Polygon API Documentation

How to download a problem or contest

Use HTTP POST-requests:

- Use problem URL and POST-parameters <login>, <password> and optionals <revision>, <type> to download a zip-file with problem package (you can use type=windows or type=linux). If the problem has the pin code, provide an optional POST-parameter <pin>. Example:

```
wget --post-data=login=vtapochkin\&password=??? \
https://polygon.codeforces.com/p85dIBF/mmirzayanov/a-plus-b
```

- Use problem URL + /problem.xml and POST-parameters <login>, <password> and optional <revision> to download a problem descriptor (much more faster, use it to check if new revision has been released). If the problem has the pin code, provide an optional POST-parameter <pin>. Example:

```
wget --post-data=login=vtapochkin\&password=??? \
https://polygon.codeforces.com/p85dIBF/mmirzayanov/a-plus-b/problem.xml
```

- Use ‘`c/ContestUID/contest.xml`’ and POST-parameters <login>, <password> to download a contest descriptor. If the contest has the pin code, provide an optional POST-parameter <pin>. Example:

```
wget --post-data=login=vtapochkin\&password=??? \
https://polygon.codeforces.com/c/50431a121273b7e31f4200e7/contest.xml
```

Introduction

To access Polygon API you just send a HTTP-request to address https://polygon.codeforces.com/api/{methodName} with method-specific parameters.

Most method calls return a JSON-object (if not specified otherwise) with three possible fields: status, comment and result.

- Status is either "OK" or "FAILED".
- If status is "FAILED" then comment contains the reason why the request failed. If status is "OK", then there is no comment.
- If status is "OK" then result contains method-dependent JSON-element which will be described for each method separately. If status is "FAILED", then there is no result. In

Authorization

Using Polygon API requires authorization. To authorize, you will need an API key, which can be generated on settings page. Each API key has two parameters: *key* and *secret*. To use the key you must add following parameters to your request.

1. *apiKey* - must be equal to *key*.
2. *time* — current time in UNIX format (e.g., System.currentTimeMillis()/1000). If the difference between server time and time, specified in parameter, will be greater than 5 minutes, request will be denied.
3. *apiSig* — signature to ensure that you know both *key* and *secret*. The first six characters of the *apiSig* parameter can be arbitrary. We recommend to choose them at random for each request. Let's denote them as *rand*. The rest of the parameter is hexadecimal representation of SHA-512 hash-code of the following string: *<rand>/<methodName>?param1=value1&param2=value2...&paramN=valueN#<secret>* where *(param_1, value_1), (param_2, value_2), ..., (param_n, value_n)* are all the request parameters (including *apiKey*, *time*, but excluding *apiSig*) with corresponding values, sorted lexicographically first by *param_i*, then by *value_i*.

Methods

problems.list

Returns a list of problems, available to the user, according to search parameters.

Parameters:

showDeleted - *bool, optional* - show/hide deleted problems (defaults to *false*)
id - problem id*, optional*
name - problem name*, optional*
owner - problem owner login*, optional*

Returns:

A list of *Problem* objects.

problem.create

Create a new empty problem. Returns a created Problem.

Parameters:

name - name of problem being created

Methods for problems

To access problem-specific API methods, you need to enter the problem by adding a *problemId* parameter to your request. For now, you need to have at least *WRITE* access for the problem to use the api. If the problem has the pin code, you need to add the parameter *pin* to your request.

problem.info

Returns a ProblemInfo object.

problem.updateInfo

Update problem info. All parameters are optional.

Parameters:

inputFile - problem's input file
outputFile - problem's output file
interactive - *boolean* - is problem interactive
timeLimit - problem's time limit in milliseconds
memoryLimit - problem's memory limit in MB

problem.updateWorkingCopy

Updates working copy.

problem.discardWorkingCopy

Discards working copy.

problem.commitChanges

Commits problem changes. All parameters are optional.

Parameters:

minorChanges - *boolean* - if *true,* no email notification will be sent
message - problem's commit message

problem.statements

Returns a map from language to a Statement object for that language.

problem.saveStatement

Update or create a problem's statement. All parameters except for *lang* are optional.

Parameters:

lang (required) - statement's language
encoding - statement's encoding
name - problem's name in statement's language
legend - problem's language
input - problem's input format
output - problem's output format
scoring - problem's scoring
interaction - problem's interaction protocol (only for interactive problems)
notes - statement notes
tutorial - problem's tutorial

problem.statementResources

Returns a list of statement resources for the problem.

