

ALGatorServer requests and answers

Tomaž Dobravec

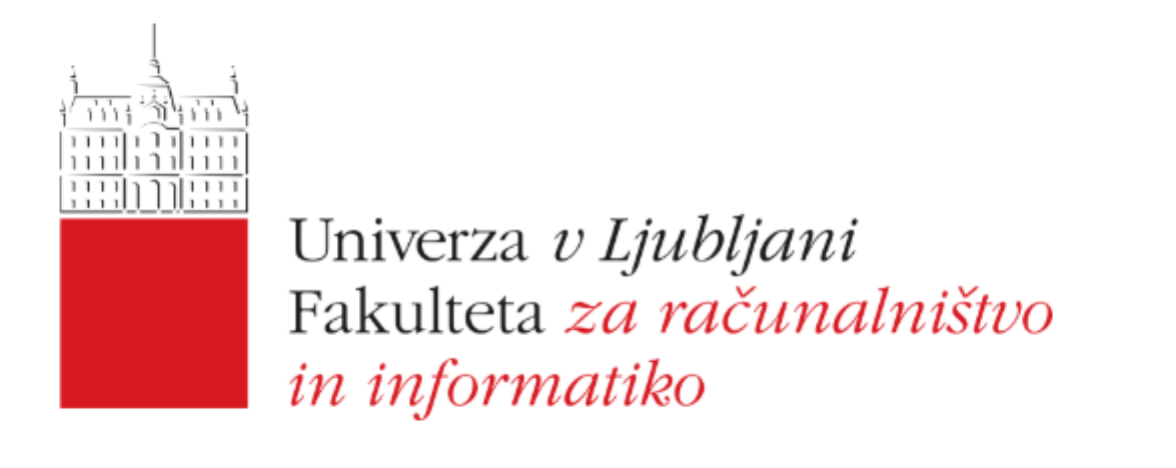


Table of content

PROJECT ADMINISTRATON 4

GETDATA 4

Type=Projects 4

Type=Project 4

Type=Algorithm 5

Type=Presenter 5

Type=ProjectSources 5

Type=ProjectProps 6

Type=ProjectDocs 6

Type=ProjectResource 6

ALTER 7

Type=NewPresenter 7

Type=RemovePresenter 7

Type=SavePresenter 7

Type=NewIndicator 7

FAMILIES and COMPUTERS 8

GETFAMILIES 8

ADDFAMILY 9

GETCOMPUTERS 10

ADDCOMPUTER 11

TASKS 12

ADDTASK 12

GETTASKS 13

GETTASK 14

TASKSTATUS 15

CLOSETASK 16

TASKRESULT 17

PAUSETASK 18

RESUMETASK 19

CANCELTASK 20

SYSTEM AND ADMINISTRATION 21

GETTIMESTAMP 21

WHO 22

STATUS 23

PRINTPATHS 24

PRINTLOG 25

ALGATOR FUNCTION 26

QUERY 26

GETQUERYRESULT 26

ADMIN 27

USERS 28

FUNCTIONS USED BY FILE EDITOR 29

GETPFILES 29

GETFILE 30

SAVEFILE 31

GETPROJECTLIST 32

FUNCTIONS USED BY RESULTS MANAGER 33

GETRESULTSTATUS 33

GETRESULTUPDATE 34

# PROJECT ADMINISTRATON

## GETDATA

Returns information about project’s entities

**Parameters**: Type, ...

**Response:**

└──> json{Status, Message, Answer: requested\_json\_data}

Type defines a type of data required. Different types require different number of parameters. Valid types are: Projects, Project, Algorithm (... more types to come)

**Examples:**

> GETDATA

└──> {"Status":**1**, "Message":"Invalid parameter. Expecting JSON with "Type" property.", "Answer": ""}

Type=Projects

> GETDATA **{"Type":"Projects"**}

└──> {"Status":**0**, "Message":"Projects", "Answer": ["BasicMatrixMul","BasicSort"]}

