Documentation for OnTime MS – Api

The audience for this documentation is developers with basic knowledge of REST calls and HTTP protocol. It is also required some knowledge in how OnTime Group Calendar works in general.

Contents

Authentication

OnTime for MS has two different login schemas depending on configuration in the Admin, under “Global” – “Backend” – “Authentication”. If not set, the system uses “Form Based”. If specified the system uses “Authentication Service”. The two authentication method are different in as well interface as functionality. But the end result is the same, to get a short time OnTime Token that can be used to the data /Login function

Form Based

The Form based authentication is done by sending a Form POST request to the auth api function, with body containing the form data “usr” (typically email) and “pwd” (password). The http header "Content-type" need to be set to "application/x-www-form-urlencoded".
A valid user and password will return a Token. This token is a short time token (default 30sec.), that can only be used to login to the data api “POST /Login” call.

Example

Request

```
POST /ontimegcms/api/1/auth HTTP/1.1
Host: localhost:8080
Accept: */*
Content-type: application/x-www-form-urlencoded
Content-Length: 33

usr=jondoe@acme.com&pwd=OnTime123
```

Response

Succes fully login

```
{
   "Status":"OK",
   "Token":"eyJUaW1lc3RhbXAiOjE0OTc0NDIzMjQzMTQsIkVtYWlsIjoibW5zQG9udGltZS5sb2NhbCJ9"
}
```

Incorrect login

```
{
    "Status": "ERROR",
    "Error": {
        "Code": "AuthenticationError",
        "Description": "User not authenticated"
    }
}
```

Authentication Service

The Authentication service is a service that allow the developer, to send the browser to another url, get authenticated and return to any specified url (by web redirect), with info about the login.
The service has internal multiple authentication schemas and thereby working together with the browser to provide support for Negotiate, Basic & NTLM, and this way giving support for SSO depending on browser and OS settings.

If a user need to login, the browser should be redirected to the Authentication Service URL specified in the OnTime configuration, with the parameter “redirect={url}”, where {url} is the url the browser should be redirected to when authenticated, regardless of successfully or denied access.
The redirected url then gets appended the parameter “?response={response_data}” that contains info about the result.
{response_data} is a base64 encoded representation of a json data just like the Form Based.

Example

```
{
   "Status":"OK",
   "Token":"eyJUaW1lc3RhbXAiOjE0OTc0NDYwMTcwMDAsIkVtYWlsIjoibW5zQG9udGltZS5sb2NhbCJ9"
}
```

If you as a developer need to embed the Authentication Service into you own code instead of use standard browser redirection (due to language choice or implementation), you could perform that “simple” as long Basic Authentication is allowed in your environment.

Request

```
GET /ontime/auth.html?redirect=none HTTP/1.1
Host: localhost:80
Accept: */*
Authorization: Basic am9obmRvZUBhY21lLmNvbTpPblRpbWUxMjM=
```

PS: Authorization header contains a base64 encoding of {user}:{password}

Response

```
HTTP/1.1 303 See Other
Location: none?response=eyJTdGF0dXMiOiJPSyIsIlRva2VuI…
```

Where the response= value from the “Location” header is the {response_data} like above.
If your developing environment automatically follow 303 redirects, then the {response_data} can be retrieved from the redirected url parameter “response=”.

In case of invalid user/password it could return

Response

```
HTTP/1.1 401 Unauthorized
Content-Type: text/html
Date: Wed, 14 Jun 2017 14:20:05 GMT
Server: Microsoft-HTTPAPI/2.0
WWW-Authenticate: Negotiate
WWW-Authenticate: NTLM
WWW-Authenticate: Basic realm="OnTimeSuite"
```

Where the “WWW-Authenticate” tells valid authentication methods supported.

General

All calls are as standard POST. If need some resource (image) to be cached by client browser, a GET has be implemented in these cases.
The main concept of all POST calls is “json in – json out”, and all with character encoding UTF-8 chars.