Parameters:

None

Returns:

A list of *File* objects.

problem.saveStatementResource

Add or edit statement resource file.

Parameters:

checkExisting - boolean, optional - if *true*, only adding files is allowed
name - file name
file - file content

problem.checker

Returns the name of the currently set checker.

problem.validator

Returns the name of the currently set validator.

problem.extraValidators

Returns an array containing the names of the currently set extra validators.

problem.interactor

Returns the name of the currently set interactor.

problem.validatorTests

Returns a list of validator tests for the problem.

Parameters:

None

Returns:

A list of *ValidatorTest* objects.

problem.saveValidatorTest

Add or edit validator test. In case of editing, all parameters except for *testIndex* are optional. In case of adding, all parameters except for *testIndex*, *testInput* and *testVerdict* are optional.

Parameters:

checkExisting - boolean, optional - if *true*, only adding validator's test is allowed
testVerdict - VALID/INVALID - validator test verdict
testIndex - index of a validator test
testInput - input of a validator test
testGroup - *optional* - test group (groups should be enabled for the testset)
testset - optional - testset name

problem.checkerTests

Returns a list of checker tests for the problem.

Parameters:

None

Returns:

A list of *CheckerTest* objects.

problem.saveCheckerTest

Adds or edits checker test. In case of editing, all parameters except for *testIndex* are optional. In case of adding, all parameters except for *testIndex*, *testInput*, *testAnswer*, *testOutput* and *testVerdict* are optional.

Parameters:

checkExisting - boolean, optional - if *true*, only adding checker test is allowed

testVerdict - OK/WRONG_ANSWER/PRESENTATION_ERROR/CRASHED - checker's test verdict
testIndex - index of a checker test
testInput - input of a checker test
testOutput - output of a checker test
testAnswer - answer of a checker test

problem.files

Returns the list of resource, source and aux files. Method returns a JSON object with three fields: *resourceFiles, sourceFiles* and *auxFiles*, each of them is a list of *File* objects.

problem.solutions

Returns the list of *Solution* objects.

problem.viewFile

Returns resource, source or aux file. In case of success, this method does not return JSON, instead it returns plain view of the file with the corresponding mime-type set.

Parameters:

type - *resource/aux/source* - requested file's type
name - file name

problem.viewSolution

Returns solution file. In case of success, this method does not return JSON, instead in returns plain view of the solution file.

Parameters:

name - solution's name

problem.script

Returns script for generating tests. In case of success, this method does not return JSON, instead it returns plain view of the script.

Parameters:

testset - testset for which the script is requested

problem.tests

Returns tests for the given testset

Parameters:

testset - testset for which tests are requested *noInputs - boolean, optional* - if *true*, returns tests without input

Returns:

A list of *Test* objects.

problem.testInput

Returns generated test input. In case of success, this method does not return JSON, instead it returns plain view.

Parameters:

testset - testset of the test
testIndex - index of the test

problem.testAnswer

Returns generated test answer. In case of success, this method does not return JSON, instead it returns plain view.

Parameters:

testset - testset of the test
testIndex - index of the test

problem.setValidator

Update validator.

Parameters:

validator - name of the validator (one of source files)

problem.setChecker

Update checker.

Parameters:

checker - name of the checker (one of source files)

problem.setInteractor

Update interactor.

Parameters:

interactor - name of the interactor (one of source files)

problem.saveFile

Add or edit resource, source or aux file. In case of editing, all parameters except for *type* and *name* are optional.

Parameters:

checkExisting - boolean, optional - if *true*, only adding files is allowed
type - file type (*resource*/*source* or *aux*)
name - file name
file - file content
sourceType - *optional* - source type (in case of a source file)
In case of type=resource there some possible additional parameters:
forTypes - *optional* - semicolon separated list of applicable file types (see ResourceAdvancedProperties)
stages - *optional* - semicolon separated list of values COMPILE or RUN, meaning the phase when the resource is applicable (currently only *stages*=COMPILE is supported)
assets - *optional* - semicolon separated list of values VALIDATOR, INTERACTOR, CHECKER, SOLUTION, meaning the asset types the resource is applicable (currently only *assets*=SOLUTION is supported)

The parameters *forTypes, stages, assets* can be present only together (it means, all of them are absent or all of them are used at the same time). They can be used only for *type=resource.*

To delete ResourceAdvancedProperties of a resource file pass empty "*forTypes=*".

problem.saveSolution

Add or edit solution. In case of editing, all parameters except for *name* are optional.