Type=Project

GETDATA {**"Type":"Project"**}

└──> {"Status":**1**, "Message":"getData of type=Project requires 'ProjectName' property.", "Answer": ""}

> GETDATA {**"Type":"Project", "ProjectName":"Bla"**}

└──>{"Status":**2**, "Message":"Project 'Bla' does not exist.", "Answer": ""}

> GETDATA {**"Type":"Project", "ProjectName":"BasicSort"**}

└──> {"Status":**0**, "Message":"Project 'BasicSort'.", "Answer": {

"Name": "BasicSort",

"Description": "…",

"Author": "…",

"Date": "…",

"Algorithms": [],

"TestSets": [],

"ProjectJARs": ["…"],

"ProjPresenters": [],

"AlgPresenters": [],

"MainProjPresenters": [],

"MainAlgPresenters": [] }

}

Type=Algorithm

> GETDATA {**"Type":"Algorithm", "ProjectName":"BasicSort", "AlgorithmName":"QuickSort"**}

└──> {"Status":0, "Message":"Algorithm 'QuickSort'.", "Answer": {

"Name": "QuickSort",

"ShortName": "Hoare",

"Description": "The Hoare's quicksort algorithm",

"Author": "tomaz",

"Date": "01, 2018",

"Language": "Java" }

}

Type=Presenter

> GETDATA **{"Type":"Presenter", "ProjectName":"BasicSort", "PresenterName":"mpPresenter2"}**

└──>{"Status":0, "Message":"Presenter 'mpPresenter2'.", "Answer": {

"Name": "mpPresenter2",

"Title": "…",

"ShortTitle": "TS1-Min",

"Query": {…},

"Layout": [[ "Graph\_1", "Table\_1" ]],

"Graph\_1": {…},

"Table\_1": {…}

}}

Type=ProjectSources

> GETDATA **GETDATA {"Type":"ProjectSources", ProjectName":"BasicSort"}**

└──> {"Status":0, "Message":"Project 'BasicSort' sources.", "Answer": {

"Input": "aW1wb3J0IHNpLmZyaS5...",

"Output": "aW1wb3J0IHNpLmZyaS...",

"Algorithm": "...",

"Indicators": {

"Check": "..."

},

"Generators": {

"Type0": "...",

"Type1": "..."

}

}

(all source files are base64 encoded)

Type=ProjectProps

> GETDATA **GETDATA {"Type":"ProjectProps", ProjectName":"BasicSort"}**

└──> {"Status":0, "Message":"Project 'BasicSort' sources.", "Answer": {

"Algorithms": [...],

"TestSets": [...],

"Parameters": {

"N": {

"Name": "N",

"Description": "The size …",

"Type": "int",

"Meta": {...}

}

},

"EM indicators": {

"Check": {

"Name": "Check",

"Description": "OK or NOK",

"Type": "string"

},

"Tmin": {...}

},

"CNT indicators": {...},

"JVM indicators": {}

}

Type=ProjectDocs

> GETDATA **{"Type":"ProjectDocs", ProjectName":"BasicSort"}**

└──> {"Status":0, "Message":"Project 'BasicSort' docs.", "Answer": {

{

"Project": "... base64 encoded of doc/project.html"

"Algorithm": "... base64 encoded of doc/algorithm.html"

"References": "... base64 encoded of doc/references.html"

"TestCase": "... base64 encoded of doc/testcase.html"

"TestSet": "... base64 encoded of doc/testset.html"

"Resources": {

"alg-abs.png": 1704884253027,

"other.png": <timestamp>

}

}

Type=ProjectResource

> GETDATA {"Type":"ProjectResource", "ProjectName":"BasicSort", "ResourceName":"alg-abs.png"}

└──> {"Status":0, "Message":"Document resource 'alg-abs.png'.", "Answer": "iVBORw0KGgoAAAANSUhEUgAAAVIAAAAuCAIAAAD1IAa... "

(content of resource is base64 encoded)

## ALTER

Alters (saves) information about project’s entities on the server

**Parameters**: Action, ...

**Response:**

└──> json{Status, Message, Answer: }

Type defines a type of data being saved.