The prefix URL for all Api calls is “[contextPath]/api/[apiVersion]/[data]/…”
Where

[contextPath] is the protocol, host, port, installPath defined by setup.
[apiVersion] is the api version for the request – currently ONLY 2 is supported.

The request is a Json string with some mandatory elements to identify request “ApplID”, “ApplVer”, “Token”, and a Data object with several request specific elements.
Result is json data with “Status” and “Token” and “Data” (result of request)

Example:
Request

```
{
   “ApplID” : ”ApiExplorer”    String*     The application id according to license
   “ApplVer” : “1.0.0”         String      The application version, only for logging
   “Token” : “xxxx…”           String*     The user token from previous response
   “Data” : {                  Object*     Object with request elements
   }
}
```

Response

```
{
   “Status” : “OK”,
   “Token” : “xxxx…”,
   “Data” : {
      ::
   }
}
```

```
{
   “Status” : “ERROR”,
   “Error” : {
      “Code” : “InvalidToken”,
      “Description” : “The provided token is invalid”,
      ::
   }
}
```

If a return Token present, this should be use during next call to API, as server can issue new token whenever needed.
Status, can be “OK” or “ERROR”. In case of “ERROR” an error object with “Code” and a “Description” will be provided, where the “Code” is a short unique code for the error, and can be used to identify how to be handle, but perhaps also as index for a translated description. “Description” is an English description of error, if no translation is available.
Other Error info can be provided for tracing/debugging purpose

General information:

*	Indicated a mandatory field
DateTime format	Is following format, and only Zulu time: YYYY-MM-DDThh:mm:ssZ

General object sets

Following is a list of “standard” objects that the request/response may use – See each API call for details and reference to these objects

{User}

```
{
   "UserID"              String
   "Email"               String
   "Type"                String        "Person"|“Room”|“Equipment”,
   "ContactSource"       String        “ActiveDirectory"|???

   "CanCreate"           Boolean
   "DeleteItems"         String        "All"|”None”
   "ReadItems"           String        "FullDetails"|”TimeOnly”|???
   "EditItems"           String        "All"|”Owned”|”None”

   "DisplayName"         String
   "FirstName"           String
   "LastName"            String
   "CompanyName"         String
   "Department"          String
   "JobTitle"            String
   "MobilePhone"         String
   "Culture":            String        Language selected on Exchange profile


   "Photo"               String        Base64 representation of a small jpg image.
   "ImageType"           String        "jpg"|”png”|”gif”
   "ImageChangeKey":     String        Value that change if image is updated.
}
```

{Group}

```
{
   “GroupID”             String
   “Name”                String
   “MembersCount”        Number
   “DisplayName”         String        Only available for Distribution List groups
   “Private”             Boolean       Only available if a private group
   “Application”         Boolean       Only available if an application group
   “SortManually”        Boolean       Only available manually sorted group
   “SortOrder”           String[]      UserIDs in sort order. (Only manual sorted groups)
}
```

{Event}

```
{
   “EventID”             String
   “Start”               Date
   “End”                 Date
   “Subject”             String
   “ShowAs”              String        “Away”|“Busy”|“Tentative”|“Free”|”WorkingElsewhere”
   “Sensitivity”         String        “Normal”|“Private”
   “Categories”          String[]
   “Type”                String        “Meeting”|”Invitation”
   “ChangeKey”           String
   “Organizer”           String        Here it’s only Display Name
   “RequiredAttendees”   String[]      Here it’s only Display Name
   “OptionalAttendees”   String[]      Here it’s only Display Name
   “IsCancelled”         Boolean
   “IsRecurring”         Boolean
}
```

Login

POST /Login

Used to verify a Token and ApplID, and getting current user and some server info.
This is the only call that accept the “onetime” token return from the Authentication.

