIndex(char)

Method: GET
URL:`^spam/users/$`

Delete all users

Method: DELETE
URL:`^spam/users/$`

Backup users

Method: POST
URL:`^spam/users/backup/$`

Find users by matching a pattern against username

- *Post data fields:* searchPattern(string)

Method: POST
URL:`^spam/users/find/$`

Get list of key indexes of users

Method: GET
URL:`^spam/users/keyindexes/$`

Restore users

Method: POST
URL:`^spam/users/restore/$`

16 TextCensor

Important: Handlers to list, add, and delete TextCensor scripts are not currently implemented.

Create a new TextCensor tester

Method: POST
URL:`^textcensor/testers/$`

Delete a TextCensor tester

Method: DELETE
URL:`^textcensor/testers/(?<context>[0-9]+)/$`

Get result of TextCensor evaluation

Method: GET
URL:`^textcensor/testers/(?<context>[0-9]+)/evaluation/$`

Dump evaluation of script

Method: GET
URL:`^textcensor/testers/(?<context>[0-9]+)/evaluation/dump/$`

Add an expression to a TextCensor tester

- *Post data fields:* name(string), weight(integer), match(integer), expression(string)
  - *match:* match-limit
  - *expression:* TC expression base64-encoded

Method: POST
URL:`^textcensor/testers/(?<context>[0-9]+)/expressions/$`

Scan an uploaded file

Method: PUT
URL:`^textcensor/testers/(?<context>[0-9]+)/scan/(?<filecontext>[0-9]+)/$`

Scan the supplied UTF-8 data string

Method: POST
URL:`^textcensor/testers/(?<context>[0-9]+)/scan/data/$`

- *Post data field:* data(string)

17 TLS Commands for Array Manager

Get whether array has a private key

Method: GET
URL:`^arraytls/hasprivatekey/$`

Get whether array has a pending certificate signing key

Method: GET
URL:`^arraytls/haspendingcertsigningkey/$`

Get the current certificate details for array

Method: GET
URL:`^arraytls/certificate/$`

Generate a certificate signing request for array

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional)

Method: POST
URL:`^arraytls/createcertificatesigningrequest/$`

Generate a certificate signing request for array, allowing to input subject alternative names

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), alternativeNames(array of strings)

  Method: POST
  URL:`^arraytls/createcertificatesigningrequest2/$`

Generate a certificate signing request for array, allowing to input signature hash algorithm and subject alternative names

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), signMethod(int: 0 - sha1, 1 - sha256, 2 - sha512), alternativeNames(array of strings)

  Method: POST
  URL:`^arraytls/createcertificatesigningrequest3/$`

Generate a self-signed certificate and its public-private key pair for array

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), days(long)

  Method: POST
  URL:`^arraytls/createprivatekey/$`

Generate a self-signed certificate and its public-private key pair for array, allowing to input subject alternative names

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), days(long), alternativeNames(array of strings)

  Method: POST
  URL:`^arraytls/createprivatekey2/$`

Generate a self-signed certificate and its public-private key pair for array, allowing to input signature hash algorithm and subject alternative names

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), days(long), signMethod(int: 0 - sha1, 1 - sha256, 2 - sha512), alternativeNames(array of strings)

  Method: POST
  URL:`^arraytls/createprivatekey3/$`

Generate a PKCS#12 backup file from the certificate and the private key currently installed on array

- *Post data fields:* password(string: optional)

Method: POST
URL:`^arraytls/exportpkcs12file/$`

Import a signed certificate supplied by a CA for array

- *Post data fields:* certificate(string), privatekey(string), passphrase(string: optional)

  Method: POST
  URL:`^arraytls/importcertificate/$`

Import a PKCS#12 backup file for array

- *Post data fields:* data(base64 string), password(string: optional)

  Method: POST
  URL:`^arraytls/importpkcs12file/$`

18 TLS Commands for Node Server

Get the current certificate details for the server

Method: GET
URL:`^nodetls/certificate/(?<serverid>[0-9]+)/$`

Generate a certificate signing request for the server

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional)

  Method: POST
  URL:`^nodetls/createcertificatesigningrequest/(?<serverid>[0-9]+)/$`

Generate a certificate signing request for the server, allowing to input subject alternative names

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), alternativeNames(array of strings)

  Method: POST
  URL:`^nodetls/createcertificatesigningrequest2/(?<serverid>[0-9]+)/$`

Generate a certificate signing request for the server, allowing to input signature hash algorithm and subject alternative names

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), signMethod(int: 0 - sha1, 1 - sha256, 2 - sha512), alternativeNames(array of strings)

  Method: POST
  URL:`^nodetls/createcertificatesigningrequest3/(?<serverid>[0-9]+)/$`

Generate a self-signed certificate and its public-private key pair for the server

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), days(long)

  Method: POST
  URL:`^nodetls/createprivatekey/(?<serverid>[0-9]+)/$`

Generate a self-signed certificate and its public-private key pair for the server, allowing to input subject alternative names

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), days(long), alternativeNames(array of strings)

  Method: POST
  URL:`^nodetls/createprivatekey2/(?<serverid>[0-9]+)/$`

Generate a self-signed certificate and its public-private key pair for the server, allowing to input signature hash algorithm and subject alternative names

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), days(long), signMethod(int: 0 - sha1, 1 - sha256, 2 - sha512), alternativeNames(array of strings)

  Method: POST
  URL:`^nodetls/createprivatekey3/(?<serverid>[0-9]+)/$`

Generate a PKCS#12 backup file from the certificate and the private key currently installed on the server

- *Post data fields:* password(string: optional)

  Method: POST
  URL:`^nodetls/exportpkcs12file/(?<serverid>[0-9]+)/$`

Get whether the server has a pending certificate signing key

Method: GET
URL:`^nodetls/haspendingcertsigningkey/(?<serverid>[0-9]+)/$`

Get whether the server has a private key

Method: GET
URL:`^nodetls/hasprivatekey/(?<serverid>[0-9]+)/$`

Import a signed certificate supplied by a CA for the server

- *Post data fields:* certificate(string), privatekey(string), passphrase(string: optional)

Method: POST
URL:`^nodetls/importcertificate/(?<serverid>[0-9]+)/$`

Import a PKCS#12 backup file for the server

- *Post data fields:* data(base64 string), password(string: optional)

Method: POST
URL:`^nodetls/importpkcs12file/(?<serverid>[0-9]+)/$`

19 TLS commands for Syslog

In this version of MailMarshal, Syslog TLS uses the certificate of the Array Manager REST API.

20 Version information

Get version information

Method: GET
URL:`^version/$`

Get configuration version

Method: GET
URL:`^version/config/`

Get product version

Method: GET
URL:`^version/product/`

Get RPC interface version

Method: GET
URL:`^version/rpcinterface/`

About LevelBlue

LevelBlue reduces risk and builds lasting resilience so organizations can innovate and advance their mission with confidence. As the world’s most analyst-recognized and largest pure-play managed security services provider, LevelBlue elevates client outcomes that matter: stronger defense, faster response, and sustained business continuity. LevelBlue combines AI-powered security operations, advanced threat intelligence, and elite human expertise to provide the most comprehensive portfolio of strategic advisory, managed security, offensive security, and incident response services.