**Examples:**

> ALTER

└──> {"Status":1, "Message":"Invalid parameter. Expecting JSON with "Action" property.", "Answer": ""}

Type=NewPresenter

> ALTER **{"Action":"NewPresenter", "ProjectName":…, "PresenterType":0**}

└──> {"Status":0, "Message":"Presenter created.", "Answer": "mpPresenter7"}

(presenter types: 0… main project presenter, 1 … project presenter, 2 … main algorithm presenter, 3 … algorithm presenter)

Type=RemovePresenter

> ALTER **{"Action":"RemovePresenter", ProjectName:…, PresenterName: …**}

└──> …

Type=SavePresenter

> ALTER **{"Action":"SavePresenter", ProjectName:BasicSort, PresenterName: mpPresenter3, PresenterData: { "Name": "mpPresenter3", "Title":** **"…", "ShortTitle": "…", "Description": "…", "Query": {…}, "Layout": [[ "Graph\_1", "Table\_1" ]], "Graph\_1": {…}, "Table\_1": {…} }**}

└──> …

Type=NewIndicator

> **alter {Action:NewIndicator}**

└──> {"Status":1, "Message":"Alter of type=NewIndicator requires 'ProjectName', 'IndicatorName' and 'IndicatorType' (optional, default='indicator') properties.", "Answer": ""}

> **alter {Action:NewIndicator, ProjectName:BasicSort, IndicatorName:Check, IndicatorType:indicator}**

└──> {"Status":0, "Message":"Indicator added.", "Answer": base64\_encoded\_indicatorTest\_Check.java\_file}

(Type: **"**indicator**" (default)**, **"**timer**"**, **"**counter**"**)

# FAMILIES and COMPUTERS

## GETFAMILIES

Returns all the computer families defined in the system.

**Parameters**: /

**Response:**

└──> json{Status, Message, Answer:[{FamilyID, Description,

Platform, Hardware}, ...]}

**Example:**

> GETFAMILIES

└──>

{

"Status":0,

"Message":"Computer families",

"Answer":[

{

"FamilyID":"F1",

"Description":"Computer used to run preliminary tests",

"Platform":"Windows 11, x64",

"Hardware":"Intel i5, 3.2GHz, 32GB RAM"

},

{

"FamilyID":"F2",

"Description":"Computers used to run algorithms",

"Platform":"Ubuntu 16.04, x64",

"Hardware":"i7, 2.8Ghz, 32GB RAM"

}

]

}

## ADDFAMILY

Add a new family to the system

**Parameters:** json{Name, FamilyID, Description, Platform,

Hardware}

**Response1:** if FamilyID contains space character

└──> json{Status:1, Message: "Error - invalid character in FamilyID

", answer: " "}

**Response2:** if FamilyID already exists

└──> json{Status:1, Message: " Error - family with this ID already

exists. ", answer: " "}

**Response3:** if family is successfully added

└──> json{Status:0, Message: "Family added", answer: "FamilyID"}

**Example:**

> ADDFAMILY {"Name":"Family-F9", "FamilyID":"F9", "Description": "Family of fast computers", "Platform":"Ubuntu 20.04, x64", "Hardware":"i9, 64GB"}

└──> {'Status':0, 'Message':'Family added', 'Answer': 'F9'}

## GETCOMPUTERS

Returns all the computers in a given family (or in all families if no FamilyID is given)

**Parameters:** no parameters or json{FamilyID}

**Response:**

└──> json{Status, Message, Answer:[{ UID, ComputerID, FamilyID, Description, Capabilities}, ...]}