Request

```
“Data” : {}
```

Response

Ex 1 – Valid Token:

```
{
   “Status” : “OK”,
   “Data” : {
      “Me” : {User},           Object      UserDetails for current logged in user.
      “Server” : {
         “Version” : …
         “SyncBackTime” …
         “SyncForwardTime” …
         “ExchangeFeatures”                1 = Basic
                                           2 = supports ShowAs ”WorkingElsewhere”, Event Body info”
          ::
      }
   }
}
```

Ex 2 – Token Expired:

```
{
   “Status” : “ERROR”,
   “Error” : {
      “Code” : “TokenExpired”,
      “Text” : “Token has been expired”
   }
}
```

Typical ErrorCodes:

- TokenInvalid
- TokenNoUser
- TokenExpired

Settings

POST /Settings/Get

Get settings for given Application and all Admin settings

Request

```
"Data" : {}
```

Response

Ex - AppIID eq "Desktop"

```
"Data" : {
   "Admin" : {              Objects from the Admin module regarding users settings
      "DefaultSettings"     Object
      "Legends"             Object
        ::                  Some few extra only relevant for the Admin client.
   },
   "ApiExplorer" : {        Objects for this Application
      "[Key1]"              Object
        ::
      "[KeyN]"              Object
   }
}
```

POST /Settings/Set

Save settings for given application. Settings are list of Key/Object pair, where key can be any json key

Request

```
"Data" : {
   "[Key1]"          Object
     ::
   "[KeyN]"          Object
}
```

Response

```
"Data" : {}
```

Group

POST /Group/List

Return a list of groups, according to parameters.
No parameters mean a list of ALL groups, available for user.

Request

```
"Data" : {
   "IncludePublic"      Boolean
   "IncludePrivate"     Boolean
   "Partial"            String       Find group entries matching start (only for public)
}
```

Response

Ex - No Param

```
"Data" : {
   "Items" : [
      {Group},
        ::
   ]
}
```

Ex - "Partial":"" and "IncludePublic":true

For partial the {group}, object is replaced by a simplified object only containing partial name, for the subgroups – This is a way for the UI to "load" group structure only when opened.

```
"Data" : {
   "Items" : [
      { "Name" : "A" ,"GroupID" : "…" },
      { "Name" : "B\\"},
      { "Name" : "C\\"},
      { "Name" : "D" ,"GroupID" : "…" },
   ]
}
```

Ex - "Partial":"B\\" and "IncludePublic":true

```
"Data" : {
   "Items" : [
      { "Name" : "B\\1" ,"GroupID" : "…" },
      { "Name" : "B\\2" ,"GroupID" : "…" },
      { "Name" : "B\\3\\"},
      { "Name" : "B\\Test\\" }
   ]
}
```

POST /Group/Search

Request

```
"Data" : {
   "Pattern"       String       Search for groups that match,
   "MaxCount"      Number       Default: 10
}
```

Response

```
"Data" : {
   "Items" : [
      {Group},                  Group Object
       ::
   ]
}
```

POST /Group/Details

Return details of a given group.

Request

```
"Data" : {
   "GroupID"          String*
}
```

Response

```
"Data" : {Group}      Group Object
}
```

POST /Group/Create

Create a Group with given Name and user IDs, and return the new GroupID.

Request

```
"Data" : {
   "Name"             String*          (Empty string is not allowed)
   "UserIDs"          String[]*
   "SortManually"     Boolean          (default: false)
   "Application"      Boolean          (default: false)
}
```

Response

```
"Data" : {
   "GroupID":         String
}
```

POST /Group/Update

Update existing group

Request

```
"Data" : {
   "GroupID"          String*
   "Name"             String
   "UserIDs"          String[]
   "SortManually"     Boolean
}
```

Response

```
"Data" : {}
```

POST /Group/Delete

Remove a given group.

Request

```
"Data" : {
   "GroupID"          String*
}
```

Response

```
"Data" : {}
```

User

POST /User/List

Returns user details of a list of UserIDs, Emails addresses or a GroupID.