Parameters:

checkExisting - boolean, optional - if *true*, only adding solutions is allowed
name - solution name
file - solution content
sourceType - *optional* - source type
tag - solution's tag (*MA* - Main, *OK*, *RJ*, *TL*, *TO* - Time Limit or Accepted, *WA*, *PE*, *ML* or *RE*)

problem.editSolutionExtraTags

Add or remove testset or test group extra tag for solution.

Parameters:

remove - boolean, if *true -* remove extra tag, if *false -* add extra tag.
name - solution name
testset - optional - testset name for which edit extra tag

testGroup - optional - test group name for which edit extra tag
Exactly one from *testset* and *testGroup* should be specified
tag - *optional -* when you add extra tag - solution's extra tag (*OK*, *RJ*, *TL*, *TO* - Time Limit or Accepted, *WA*, *PE*, *ML* or *RE*)

problem.saveScript

Edit script.

Parameters:

testset - testset of the script
source - script source

problem.saveTest

Add or edit test. In case of editing, all parameters except for *testset* and *testIndex* are optional.

Parameters:

checkExisting - boolean, optional - if *true*, only adding new test is allowed
testset - testset of the test
testIndex - index of the test
testInput - test input
testGroup - *optional* - test group (groups should be enabled for the testset)
testPoints - *optional* - test points (points should be enabled for the problem)
testDescription - optional - test description
testUseInStatements - bool, optional - whether to use test in statements
testInputForStatements - optional - test input for viewing in the statements
testOutputForStatements - optional - test output for viewing in the statements
verifyInputOutputForStatements - bool, optional - whether to verify input and output for statements

problem.setTestGroup

Set test group for one or more tests. It expects that for specified testset test groups are enabled.

Parameters:

testset - testset of the test(s)
testGroup - test group name to set
testIndex - index of the test, you can specify multiple parameters with the same name *testIndex* to set test group to many tests of the same testset at the same time.
testIndices - list of test indices, separated by a comma. It's alternative for *testIndex,* you should use only one from these two ways

problem.enableGroups

Enable or disable test groups for the specified testset.

Parameters:

testset - testset to enable or disable groups
enable - bool - if it is true test groups become enabled, else test groups become disabled

problem.enablePoints

Enable or disable test points for the problem.

Parameters:

enable - bool - if it is true test points become enabled, else test points become disabled

problem.viewTestGroup

Returns test groups for the specified testset.

Parameters:

testset - testset name
group - optional - test group name

Returns:

A list of *TestGroup* objects.

problem.saveTestGroup

Saves test group. Use if only to save a test group. If you want to create new test group, just add new test with such test group.

Parameters:

testset - testset name
group - test group name
pointsPolicy - *optional* - COMPLETE_GROUP or EACH_TEST (leaves old value if no specified)
feedbackPolicy - *optional* - NONE, POINTS, ICPC or COMPLETE (leaves old value if no specified)
dependencies - optional - string of group names from which group should depends on separated by a comma

problem.viewTags

Returns tags for the problem.

Parameters:

None

Returns:

A list of strings – tags for the problem.

problem.saveTags

Saves tags for the problem. Existed tags will be replaced by new tags.

Parameters:

tags – string of tags, separated by a comma. If you specified several same tags will be add only one of them.

problem.viewGeneralDescription

Returns problem general description.

Parameters:

None

Returns:

A string – the problem general description.

problem.saveGeneralDescription

Saves problem general description.

Parameters:

description – *string* – the problem general description to save. The description may be empty.

problem.viewGeneralTutorial

Returns problem general tutorial.

Parameters:

None

Returns:

A string – the problem general tutorial.

problem.saveGeneralTutorial

Saves problem general tutorial.

Parameters:

tutorial – *string* – the problem general tutorial to save. The tutorial may be empty.

problem.packages

Returns a list of *Package* objects - list all packages available for the problem.

Parameters:

None

problem.package

Download a package as a zip-archive.

Parameters:

packageId - package id
type – string, optional – type of the package: “standard”, “linux” or “windows”. Standard packages don't contain generated tests, but contain windows executables and scripts for windows and linux via wine. Linux packages contain generated tests, but don't contain compiled binaries. Windows packages contain generated tests and compiled binaries for Windows. If not provided, “standard” is used.

problem.buildPackage

Starts to build a new *Package*.

