D:Wave
The Quantum Computing Company™

Solver API REST Web Services Developer Guide

USER MANUAL

2020-02-19

Overview

This document describes the SAPI REST interface, including the calls you may use to submit and manage problems and access the available solvers.

CONTACT

Corporate Headquarters
3033 Beta Ave
Burnaby, BC V5G 4M9
Canada
Tel. 604-630-1428

US Office
2650 E Bayshore Rd
Palo Alto, CA 94303

Email: info@dwavesys.com

www.dwavesys.com

D-Wave User Manual 09-1007A-H

Notice and Disclaimer

D-Wave Systems Inc. (D-Wave), its subsidiaries and affiliates, makes commercially reasonable efforts to ensure that the information in this document is accurate and up to date, but errors may occur. NONE OF D-WAVE SYSTEMS INC., its subsidiaries and affiliates, OR ANY OF ITS RESPECTIVE DIRECTORS, EMPLOYEES, AGENTS, OR OTHER REPRESENTATIVES WILL BE LIABLE FOR DAMAGES, CLAIMS, EXPENSES OR OTHER COSTS (INCLUDING WITHOUT LIMITATION LEGAL FEES) ARISING OUT OF OR IN CONNECTION WITH THE USE OF THIS DOCUMENT OR ANY INFORMATION CONTAINED OR REFERRED TO IN IT. THIS IS A COMPREHENSIVE LIMITATION OF LIABILITY THAT APPLIES TO ALL DAMAGES OF ANY KIND, INCLUDING (WITHOUT LIMITATION) COMPENSATORY, DIRECT, INDIRECT, EXEMPLARY, PUNITIVE AND CONSEQUENTIAL DAMAGES, LOSS OF PROGRAMS OR DATA, INCOME OR PROFIT, LOSS OR DAMAGE TO PROPERTY, AND CLAIMS OF THIRD PARTIES.

D-Wave reserves the right to alter this document and other referenced documents without notice from time to time and at its sole discretion. D-Wave reserves its intellectual property rights in and to this document and its proprietary technology, including copyright, trademark rights, industrial design rights, and patent rights. D-Wave trademarks used herein include D-Wave®, D-Wave 2X™, D-Wave 2000Q™, Leap™, and the D-Wave logos (the D-Wave Marks). Other marks used in this document are the property of their respective owners. D-Wave does not grant any license, assignment, or other grant of interest in or to the copyright of this document, the D-Wave Marks, any other marks used in this document, or any other intellectual property rights used or referred to herein, except as D-Wave may expressly provide in a written agreement. This document may refer to other documents, including documents subject to the rights of third parties. Nothing in this document constitutes a grant by D-Wave of any license, assignment, or any other interest in the copyright or other intellectual property rights of such other documents. Any use of such other documents is subject to the rights of D-Wave and/or any applicable third parties in those documents.

All installation, service, support, and maintenance of and for the D-Wave System must be performed by qualified factory-trained D-Wave personnel. Do not move, repair, alter, modify, or change the D-Wave System. If the equipment is used in a manner not specified by D-Wave, the protection provided by the equipment may be impaired. Do not provide access to the customer site to anyone other than authorized and qualified personnel. Failure to follow these guidelines may result in disruption of service, extended downtime, damage to equipment (customer's, D-Wave's, and/or third parties'), injury, loss of life, or loss of property.

Contents

CHAPTER 1

ABOUT THIS DOCUMENT

The Solver API (SAPI) is an interface to the D-Wave quantum processing unit (QPU) and other solvers.

1.1 Intended Audience

This document is intended for developers who want to build their own applications on top of the SAPI REST interface. Readers should be familiar with:

- HTTP protocol
- JSON
- curl

1.2 Scope

This document describes the SAPI REST interface, including the calls you may use to submit and manage problems and access the available solvers.