Request

```
"Data" : {
   "UserIDs"          String[]
   "Emails"           String[]
   "GroupID"          String
}
```

Note: Combining "UserIDs", "Emails", "GroupID" has undefined result. It's not (by api) given if the result includes only one of the request, all or an error.

Response

```
"Data" : {
   "Items" : [
      {User}                   User Object
       ::
   ],
   "SortOrder" : ["userID",… ]    Only if by GroupID and group is manually sorted.
}
```

POST /User/Search

Search for user by Pattern, and return MaxCount

Request

```
"Data" : {
   "Pattern"          String*      (future split with space)
   "UserType"         String       "Person"|"Room"|"Equipment"  (none means all)
   "MaxCount"         Number       Default 10 – Absolut max ? (100)
}
```

Response

```
"Data" : {
   "Items" : [
      {User},         User Object
       ::
   ]
}
```

Room

POST /Room/List

Returns a list of all Rooms.

Request

```
"Data" : {}
```

Response

```
"Data" : {
   "Items" : [
      {User},            User Object
       ::
   ]
}
```

POST /Room/Options

Returns a list of options for search criteria – pt. only Locations

Request

```
"Data" : {}
```

Response

```
"Data" : {
   "Locations" : […]     Array of unique Locations from all rooms.
}
```

Equipment

POST /Equipment/List

Returns a list of all Equipment's.

Request

```
"Data" : {}
```

Response

```
"Data" : {
   "Items" : [
      {User},            User-object
       ::
   ]
}
```

POST /Equipment/Options

Returns a list of options for search criteria – pt. only Locations

Request

```
"Data" : {}
```

Response

```
"Data" : {
   "Locations" : […]     Array of unique Locations from all rooms.
}
```

Event

POST /Event/List

Request

```
"Data" : {
   "UserIDs"       String[]*
   "From"          Date*
   "To"            Date*
}
```

Response

```
"Data" : {
   "Items" : [
      {
         "UserID"              String      UserID as in request
         "Events" : [
            {Event}            Event-Object
              ::
         ]
      }
   ]
}
```

POST /Event/Details

Events retrieved from POST /Event/List are limited on several information, so to get all detailed event

Request

```
"Data" : {
   "EventID"String*
}
```

Response

```
"Data" : {
   "Event" : {        Event-object with modifications
       ::
      "RequiredAttendees"      String[]         Array of emails of Required attendees
      "OptionalAttendees"      String[]         Array of emails of Optional attendees
      "Body"                   String           The content of body
      "EffectiveRights" : {    Object           Object of effective rights for call user
         "CreateAssociate"     Boolean
         "CreateContents"      Boolean
         "CreateHierarchy"     Boolean
         "Delete"              Boolean
         "Modify"              Boolean
         "Read"                Boolean
         "ViewPrivateItems"    Boolean
      }
   }
}
```

POST /Event/Create

Note: Not provided properties, means use default

Request

```
"Data" : {
   "UserID"              String*
   "Start"               Date*
   "End"                 Date*
   "Subject"             String    (max 255 chars)                    (default: empty)
   "ShowAs"              String                                       (default: "Busy")
   "Sensitivity"         String                                       (default: "Normal")
   "Body"                String    (newline "\n" as line separator)   (default: empty)
   "Categories"          String[]                                     (default: none)
   "Location"            String                                       (default: empty)
   "RequiredAttendees"   String[]  (max 255 chars pr. email)          (default: none)
   "OptionalAttendees"   String[]  (max 255 chars pr. email)          (default: none)
}
```

Response

```
"Data" : {
   "Event" : {Event}     Event-Object
}
```

POST /Event/Update

Note: Not provided fields, means don't change this.

