REST API Sample/Excerpt

Gargantua 8 REST API

Table of contents

Gargantua 7 REST API Basics

Gargantua 7 REST API Basics

This document describes the REST API of the Gargantua server.

Communicating with the Gargantua server

The Gargantua server handles requests. A request comes in form of a command with an input data structure and yields an output data structure upon completion.

Gargantua server commands fall in several different categories, as described in the table below

Administration
: Commands that will handle administration objects; these are objects that are server-wide scoped, like users, groups, databases, etc.

Design
: Commands that will handle design objects; these objects are defined on a per-database schema, and are used for definingt runtime behavior of the server, like attributes, forms, workflow processes, etc.

Production
: Production commands handle all content stored within a Gargantua server. These can furthermore be divided into document management, workflow and search.

REST API access URL

The REST API is accessible on the `/api/rest` path relative to the base URL of the Gargantua web application.

Example: if the web application is accessible at `http://example.domain.com/siatel-web/` then the REST API is accessible at the `http://example.domain.com/siatel-web/api/rest`.

The generic form of an access URL is

```
http://example.domain.com:port/siatel-web/api/rest/<service>/<command>/<object>/<extra>
```

<service>
: The name of the service that must handle the request; available services are : `auth`, `admin`, `design`, `ged`, `flow`, `search`, `security`, `storage` and `misc`; refer to the next section for details for each service.

<command>
: The name of the command to be executed, normally a verb; the command set is service and object dependent, and are described individually; here are some examples : `login`, `logout`, `create`, `update`, `delete`, etc.

<object>

The target object type of the call command request; these, too are service dependent; for instance, the ged service will handle object types folder, document, etc.

<extra>

An extra element for commands that require it; this extra element is documented for commands that require it.

> ***Depending on the service and command involved, the object type may be optional, as well as the extra element.***

Below are a couple of example of URLs :

```
http://example.domain.com:port/siatel-web/api/rest/auth/login/
```

```
http://example.domain.com:port/siatel-web/api/rest/ged/create/folder
```

```
http://example.domain.com:port/siatel-web/api/rest/ged/read/document
```

REST API SERVICES

The list below briefly describes the available services for the REST API

auth

handles all requests related to a user session (authentication, termination, etc.)

admin

handles all request on administration objects

design

handles all requests on design objects

ged

handles document management related requests; please note that documents within workflow processes are also handled by this service

flow

handles all task and process related requests; process content is handled by the ged service

search

handles search requests; this include all types of search available on the server

security

handles security management requests for all Gargantua objects

storage

handles file storage requests, most notably storing and retrieveing files

misc

handles various commands that do not fall in any of the above categories, such as sending an email or retrieving pdf renditions of a document

Each request sent to a Gargantua server must include a connection token in the HTTP header, using field name 'conn', except for the authentication command that does not require this token, but returns it.

The request body is the input data structure, in JSON format; for commands that require it, a file can be attached to the request (requires the POST method).

The request outcome is the response body, also in JSON format.

Here are some examples (for complete details, see each commands individually)

This section describes the input and output JSON-formatted data structures

Input data structure

The input data structure uses the following keys; the combimation of these keys are command dependent and are described for each command individually.

flags

`long`; specifies the command flags; see each command for details.

offset

`int`; an offset for the starting element of an enumeration command's list of results; usually optional, defaults to `0`.

count

`int`; the maximum number of elements to be returned by an enumeration command; usually optional, defaults to `-1` which represents 'no limit'.

criteriaIds

`array`; an array of filter criteria for an enumeration; see each command for details and possible filtering criteria.

object

`object`; a Gargantua object.

objectId

`string`; the identifier (id) of a Gargantua object.

objectLabel

`string`; the name (label) of a Gargantua object.

parentId

`string`; the identifier of the parent object when working with nested objects; for example when creating a document the `parentId` specifies a folder or a process.

targetId

`string`; the identifier of a destination for a move or copy-paste command.

afterId

`string`; the identifier of a position reference within a container for a reposition command.

user

`string`; the login name of the user when opening a session (login command).

pass

`string`; the user password when opening a session (login command).