1.3 Related Documents

See also the following related documents:

- *Getting Started with the D-Wave System*
- *D-Wave Solver Properties and Parameters: Reference Guide*

CHAPTER 2

GETTING STARTED

The Solver API (SAPI) is a front-end for the D-Wave quantum processing unit (QPU) solver[1] and a variety of other advanced software solvers. A REST-based web services API, SAPI is used internally by D-Wave client applications.

SAPI allows you to formulate a wide range of problems amenable to quantum acceleration. The API is fairly simple to use; it exposes only a few basic calls. Most of the work for you will be restructuring your problem into a format that the API can answer, and then parsing results.

2.1 What is REST?

REST, which stands for Representational State Transfer, is a lightweight web services protocol. It came into being as a response to heavyweight Web services protocols such as SOAP and XML-RPC, which rely on a predefined messaging format and methods to pass them back and forth between servers and clients. REST, on the other hand, specifies no such limitations; you can use any message format that you like (whether it be JSON, XML, HTML, serialized data, or even straight text), and operates on the standard HTTP verbs of GET, DELETE, and POST for its operations.

The rules that exist for REST client/server interactions can be completely defined by the application requirements. So if you define a REST interface that is intended to be consumed by a JavaScript client, you probably want to have the data returned in a JSON format, while if you intend for the data to be consumed by a PHP client, then perhaps serialized data or XML is a better choice.

Each REST web services call is a simple HTTP request, using one of the standard HTTP verbs:

- GET—Retrieve data from a server
- POST—Send data to a server
- DELETE—Delete a resource from a server

[1] A solver is simply a resource that receives a problem and returns an answer; it may be a D-Wave QPU or a software emulation of one.

2.2 HTTP Response Codes

Table 2.1: HTTP Response Codes

Code	Explanation
200 OK	No error.
201 CREATED	Creation of a resource was successful.
202 ACCEPTED	Problem cancellation request was received.
304 NOT MODIFIED	The resource hasn't changed since the time specified in the request's `If-Modified-Since` header.
400 BAD REQUEST	Invalid request URI or header, or unsupported nonstandard parameter.
401 UNAUTHORIZED	Authorization required. This error can also mean that incorrect user credentials were sent with the request.
403 FORBIDDEN	Unsupported standard parameter, or authentication or authorization failed.
404 NOT FOUND	Resource (such as a problem) not found.
409 ALREADY TERMINATED	Problem reached one of terminal states before the call.
429 TOO MANY REQUESTS	API request rate exceeds the permissible limit.
500 INTERNAL SERVER ERROR	Internal error. This is the default code that is used for all unrecognized server errors.

2.3 Setup

Examples in this document use `curl` to send HTTP requests to the SAPI web service. Set the base URL and the API token environment variables to the home URL of the SAPI web service and an API key on the server hosting the web service respectively.

For example:

```
export SAPI_HOME=https://cloud.dwavesys.com/sapi/
export SAPI_TOKEN=ProjectA-tokens-string
```

Note: All URLs referenced in the documentation have the following base: https://cloud.dwavesys.com/sapi/. This API endpoint can vary based on the system setup and is referenced throughout the document as `SAPI_HOME`.

2.4 Authentication and Access

All requests to SAPI require users to authenticate using an API token. API tokens are associated with *projects*. Each project has a priority and one or more solvers that users who are assigned to that project can access. Each problem that SAPI accepts is authenticated via the token and sent to a solver for solution. API tokens do not differentiate between solvers in their project.

An API token is sent to SAPI in the form of HTTP header `X-Auth-Token`. A simple example of getting list of remote solvers using `curl` is as follows:

```
SAPI_HOME=https://cloud.dwavesys.com/sapi/
SAPI_TOKEN=ProjectA-tokens-string
curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/solvers/remote/
```

Projects and individual users may have quotas configured, which control solver access time.

CHAPTER 3

PROBLEM RESOURCES

A problem resource represents a problem submitted by a user to a solver, including large problems uploaded in multiple parts. See the *Data Encoding* chapter to understand the supported data types for problems (and solutions).

3.1 SAPI Endpoints

The following table provides an overview of the SAPI URLs that map to problem resources.