**Example:**

> getcomputers {"FamilyID":"F1"}

└──>

{

"Status":0,

"Message":"Computers",

"Answer":[

{

"UID":"6ght6zfhjre6",

"ComputerID":"C1",

"FamilyID":"F1",

"Description":"My notebook",

"Capabilities":[

"EM",

"CNT"

]

},

{

"UID":"7gf64jh3jre7",

"ComputerID":"C2",

"FamilyID":"F1",

"Description":"PC",

"Capabilities":[

"EM",

"CNT",

"JVM"

]

}

]

}

## ADDCOMPUTER

Add a new computer to a given family

**Parameters:** json{Name, FamilyID, ComputerID, Description, IP, Capabilities}

**Response1:** if ComputerID contains space character

└──> json{Status:1, Message: "Error - invalid character in

ComputerID", answer: " "}

**Response2:** if ComputerID already exists (in this family)

└──> json{Status:1, Message: " Error - computer with this ID already

exists.", answer: " "}

**Response3:** if computer is successfully added

└──> json{Status:0, Message: "Computer added",

answer: {"ComputerID":"C2","UID":"jEzm2Ws7uX"}}

**Example:**

> ADDCOMPUTER {"Name": "Computer-C1"," FamilyID":"F9", "ComputerID": "C2", "Description":"My Copm", "IP":"1.2.3.4", "Capabilities": ["em","jvm"]}

└──> {'Status':0, 'Message':'Computer added', 'Answer': {"ComputerID" :"C2", "UID":"jEzm2Ws7uX"}}

# TASKS

## ADDTASK

Add a new task to a task queue.

(Note: tasks in queue are ordered by Status (QUEUED, INPROGRESS, OTHERS), Priority (higher priority first) and Date (the oldest first).

**Parameters:** json(Project(required), Algorithm(required), Testset(required), Mtype (default:em), Priority (default: 5), Family (default: ""))

**Response1:** if parameter is not json

└──> {‘Status’:1, ‘Message’: "Invalid parameter. Expecting JSON with "Project", "Algorithm" and "Testset" properties.

"}

**Response2: on error**

{'Status':2, 'Message':'Invalid task. Project '…' does not exist.', 'Answer': ''}

**Response3:** if task is successfully added

└──> {'Status':0, 'Message':'Task added', 'Answer': TaskID}

**Example:**

>addtask {"Project":"BasicSort", "Algorithm":"QuickSort", "Testset":"TestSet2", "MType":"cnt"}

└──> {'Status':0, 'Message':'Task added', 'Answer': 610}

## GETTASKS

Get a list of all tasks in a selected task-list (active, closed, archived).

**Parameters:** json(Type(optional))

(possible types: “active” (default), “closed”, “archived”)

**Response1:** if type is not valid

└──> {'Status':1, 'Message':'Invalid type, 'active', 'closed' or 'archived' expected.', 'Answer': ''}

**Response2:** if OK

└──> list of tasks

**Example:**

>gettasks {"Type":"closed"}

└──> [{"Name":"Task-708","TaskID":708,"Project":"g","Algorithm":"x","Testset":"t","MType":"em","Family":"","Status":"FAILED","Progress":0,"StatusDate":1671616249724,"ComputerUID":"ly6XSsuH2T","Priority":5,"Msg":"Execution failed, : Invalid project name."},

{"Name":"Task-709","TaskID":709,"Project":"P1","Algorithm":"a1","Testset":"t2","MType":"em","Family":"","Status":"FAILED","Progress":0,"StatusDate":1671616253426,"ComputerUID":"ly6XSsuH2T","Priority":5,"Msg":"Execution failed, : Invalid project name."},

{"Name":"Task-710","TaskID":710,"Project":"P2","Algorithm":"a1","Testset":"t2","MType":"em","Family":"","Status":"FAILED","Progress":0,"StatusDate":1671616256079,"ComputerUID":"ly6XSsuH2T","Priority":5,"Msg":"Execution failed, : Invalid project name."}]