For example, a read user command will send a request like the one below:

```
{
    "flags":1080933179801481247,
    "object": {
        "prefix":"USER",
        "id":"USER0000000000D0000000000000000000000"
    }
}
```

Output data structure

The output data structure has the following keys

result

`string`; the outcome of a command call, `"ok"` in case of success, `"fail"` otherwise, in which case the `"error"` key provides additional details.

dtoOut

`object`; the result(s) of a command call; this is actually the payload of the server response to the command request; see each command for details.

error

`string`; details provided in case the command call did not succeed; usually a java exception stacktrace.

When an object or list of objects is returned, each JSON object mapped to a Gargantua object will use the keys described in this section.

For example the response of a read attribute command would look like this:

```
{
    "result": "ok",
    "dtoOut": {
        "object": {
            "prefix": "DFAT",
            "id": "DFAT000200000076000000000000000000000000",
            "typeId": "DFTY000200000001000000000000000000000000",
            "value": null,
            "label": "string attribute",
            "ownerId": "USER000000000001000000000000000000000000",
            "flags": 0,
            "dateC": 1493106793024,
            "dateM": 1493106793024,
            "description": "",
            "data": null,
            "packages": []
        }
    }
}
```


Data types

Data types are restricted to the data types allowed by the JSON format; the following conversions must be applied to higher level language data types

timestamps

Long (64 bit) values that represent a Java time, also known as Unix time (01/01/1970 00:00:00 UTC based). For example the `dateC` and `dateM` fields of most Gargantua objects.

flags

Long (64 bit) values representing bitwise combinations on command options or Gargantua object properties depending on the command context; the `Flag` java class documentation (javadoc) details the significance for each bit in the 0-62 range.

strings

String values in UTF-8 encoding.

Object keys

prefix

`string`; first 4 characters of a Gargantua object id; indicates the object type; used to indicate to a command the object type when an object id is not available (for example retrieving an object by label); always the first key inside an `"object"` key.

id

`string`; the object identifier.

label

`string`; the name of the object; limited to 256 characters.

dateC

`long`; object creation timestamp.

dateM

`long`; object last modification timestamp.

flags

`long`; object flags

ownerId

`string`; id of the user who is the owner of the object.

formId

`string`; the identifier of the object form (for folders, pins, documents, processes).

attributes

`array`; the array of user defined attributes of an object (for folders, pins, documents, processes); each element in this array is a subsequent Gargantua 'attribute' object and consequently bears some of the keys described in this section.

typeId

`string`; the type identifier of an attribute

value

`string`, `number`, `boolean` or `null`; the value of an attribute; for date/timestamp attributes, the value (if non-null) is numeric and represents a `timestamp`.

fileId

`string`; a stored file's identifier

description

`string`; a description field for administration and design objects.

packages

`array`; an array of Gargantua "package" objects to which administration and design objects may be assigned to.

data

`string`; a field to hold object custom data for some administration and design objects; used by `attribute` and `assignment` objects.

Additional information

This section addresses detailed information on several topics.

Object flags

Objects generally have complex properties; to avoid unnecessary database overload, a caller application can request only certain properties for the objects it retrieves; similarly, when creating or updating an object, a caller application can specify which fields are provided for a given operation.

All flags are bit values, which means combinations can be made using bitwise operators.

Flags for different objects may overlap; this means that a flag assigned for one particular object may have the same numeric value as the flag assigned to a different object. For instance, the "real name" field of the "user" object has the same numeric value (same bit) as the "script type" field of the "script" object. However, this never results in conflicts since requests never return multiple objects of different types.

Common flags

These flags apply to all objects that share the corresponding properties. For a complete list of flags and combinations, please use the generated source code files.