Table 3.1: SAPI URLs

URL	Method	Reference
/bqm/multipart	POST	*Initiate a Multi-Part Upload*
/bqm/multipart/<id>/part/<part>	PUT	*Upload Data in a Multi-Part Upload*
/bqm/multipart/<id>/combine	POST	*Complete a Multi-Part Upload with Checksum*
/bqm/multipart/<id>/status	GET	*Status for a Multi-Part Upload*
/problems/	POST	*Submit Problem*
/problems/	DELETE	*Cancel Multiple Problems*
/problems/<problem_id>	DELETE	*Cancel a Problem by ID*
/problems/<filters>	GET	*List Problems*
/problems/<problem_id>/	GET	*Get Problem*
/problems/<problem_id>/info	GET	*Get Problem Information*
/problems/<problem_id>/answer/	GET	*Get Problem Answer*
/problems/<problem_id>/messages/	GET	*Get Problem Messages*

3.2 Problem Lifecycle

A problem may have one of the following statuses:

Table 3.2: Problem Status

Status	Description
PENDING	Problem was submitted and is queued for processing.
IN_PROGRESS	Problem is currently being processed.
COMPLETED	Problem completed successfully.
FAILED	Problem processing failed.
CANCELLED	Problem was cancelled.

Note: If a quota is reached before a problem is processed, the problem remains in a PENDING state until the quota resets or the problem expires.

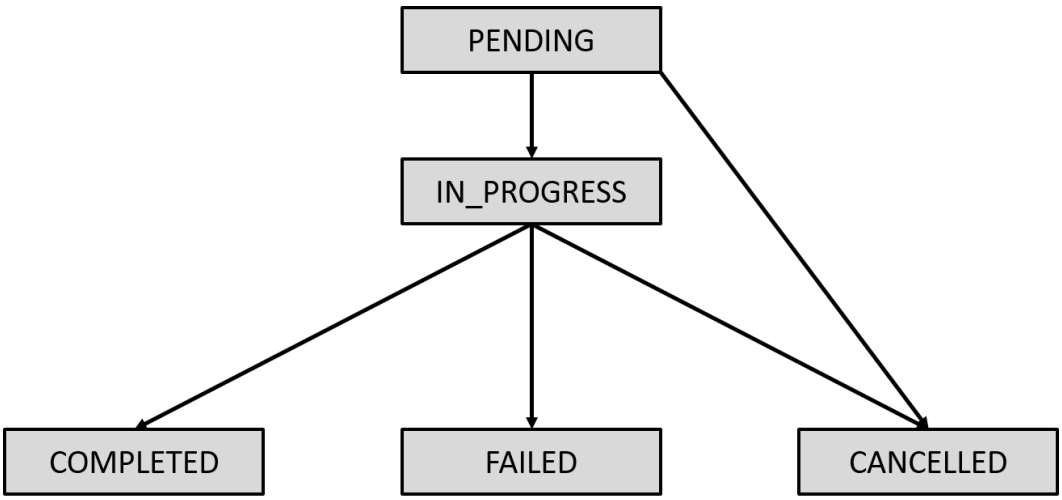

Figure 3.1: Initially a problem is in the PENDING state. When the D-Wave system starts to process a problem, its state changes to IN_PROGRESS. After completion, the problem status changes to either COMPLETED or FAILED (if an error occurred). COMPLETED, FAILED, and CANCELLED are all terminal states. After a problem enters a terminal state, its status does not change. Users can cancel a problem at any time before it reaches its terminal state.

3.3 Large Problem Upload

For large problems (starting from several megabytes and higher), upload the problem data in multiple parts using the `$SAPI_HOME/bqm/multipart` endpoints.