Parameters:

full - bool - defines whether to build full package, contains “standard”, “linux” and “windows” packages, or only “standard”
Standard packages don't contain generated tests, but contain windows executables and scripts for windows and linux via wine. Linux packages contain generated tests, but don't contain compiled binaries. Windows packages contain generated tests and compiled binaries for Windows.
verify - bool - if that parameter is true all solutions will be invoked on all tests to be sure that tags are valid. Stress tests will run the checker to verify its credibility.

Contest methods

To access contest methods, you have to include *contestId* parameter in your request. If the contest has the pin code, you need to include the parameter *pin* in your request.

contest.problems

Returns a list of *Problem* objects - problems of the contest.

Return objects

Problem

Represents a polygon problem.
id - problem id
owner - problem owner handle
name - problem name
deleted - boolean - is a problem deleted
favourite - boolean - is a problem in user's favourites
accessType - *READ/WRITE/OWNER* - user's access type for the problem
revision - current problem revision
latestPackage - latest revision with package available (may be absent)
modified - is the problem modified

ProblemInfo

Represents the problem's general information.
inputFile - problem's input file
outputFile - problem's output file
interactive - *boolean* - is problem interactive
timeLimit - problem's time limit in milliseconds
memoryLimit - problem's memory limit in MB

Statement

Represents a problem's statement.
encoding - statement's encoding
name - problem's name in statement's language
legend - problem's language
input - problem's input format
output - problem's output format
scoring - problem's scoring
interaction - problem's interaction protocol (only for interactive problems)
notes - statement notes
tutorial - problem's tutorial

File

Represents a resource, source or aux file.

name - file name
modificationTimeSeconds - file's modification time in unix format
length - file length
sourceType (present only for source files) - source file type
resourceAdvancedProperties - (may be absent) Problem resource files can have extra property called *resourceAdvancedProperties* of type ResourceAdvancedProperties

ResourceAdvancedProperties

Represents special properties of resource files. Basically, they stand for compile- or run-time resources for specific file types and asset types. The most important application is IOI-style graders.
Example: `{"forTypes":"cpp.*","main":false,"stages":["COMPILE"],"assets":["SOLUTION"]}`
forTypes - colon or semicolon separated list of file types this resource if applied, wildcards are supported (example: "cpp.*" or "pascal.*;java.11")
main - currently reserved to be false,
stages - array of possible string values COMPILE or RUN, meaning the phase when the resource is applicable,
assets - array of possible string values VALIDATOR, INTERACTOR, CHECKER, SOLUTION, meaning the asset types the resource is applicable.

Solution

Represents a problem solution.
name - solution name
modificationTimeSeconds - solution's modification time in unix format
length - solution length
sourceType - source file type
tag - solution tag

Test

Represents a test for the problem.
index - test index
manual - boolean - whether test is manual of generated
input - test input (absent for generated tests)
description - test description (may be absent)
useInStatements - boolean - whether test is included in statements
scriptLine - script line for generating test (absent for manual tests)
group - test group (may be absent)
points - test points (may be absent)
inputForStatement - input for statements (may be absent)
outputForStatement - output for statements (may be absent)
verifyInputOutputForStatements - boolean - whether to verify input and output for statements (may be absent)

TestGroup

Represents a test group in testset.
name - test group name
pointsPolicy - test group points policy, COMPLETE_GROUP for the complete group points policy, EACH_TEST for the each test points policy
feedbackPolicy - test group feedback policy, COMPLETE for the complete feedback policy, ICPC for the first error feedback policy, POINTS for the only points feedback policy, NONE for no feedback policy.
dependencies - list of group names from which this group depends on (may be empty)

Package

Represents a package.
id - package's id
revision - revision of the problem for which the package was created
creationTimeSeconds - package's creation time in unix format
state - PENDING/RUNNING/READY/FAILED package's state
comment - comment for the package
type - type of the package: "standard", "linux" or "windows". Standard packages don't contain generated tests, but contain windows executables and scripts for windows and linux via wine. Linux packages contain generated tests, but don't contain compiled binaries. Windows packages contain generated tests and compiled binaries for Windows.

ValidatorTest

Represents a validator's test for the problem.
index - validator test index
input - validator test input
expectedVerdict - INVALID/VALID - expected verdict for the validator test
testset - validator test set (may be absent)
group - validator test group (may be absent)

CheckerTest

Represents a checker's test for the problem.
index - checker test index
input - checker test input
output - checker test output
answer - checker test answer
expectedVerdict - OK/WRONG_ANSWER/CRASHED/PRESENTATION_ERROR - expected verdict for the validator's test