Request

```
"Data" : {
   "EventID"             String*
   "ChangeKey"           String
   "Start"               Date
   "End"                 Date
   "Subject"             String    (max 255 chars)
   "Location"            String
   "Categories"          String[]
   "ShowAs"              String
   "Sensitivity"         String
   "RequiredAttendees"   String[]  (max 255 chars pr. email)
   "OptionalAttendees"   String[]  (max 255 chars pr. email)
}
```

Response

```
"Data" : {
   "Event" : {Event}     Event-object
}
```

POST /Event/Delete

Request

```
"Data" : {
   "EventID"             String*
   "ChangeKey"           String
}
```

Response

No Data, only Status

Free

POST /Free/Rooms

Get a list of free Rooms, within a given time and options.
A room with no capacity, should not compare capacity.
Optional fields not in request shouldn't be compared

Request

```
"Data" : {
   "From"          Date*
   "To"            Date*
   "Capacity"      String
   "Location"      String
}
```

Response

```
"Data" : {
   "UserIDs"       String[]
}
```

POST /Free/Equipments

Get a list of free Equipment's, within a given time and options.
A room with no capacity, should not compare capacity.
Optional fields not in request shouldn't be compared

Request

```
"Data" : {
   "From"          Date*
   "To"            Date*
   "Capacity"      String
   "Location"      String
}
```

Response

```
"Data" : {
   "UserIDs"       String[]
}
```

POST /Free/Search

Get a combined free time list of a set of users

Request

```
"Data" : {
   "From"            Date*
   "To"              Date*
   "UserIDs"         String[]*
}
```

Response

```
"Data" : {
   "Items" : [
      {
      Start          Date
      End            Date
      },
      ::
   ]
}
```

Comment

Currently working hours are not part of OnTime API. When implemented, this would be a client task, as working hours is in user's local time, and the server don't have any info about this.

Mail

POST /Mail/Send

Send a mail to a user.

Request

```
"Data" : {
   "To"            String*   (Email of receiver)
   "Subject"       String
   "Body"          String    (newline "\n" are line separator)
}
```

Response

No Data, only a Status.

Image

POST /Image/Details

Request base64 representation of a user's image.
In case the image requested size isn't available take the nearest. Know sizes are currently 48x48 and 96x96

Request

```
"Data" : {
   "UserID"        String*
   "Width"         Number*
   "Height"        Number*
}
```

Note: Sizes can currently only be "Width:48, Height:48" or "Width:96, Height:96"

Response

```
"Data" : {
   "Base64"        String
   "Width"         Number
   "Height"        Number
   "Type"          String       ("jpg"|"png"|"gif")
}
```

GET /image/get

URL

.../image/get/{UserID}_{Size}.{ImageType}[?ImageChangeKey]

Size can be either "48x48" or "96x96"
The "ImageType" is from User object.
The "ImageChangeKey" is from User object, but optional, and "only" to ensure re-cache in client.
In case the server doesn't have a user image, the api returns a dummy 1x1 pixel image.

Version history:

8 jun. 2017 – api 2 rev 2
Authentication section added.
New layout of this document.
SortManually added on the Groups requests.
Made a General object sets section.

29 aug. 2016 – api 2 rev 1
'…/Room/Options': New.
'…/Equipment/List': New.
'…/Equipment/Options': New.
'…/Free/Rooms': New.
'…/Free/Equipments': New.
'…/Free/Search': New.
'…/Group/Search': New
'…/Mail/Send': New
'…/Image/Get': New.
'…/Image/Details': New.
'…/User/List':
- Added GroupID in request
- Added OfficeLocation
- Added ImageType
- Added ImageChangeKey

'…/User/Search':
- Added UserType

'…/Group/List':
- Removed Seperator,
- Partial implemented,
- Added InclPrivate, InclPublic
- Added MembersCount
- Added Application

'…/Group/Details':
- Returns UserIDs instead of User objects
- Application added

'…/Group/Create':
- Application added

18 jan. 2016 - api 1 rev 4
'…/Event/Details'
'…/Group/Members' renamed to '…/Group/Details'

21 dec. 2015 - api 1 rev 3
"Data" in request, and fixup

21 dec. 2015 - api 1 rev 2
Multible updates

16 dec. 2015 - api 1 rev 1
First version of this documentation