## GETTASK

Get the most appropriate task for a given computer

(Note: task t is approptiate for computer c if

(t.family=empty || task.family == c.family) AND

(c.capabilities.contains(t.mType)

)

**Parameters:** json(ComputerUID)

**Response1:** if parameter is not json

└──> {'Status':1, 'Message':'Invalid number or type of parameters', 'Answer': 'Expecting JSON with "ComputerUID" property.'}

**Response2:** if wrong ComputerID is given or computer is malconfigured (wrong or missing family)

└──> {'Status':2, 'Message':'NO\_TASKS', 'Answer': 'Invalid computer or computer family'}

**Response3:** if not tasks are available for this computer

└──> {'Status':3, 'Message':'NO\_TASKS', 'Answer': 'No tasks available for this computer.'}

**Response4:** if everything is OK and task exists

└──> {'Status':0, 'Message':'Task for computer uid, 'Answer': STask json}

**Example:**

> gettask {ComputerUID:kh16D5FWID}

└──> {'Status':0, 'Message':'Task for computer kh16D5FWID', 'Answer': {"Name":"Task-660","TaskID":660,"Project":"PP1","Algorithm":"AA4","Testset":"TS1","MType":"em","Family":"","Status":"INPROGRESS","Progress":0,"StatusDate":1671524117553,"ComputerUID":"kh16D5FWID","Priority":7}}

## TASKSTATUS

Returns task status (json with all task’s properties).

Status of both, active and closed tasks, is available via this call.

**Parameters:** json(TaskID)

**Response1:** if parameter is not json

└──> {'Status':1, 'Message':'Invalid number or type of parameters', 'Answer': 'Expecting JSON with "TaskID" property.'}

**Response2: if task does not exist**

└──> {'Status':2, 'Message':'Unknown task.', 'Answer': 67}

(note: answer == TaskID

**Response3: if everithing is OK**

└──> json with task properties

**Example:**

> taskstatus {"TaskID":700}

└──> {"Name":"Task-700","TaskID":700,"Project":"BasicSort","Algorithm":"QuickSort","Testset":"Testsečšžt","MType":"em","Family":"","Status":"FAILED","Progress":0,"StatusDate":1671545280786,"ComputerUID":"ly6XSsuH2T","Priority":1,"Msg":"Execution failed, : Invalid testset name."}

## CLOSETASK

Cloese the task (i.e. moves task from taskQueue oldTaskQueue and change status of the task)

This request is used by TaskClient to tell ALGatorServer that the task has been closed (due to the error or because all tests were performed)

**Parameters:** json(ExitCode, TaskID, Message)

If ExitCode==0, task was completed (all tests were performed) otherwise an error with ExitCode and Message description occured.

**Response1:** if parameter is not json

└──> {'Status':1, 'Message':'Invalid number or type of parameters', 'Answer': 'Expecting JSON with "ExitCode" and "TaskID" properties.'}

**Response2:** if unknown task is given

└──> Task not found.

**Response3:** if everything is OK

└──> OK

**Example:**

**>pauseTask {TaskID:920}**

> closetask {ExitCode: 0, TaskID:700, Message:"Done."}

└──> OK

## TASKRESULT

Request is called by TaskClient when one test is finished; client sends taskresult request (with json result of this test) and recieves the answer: CONTINUE or something else. If answer is CONTINUE, client continues with the next test otherwise it gives control to runClient() method to ask for another task.