id

0x0000000000000001 hex or 1 decimal

label

0x0000000000000002 hex or 2 decimal

ownerId

0x0000000000000004 hex or 4 decimal

flags

0x0000000000000008 hex or 8 decimal

description

0x0000000000000010 hex or 16 decimal

formId

0x0000000000000080 hex or 128 decimal

attributes

0x0000000000000100 hex or 256 decimal

fileId

0x0000000000000200 hex or 512 decimal

dateC

0x0000000000001000 hex or 4096 decimal

dateM

0x0000000000002000 hex or 8192 decimal

Specific flags

EXAMPLES

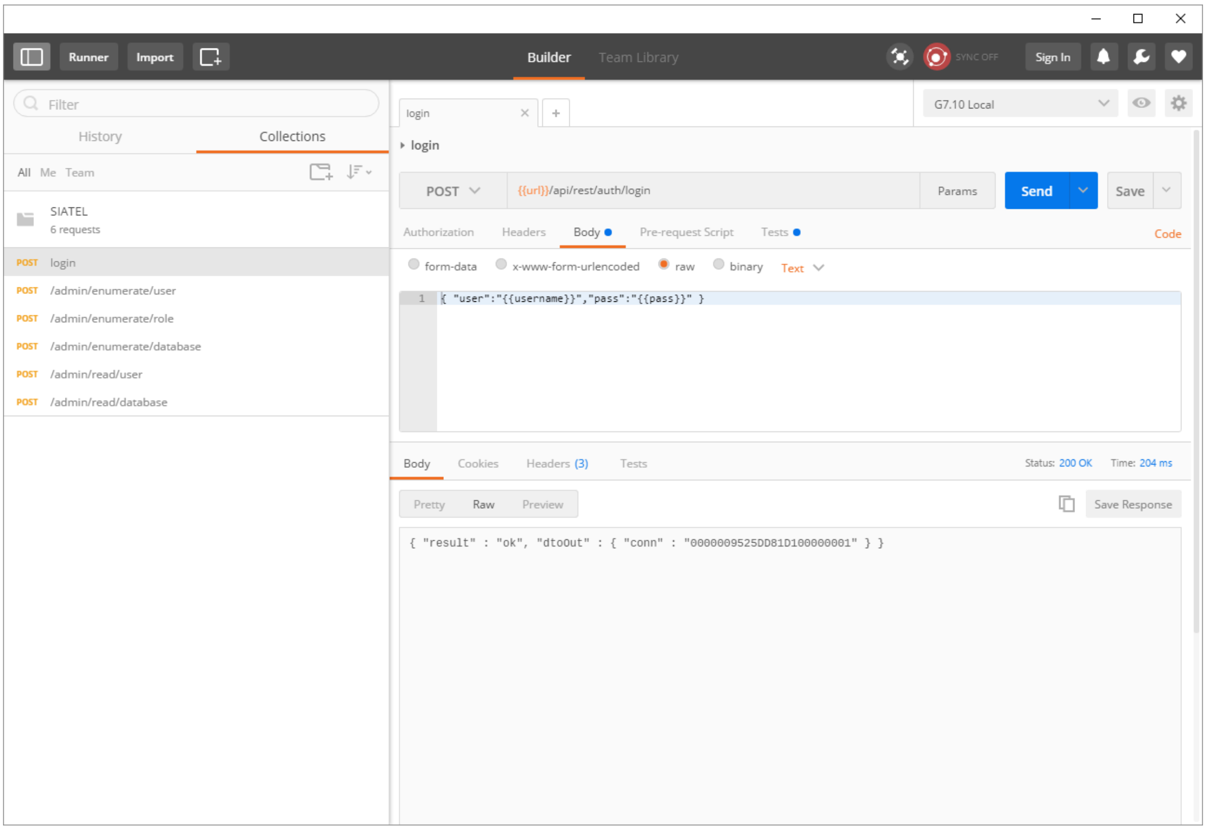
This section includes a few examples on how to use the Gargantua REST API.

TEST ENVIRONMENT

A test environment can be setup to test the Gargantua REST API. This environment can also be used to validate requests made by a calling application.