This example uses `curl` to upload a 13 megabyte problem. For simplicity, the problem is stored in two files of 7 MB and 6 MB, respectively. Both commands and responses are shown. Status requests are optional. For details, see the command descriptions below.

```
$ curl  -H "X-Auth-Token: $SAPI_TOKEN" -H "Content-type: application/json" -X POST
 ↪$SAPI_HOME/bqm/multipart/ -d '{"size":13000000}'
{"id":"60aa9e10-642b-4a90-a85e-d8a28ceab065"}

$ id=60aa9e10-642b-4a90-a85e-d8a28ceab065
$ curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/bqm/multipart/$id/status
{"status": "UPLOAD_IN_PROGRESS", "parts": []}

$ curl -H "X-Auth-Token: $SAPI_TOKEN" -H "X-Content-MD5: ZYY9Mp6Z/7RnrBoHWU+aWQ==" -
 ↪X PUT $SAPI_HOME/bqm/multipart/$id/part/1 -d @"7Mbqm.txt"
$ curl  -H "X-Auth-Token: $SAPI_TOKEN" -H "X-Content-MD5: o6mhPrp5RVP/c15S/+sYJQ==" ␣
 ↪-X PUT $SAPI_HOME/bqm/multipart/$id/part/2 -d @"6Mbqm.txt"

$ curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/bqm/multipart/$id/status
{"status": "UPLOAD_IN_PROGRESS", "parts": [{"part_number": 1, "checksum":
 ↪"65863d329e99ffb467ac1a07594f9a59"}, {"part_number": 2, "checksum":
 ↪"a3a9a13eba794553ff735e52ffeb1825"}]}

$ curl  -H "X-Auth-Token: $SAPI_TOKEN" -H "Content-type: application/json" -X POST
 ↪$SAPI_HOME/bqm/multipart/$id/combine/ -d '{"checksum":
 ↪"11d12dd76abd0668bf385e8fbf4562a8"}'
{}

$ curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/bqm/multipart/$id/status
{"status": "UPLOAD_COMPLETED", "parts": []}
```

3.3.1 Initiate a Multi-Part Upload

To initiate the uploading of a large problem in multiple parts, send an HTTP POST request to `$SAPI_HOME/bqm/multipart/`. The POST request body should contain the number of bytes, *size*, as an integer, required to store the problem.

This example uses `curl`:

```
$ curl  -H "X-Auth-Token: $SAPI_TOKEN" -H "Content-type: application/json" -X POST
 ↪$SAPI_HOME/bqm/multipart/ -d '{"size":<size>}'
```

Response

The system returns an ID of type UUID for the uploaded problem.

Response codes are as follows:

Table 3.3: Response Codes for `Initiate a Multi-Part Upload`

Response Code	Meaning
200	Success.
400	Failed to initiate.
500	Unspecified failure.

3.3.2 Upload Data in a Multi-Part Upload

To upload parts of a large problem uploaded in multiple parts, send an HTTP PUT request to `$SAPI_HOME/bqm/multipart/<id>/part/<part>/` for each part. The POST request body should contain the data for one part of the problem and the header should contain the MD5 checksum.

Each part's size should be between 5 MB to 10 MB (note that 1 MB is 1048576 bytes). If you are uploading three or more parts, except for the last part, all parts must be the same size. For example, to upload a 22 MB problem, you might split it into four 5 MB parts and a last part of 2 MB or into two parts of 10 MB and a last part of 2 MB or other permutations that follow these two rules.

This example uses `curl`:

```
curl -H "X-Auth-Token: $SAPI_TOKEN" -H "X-Content-MD5: <MD5 checksum>" -X PUT $SAPI_
 ↪HOME/bqm/multipart/$id/part/<part number> -d <data for one part>
```

The MD5 checksum can be found, for example, using the Python hashlib library function *hashlib.md5()*. The following Python code example finds the checksum for a 7 MB problem part saved in a file named *7Mbqm.txt*:

```
>>> import hashlib
>>> import base64
>>> hash_md5_7m = hashlib.md5()
>>> with open("7mbqm.txt", "r") as file:
        for chunk in iter(lambda: file.read(4096), ""):
            hash_md5_7m.update(chunk.encode())
>>> print(base64.b64encode(hash_md5_7m.digest()).decode('utf-8'))
ZYY9Mp6Z/7RnrBoHWU+aWQ==
```

Response

Response codes are as follows:

Table 3.4: Response Codes for `Upload Data in a Multi-Part Upload`

Response Code	Meaning
200	Success.
400	Failed to upload or invalid checksum.
404	Failed to find specified upload.
500	Unspecified failure.