**Parameters:** json(ComputerUID, TaskID, TestNo, Result)

**Responses:**

└──> {'Status':0, 'Message':'Result accepted, continue.', 'Answer': 'CONTINUE'}

└──> {'Status':0, 'Message':'Result accepted, task canceled | paused.', 'Answer': 'BREAK'}

└──> {'Status':0, 'Message':'Result accepted, task queued.', 'Answer': 'QUEUED'}

└──> {'Status':1, 'Message':'Invalid number or type of parameters', 'Answer': 'Expecting JSON with "ComputerUID", "TaskID", "TestNo" and "Result" properties.'}

└──> {'Status':2, 'Message':'Invalid task number', 'Answer': tid}

└──> {'Status':3, 'Message':'Task does not belong to this computer', 'Answer': cUID}

Note: all answers except the one with Status:0 and Answer: CONTINUE are signals to TaskClient to abandon the current job.

## PAUSETASK

Pauses the execution of a task

**Parameters:** json(TaskID)

**Example:**

>pauseTask {TaskID:914}

**Responses:**

└──> {'Status':0, 'Message':'OK', 'Answer': 'Status changed'}

└──> {'Status':1, 'Message':'Invalid number or type of parameters', 'Answer': 'Expecting JSON with "TaskID" property.'}

└──> {'Status':2, 'Message':'Task not found.', 'Answer': 914}

## RESUMETASK

Resumes the execution of a task.

**Parameters:** json(TaskID)

**Example:**

>resumeTask {TaskID:914}

**Responses:**

└──> {'Status':0, 'Message':'OK', 'Answer': 'Status changed'}

└──> {'Status':1, 'Message':'Invalid number or type of parameters', 'Answer': 'Expecting JSON with "TaskID" property.'}

└──> {'Status':2, 'Message':'Task not found.', 'Answer': 914}

└──> {'Status':3, 'Message':'Can not resume - task is not paused.', 'Answer': 914}

## CANCELTASK

Cancel (delete) the task.

**Parameters:** json(TaskID)

**Example:**

>cancelTask {TaskID:914}

**Responses:**

└──> {'Status':0, 'Message':'OK', 'Answer': 'Status changed'}

└──> {'Status':1, 'Message':'Invalid number or type of parameters', 'Answer': 'Expecting JSON with "TaskID" property.'}

└──> {'Status':2, 'Message':'Task not found.', 'Answer': 914}

# SYSTEM AND ADMINISTRATION

## GETTIMESTAMP

Returns the current timestamp of the server.

This is used to label tasks on clients (so that they have servers timestamp and can therefore be compared eventhough they were maintained by different clients).

**Parameters:** no parameters required

**Response1:**

└──> {'Status':0, 'Message':'Tmestamp', 'Answer': timestamp}

**Example:**

>gettimestamp

└──> {'Status':0, 'Message':'Tmestamp', 'Answer': 1671525005083}

## WHO

Returns number of requests this ALGatorServer has processed so far.

**Parameters:** no parameters required

**Response1:**

└──> number\_of\_processed\_requests

**Example:**

>who

└──> 23

## STATUS

Returns status information of ALGatorServer

**Parameters:** no parameters required

**Response1:**

└──> status\_string

**Example:**

>status

└──> Server at 192.168.6.104 on for 26 min, 55 sec. Processed requests: 484. Tasks: 0 running, 1 pending, 0 queued, 0 paused

## PRINTPATHS

Returns ALGATOR\_ROOT and DATA\_ROOT used by this server

**Parameters:** no parameters required

**Response1:**

└──> ALGATOR\_ROOT=…, DATA\_ROOT=…

**Example:**

> PRINTPATHS

└──> ALGATOR\_ROOT=D:\Users\tomaz\OneDrive\ULFRI\ALGATOR\_ROOT, DATA\_ROOT=D:\Users\tomaz\OneDrive\ULFRI\ALGATOR\_ROOT\data\_root

## PRINTLOG

Returns last n lines from ALGatorServer log file

**Parameters:** optional parameter n (default=10)

**Response1:**

└──> n lines of txt file with log content,

**Example:**

> PRINTLOG 5

└──>

Content of 'D:\Users\tomaz\OneDrive\ULFRI\ALGATOR\_ROOT\data\_root\log\server\server.log' (last 5 lines) :