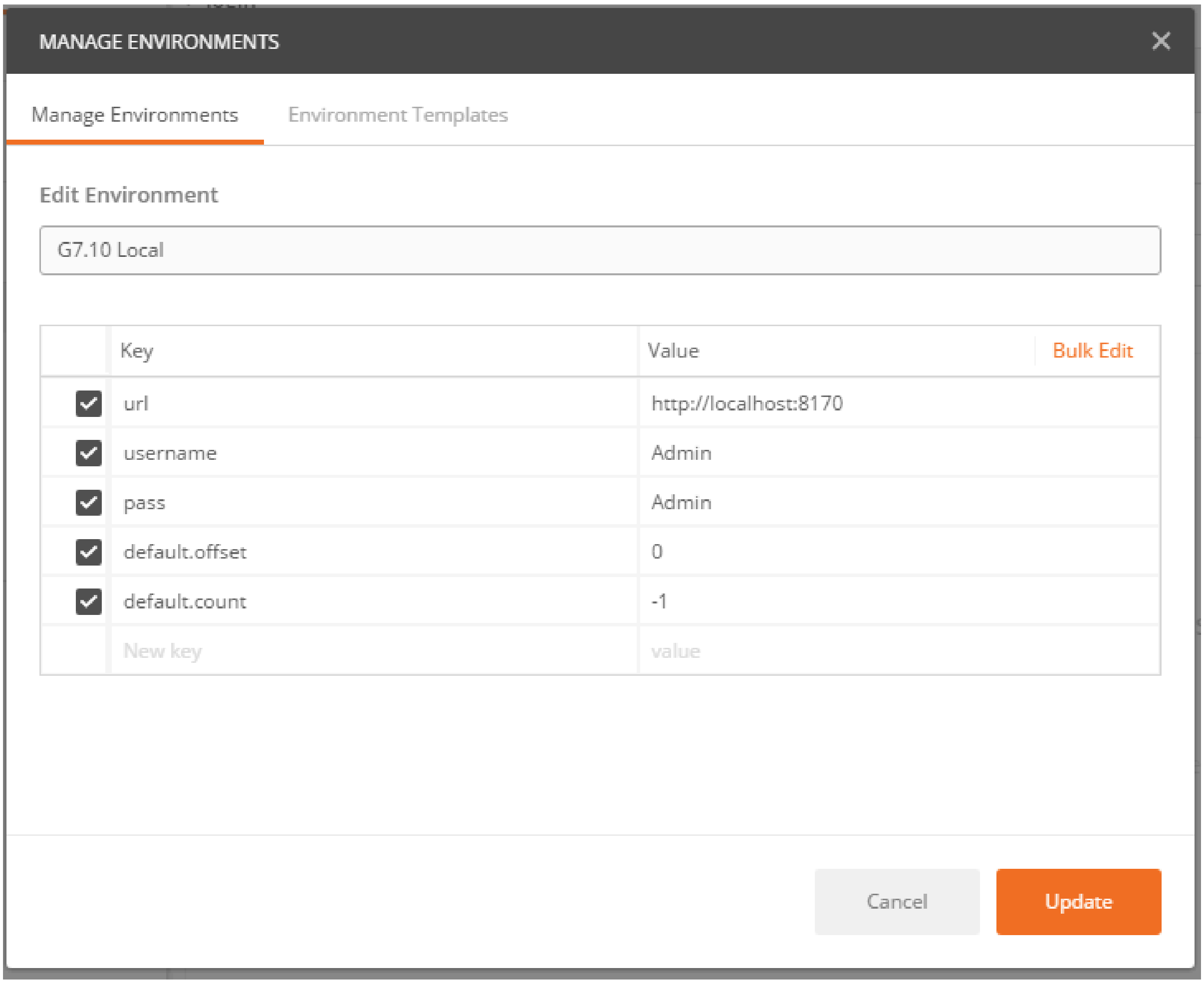
The Postman software package can be used to connect to a Gargantua server; it can be downloaded from :

https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop?hl=en



Advanced settings

The Postman environment can be setup to use parametrized requests and execute tests; this can help in automating the validation process. All our examples use a set of simple keys to indicate the server URL, user name and password.



Authentication

The autentication request does not require the conn header value; instead, it returns one.

Request URL :

```
http://domain.example.com:port/siatel-web/api/rest/auth/login
```

Request headers :

```
none
```

Request body :

```
{
    "user":"Admin",
    "pass":"Admin"
}
```

Response body :

```
{
    "result" : "ok",
    "dtoOut" : {
        "conn" : "00000070258A928000000001"
    }
}
```

At this point, a caller application should save the connection token for later use.

The connection token is always a 24 digit hexadecimal number.

ENUMERATE FIRST 2 USERS

This example retrieves the first three users available in the system.

Request URL :

```
http://domain.example.com:port/siatel-web/api/rest/admin/enumerate/user
```

Request headers :

```
"conn" : <previously saved connection token>
```

The connection token should have been saved by a previous call to /auth/login; see here for an example.

Request body :

```
{
    "flags":1080933179801481247,
    "offset":0,
    "count":2
}
```

The value used for flags in this example ("`1080933179801481247`") is the decimal equivalent of hexadecimal value "`0xF003F000000301F`", which is the numeric value of "`Flag.USER_ALL`". See the flags section for more details.