3.3.3 Complete a Multi-Part Upload with Checksum

To complete the upload a large problem uploaded in multiple parts, send an HTTP POST request to `$SAPI_HOME/bqm/multipart/<id>/combine` with the combined checksum. The POST request body should contain the checksum for the entire problem, as a string.

Note: Amazon calculates the checksum of the combined file in a particular way. Rather than the checksum from combining the parts, it concatenates all the checksums of the parts and then calculates the checksum of that concatenation. Your checksum must follow this same format.

This example uses `curl`:

```
curl  -H "X-Auth-Token: $SAPI_TOKEN" -H "Content-type: application/json" -X POST
 ↪$SAPI_HOME/bqm/multipart/$id/combine/ -d '{"checksum":"<Amazon-style checksum>"}'
```

The checksum can be found, for example, using the Python hashlib library function *hashlib.md5()*. The example below finds the concatenated checksum for a large problem saved in two parts with MD5 checksums (see *Upload Data in a Multi-Part Upload* for information on calculating checksums for the parts) in variables *hash_md5_7m* and *hash_md5_6m*:

```
>>> import hashlib
>>> print((hashlib.md5(bytes.fromhex(
           hash_md5_7m.hexdigest() +
           hash_md5_6m.hexdigest())
           )).hexdigest())
11d12dd76abd0668bf385e8fbf4562a8
```

Response

Response codes are as follows:

Table 3.5: Response Codes for `Complete a Multi-Part Upload with Checksum`

Response Code	Meaning
200	Success.
400	Failed to upload, no or invalid checksum.
404	Failed to find specified upload.
500	Unspecified failure.

3.3.4 Status for a Multi-Part Upload

To get the status of a large problem uploaded in multiple parts, send an HTTP GET request to `$SAPI_HOME/bqm/multipart/<id>/status`. The GET request should contain no body.

This example uses `curl`:

```
curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/bqm/multipart/$id/status
```

Response

The system returns the status as a string with parts and checksums.

Response codes are as follows:

Table 3.6: Response Codes for `Complete a Multi-Part Upload with Checksum`

Response Code	Meaning
200	Success.
400	Failed to upload, no or invalid checksum.
404	Failed to find specified upload.
500	Unspecified failure.

3.4 Submit Problem

3.4.1 Request

To submit problems, send an HTTP POST request to `$SAPI_HOME/problems/`. The POST request body should contain JSON-encoded problem information. When possible, if you have more than one problem, submit them in a single query.

This example uses `curl`:

```
curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/problems -X POST
-d '[{"type":"ising","solver":"c4-sw_sample","data":{"lin":
 ↪"AAAAAAAA4L8AAAAAADwPwAAAAAAAAAAAAAAAAA+H+amZmZmZnJPwAAAAAAAPh/
 ↪AAAAAAAA+H8A\r\nAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAA\r\nAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/AAAA\r\nAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/AAAAAAAA+H8AAAAA\r\nAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAA\r\nAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA\r\n+H8AAAAAAAD4fwAAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4\r\nfwAAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/\r\nAAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/AAAAAAAA+H8A\r\nAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAA\r\nAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAA\r\nAAAA+H8AAAAAAAD4fwAAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAA\r\nAAD4fwAAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAA\r\nAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/AAAAAAAA\r\n+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/AAAAAAAA+H8AAAAAAAD4\r\nfwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪\r\nAAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8A\r\nAAAAAAD4fwAAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fw==","quad":"AAAAAAAA8L8AAAAAADgP5qZmZmZmem/","format":"qp
 ↪"},"params":{"answer_mode":"histogram","num_reads":10}}]'
```