2023/11/22 12:48:24: [REQUEST]: status

2023/11/22 12:48:24: [REQUEST]: LIST

2023/11/22 12:48:24: [RESPONSE]: ERROR:: Unknown request

2023/11/22 12:48:24: [RESPONSE]: Server at 192.168.6.104 on for 2 min, 1 sec. Processed requests: 10. Tasks: 0 running, 1 pending, 0 queued, 0 paused;

2023/11/22 12:48:50: [REQUEST]: PRINTLOG 5

# ALGATOR FUNCTION

## QUERY

Runs a query and returns a result.

**Parameters:** json (ProjectName, Query, ComputerID (optional), Parameters (optional))

**Response1:**

└──> {'Status':0, 'Message': "Query result", 'Answer': ' CSV

table with query response data '}

**Example:**

> query {"**ProjectName**":"BasicSort", "**Query**":{"Algorithms":["\*"], "TestSets":["$0"]},"**Parameters**":["TestSet2"]}

**Responses:**

└──> {"Status":0, "Message":"Query result", "Answer": "ID;Testset;InstanceID;Pass;N 1;TestSet1;Test-1;DONE;100 2;TestSet1;Test-2;DONE;100 3;TestSet1;Test-3;DONE;… "}

## GETQUERYRESULT

Runs a query and returns a result. (old style function; depricated)

**Parameters:** project\_name query\_name [optional query params]

**Response1:**

└──> CSV table with query response data

**Example:**

>GETQUERYRESULT BasicSort query2 QuickSort

└──> ID;Testset;InstanceID;Pass;N;BubbleSort.Tfirst;InsertionSort.Tfirst;JavaSort.Tfirst;QuickSort.Tfirst

1;TestSet2;test1;DONE;10000;191334;32755;921;1690;187114

6;TestSet2;test6;DONE;11000;232865;35494;852;1758;231232;

…

## ADMIN

Runs algator.Admin with given parameters in the ALGatorServers’ thread (not as separated process).

**Parameters:** algator.Admin parameters (-i, -cp Project, ...)

**Response1:**

└──> usual algator.Admin response

**Example:**

>admin -i

└──>

{"Projects": [

"BasicMatrixMul",

"BasicSort"

]}

## USERS

Runs algator.Users with given parameters in the ALGatorServers’ thread (not as separated process).

**Parameters:** algator.Users parameters (-u, -p, request, ...)

**Response1:**

└──> usual algator.Users, response

**Example:**

>users -u tomaz -p ptomaz canuser someone can\_read BasicSort

└──> true

# FUNCTIONS USED BY FILE EDITOR

## GETPFILES

Returns a html list of files of a project

**Parameters:** json(Project)

**Response1:** if parameter is not json

└──> {'Status':1, 'Message':'Invalid number or type of parameters', 'Answer': 'Expecting JSON with "Project" property.'}

**Response2:** if everything is OK

└──> {'Status':0, 'Message':'Project files', 'Answer': 'base64\_encoded\_list\_of\_projects'\_files}

**Example:**

>getpfiles {"Project":"BasicSort"}

└──>{'Status':0, 'Message':'Project files', 'Answer': 'PHVsIGlk PSJ0cmVlVUwiPgog…'}

## GETFILE

Returns a content of a file.

**Parameters:** json(Project, File)

**Response1:** if parameter is not json

└──> {'Status':1, 'Message':'Invalid number or type of parameters', 'Answer': 'Expecting JSON with "Project" and "File" properties.'}

**Response2:** if everything is OK

└──> {'Status':0, 'Message':'File content', 'Answer': 'base64\_encoded\_file\_content'}

**Example:**

> getfile {"Project":"BasicSort", "File": "proj/BasicSort.atp"}

└──>   
{'Status':0, 'Message':'File content', 'Answer': 'eyJQcm9qZWN0IjogewogICJOYW1lIjogIkJhc2ljU….'}

## SAVEFILE

Saves a content into a text file.