Response body :


```
{
    "result" : "ok",
    "dtoOut" : {
        "list":[
            {
                "prefix":"USER",
                "id":"USER000000000063000000000000000000000000",
                "label":"user1",
                "ownerId":"USER000000000001000000000000000000000000",
                "flags":17592186044417,
                "dateC":1477429200392,
                "dateM":1491771600246,
                "description":null,
                "realName":"test user 1",
                "privileges":251658240,
                "externalDirectoryId":"ldap://192.168.1.201:389/
CN=user1,CN=Users,dc=siatel,dc=ro",
                "dateLogin":-1,
                "dateLogout":-1,
                "profileId":"USPR000000000001000000000000000000000000",
                "workingPlannerId":null,
                "accessRestrictedByWorkingPlanner":false,
                "email":"user1@siatel.ro"
            },
            {
                "prefix":"USER",
                "id":"USER000000000061000000000000000000000000",
                "label":"user2",
                "ownerId":"USER000000000001000000000000000000000000",
                "flags":4294967296,
                "dateC":1473954415064,
                "dateM":1479732629742,
                "description":"a",
                "realName":"test user 2",
                "privileges":1152,
                "externalDirectoryId":null,
                "dateLogin":1474518837376,
                "dateLogout":-1,
                "profileId":"USPR000000000001000000000000000000000000",
                "workingPlannerId":"PLAN000000000000000000000000000000000000",
```

```
                "accessRestrictedByWorkingPlanner":false,
                "email":null
            }
        ]
    }
}
```

Retrieve user details by ID

This example retrieves the details of an existing user.

Request URL :

```
http://domain.example.com:port/siatel-web/api/rest/admin/read/user
```

Request headers :

```
"conn" : <previously saved connection token>
```

The connection token should have been saved by a previous call to /auth/login; see here for an example.

Request body :

```
{
    "flags":1080933179801481247,
    "object":{
        "prefix":"USER",
        "id":"USER00000000006300000000000000000000000"
    }
}
```

The value used for flags in this example ("1080933179801481247") is the decimal equivalent of hexadecimal value "0xF003F000000301F", which is the numeric value of "Flag.USER_ALL". See the flags section for more details.

Response body :

```
{
    "result" : "ok",
    "dtoOut" : {
        "object": {
            "prefix":"USER",
            "id":"USER00000000006300000000000000000000000",
```

```
            "label":"user1",
            "ownerId":"USER000000000001000000000000000000000000",
            "flags":17592186044417,
            "dateC":1477429200392,
            "dateM":1491771600246,
            "description":null,
            "realName":"test user 1",
            "privileges":251658240,
            "externalDirectoryId":"ldap://192.168.1.201:389/
CN=user1,CN=Users,dc=siatel,dc=ro",
            "dateLogin":-1,
            "dateLogout":-1,
            "profileId":"USPR000000000001000000000000000000000000",
            "workingPlannerId":null,
            "accessRestrictedByWorkingPlanner":false,
            "email":"user1@siatel.ro"
        }
    }
}
```

CREATE A FOLDER

This example shows how to create a folder; the folder is given a name and some attributes.

Request URL :

```
http://domain.example.com:port/siatel-web/api/rest/ged/create/folder
```

Request headers :

```
"conn" : <previously saved connection token>
```

The connection token should have been saved by a previous call to /auth/login; see [here](#) for an example.

Request body :

```
{
   "parentId": "FLDR003E001852850000000000000000000000000",
   "object": {
      "prefix": "FLDR",
      "id":"FLDR003E00000000000000000000000000000000",
      "formId": "DFFR003E0000002000000000000000000000000",
      "label": "folder created using rest api"
      "attributes": [
         {
```

```
            "prefix": "DFAT",
            "id": "DFAT003E0000006400000000000000000000000000",
            "typeId": "DFTY003E0000000100000000000000000000000000",
            "value": "987654321"
         },
         {
            "prefix": "DFAT",
            "id": "DFAT003E0000006500000000000000000000000000",
            "typeId": "DFTY003E0000000100000000000000000000000000",
            "value": "ANDERSON"
         },
         {
            "prefix": "DFAT",
            "id": "DFAT003E0000006600000000000000000000000000",
            "typeId": "DFTY003E0000000100000000000000000000000000",
            "value": "KENEATH"
         }
      ]
   }
}
```


Response body :


```
{
    "result" : "ok",
    "dtoOut" : {
        "objectId":"FLDR003E0018528700000000000000000000000000",
        "dateC":1493046208760
    }
}
```


The API will return the newly created folder's ID along with the creation timestamp.