**Parameters:** json(Project, File, Content, Length)

Note: Content has to be base64 encripted, Length in length of encripted content.

**Response1:** if parameter is not json

└──> {'Status':1, 'Message':'Invalid number or type of parameters', 'Answer': 'Expecting JSON with "Project", "File", "Content" and "Length" properties.'}

**Response2:** if length mismatch

└──> {'Status':2, 'Message':'File save error', 'Answer': 'File content length mismatch'}

**Response3:** if everything is OK

└──> {'Status':0, 'Message':'OK', 'Answer': 'File saved.'}

**Example:**

> savefile {"Project":"BasicSort", "File":"test.text", "Content":"eyJQcm9qZWN0", "Length":12}

└──> {'Status':0, 'Message':'OK', 'Answer': 'File saved.'}

## GETPROJECTLIST

Get a list of all projects in DATA\_ROOT.

**Parameters:** no parameters

**Response:**

└──> {'Status':0, 'Message':'Project list', 'Answer': 'base64\_encripted\_list\_of\_projects'}

**Example:**

> getprojectslist

└──> {'Status':0, 'Message':'Project list', 'Answer': 'eyJQcm9qZWN0cyI6WyJCYXNpY01hdHJpeE11bCIsIkJ…'}

**Example of decripted Answer:**

{"Projects":["BasicMatrixMul","BasicSort"]}

# FUNCTIONS USED BY RESULTS MANAGER

These functions are used to support ALGatorShell/Project/Results functionality.

## GETRESULTSTATUS

Returns array of result statuses.

**Parameters:** json(Project, MType (optional))

**Response1:** if parameter is not json

└──>

|  |  |
| --- | --- |
| {'Status':1, 'Message':'Invalid number or type of parameters', 'Answer': 'Expecting JSON with "Project" and "MType" (optional) property.'} | ▲ |
|  |  |

**Response2:** if everything is OK

└──> {'Status':0, 'Message':' Result status', 'Answer': 'base64\_encoded\_result\_status'}

**Example:**

> >getresultstatus {"Project": "BasicSort"}

└──>  
{'Status':0, 'Message':'Result status', 'Answer': 'eyJSZXbHR...'}

**Part of decripted Answer:**

{"Results":[{"Algorithm":"Alg42","TestSet":"TestSet0","MType":"em","Status":{"F7":[{"Fmy":"F7","Cmp":"C1","CID":"","RS":0,"FS":"(0/10)","TS":"","TID":0}],"F0":[{"Fmy":"F0","Cmp":"C0","CID":"Gs9ayiuRYx","RS":0,"FS":"(0/10)","TS":"","TID":0},{"Fmy":"F0","Cmp":"F1","CID":"","RS":0,"FS":"(0/10)","TS":"","TID":0}],"F1":[{"Fmy":"F1","Cmp":"C0","CID":"TnrM5PG4R5","RS":0,"FS":"(0/10)","TS":"","TID":0},{"Fmy":"F1","Cmp":"C4","CID":"","RS":0,"FS":"(0/10)","TS":"","TID":0}],"F2":[{"Fmy":"F2","Cmp":"C2","CID":"",...

## GETRESULTUPDATE

Returns a status of a given result string (changed (minor/major)) or not).

FUnction is used to properly show

**Parameters:** json(ID)

**Response1:** if parameter is not json

└──> {'Status':1, 'Message':'Invalid number or type of parameters', 'Answer': 'Expecting JSON with "ID" property.'}

**Response2:** if task id is invalid

└──> {'Status':2, 'Message':'No results', 'Answer': 'Results for this id do not exist.'}

**Example:**

> getresultupdate {"id": "1149"}

└──>{'Status':0, 'Message':'Minor/Major', 'Answer': 'Status'}