An example of submitting a large problem uploaded as a multi-part problem:

```
curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/problems -X POST
-d '[{"type":"ising","solver":"hybrid-solver1","data":{'format': 'bqm-ref', 'data':
 ↪'9b515c01-e311-46c3-b696-ded3fdfe5d0b'},"params":{"answer_mode":"histogram","num_
 ↪reads":1}}]'
```

3.4.2 Problem Data

The JSON data for the submitted problem must contain the following properties:

Table 3.7: Key-Value Pairs for `Submit Problem` Request

Key	Value
solver	Solver to be used.
data	Problem data; see the *Data Encoding* section.
type	One of the supported problem types (`ising` or `qubo`).
params	Solver-specific parameters. See *Data Encoding for Solver Parameters*.

Note: The allowed problem types are listed in the solver's properties.

3.4.3 Response

The system returns a list with the results to the corresponding problems in the JSON request.

Error in problem (example):

```
{
    "error_code": 400,
    "error_msg": "Missing parameter 'num_reads' in problem JSON"
}
```

Table 3.8: Key-Value Pairs for `Submit Problem` Response with Error

Key	Value
error_code	Error code describing the type of error encountered; 400 for missing parameters, 403 for no access to solver, 429 for too many requests.
error_msg	Message describing the error encountered

Successful completion (example):

```
{
    "status": "COMPLETED",
    "earliest_estimated_completion": "2018-08-13T20:47:03.009543Z",
    "solved_on": "2018-08-13T20:47:03.066239Z",
    "solver": "c4-sw_sample",
    "submitted_on": "2018-08-13T20:47:03.009343Z",
    "answer": {
        "num_variables": 128,
        "format": "qp",
        "energies": "zczMzMzMDMA=",
        "num_occurrences": "CgAAAA==",
        "active_variables": "AAAAAAEAAAACAAAABAAAAA==",
        "solutions": "sA==",
        "timing": {}
        },
    "latest_estimated_completion": "2018-08-13T20:47:03.009543Z",
```

```
    "type": "ising",
    "id": "d8b0a7d0-01e5-4d27-bae2-93cd23f3204c"
  }
```

Completion with errors (example):

```
{
    "status": "FAILED",
    "solved_on": "2018-01-18T10:26:00.020954",
    "solver": "c4-sw_sample",
    "submitted_on": "2018-01-18T10:25:59.941674",
    "type": "ising",
    "id": "f004cea4-0f18-4b44-8aca-4e3acbb6831b",
    "error_message": "Some error message"
}
```

Table 3.9: Key-Value Pairs for `Submit Problem` Responses

Key	Value
answer	Content of the answer depends on the solver and parameters used. See *Data Encoding for Solutions* for more details about the structure of this field.
earliest_estimated_completion	Estimated completion time: beginning of range.
error_message	Error message generated by the system if problem status is FAILED.
id	Unique identifier of the problem; can be used later to retrieve problem information, the answer, and the messages.
latest_estimated_completion	Estimated completion time: end of range.
solved_on	If this problem is in terminal state (COMPLETED, CANCELLED or FAILED), time when problem was solved. Empty otherwise.
solver	Name of the solver to which the problem was submitted.
status	One of the problem states as defined in *Problem Lifecycle*.
submitted_on	Time when problem was submitted.
type	Problem type as specified when the problem was submitted.

3.5 Cancel Multiple Problems

3.5.1 Request

To cancel PENDING problems, send an HTTP DELETE request to `$SAPI_HOME/problems/`. The request body should be a JSON-encoded list of problem IDs; if the request body is empty, the request has no effect. When possible, if you have more than one problem to cancel, submit them in a single query.

This example uses `curl`:

```
curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/problems/?id=792749f9-3bbc-4eea-84da-
 ↪e51726ac0be6,id2,id3 -X DELETE
```

3.5.2 Response

The system returns a list with the results to the corresponding problems in the JSON request in one of the following formats.

Error in problem (example):

```
{
    "error_code": 409,
    "error_msg": "Problem already terminated"
}
```

Response codes are as follows:

Table 3.10: Response Codes for `Cancel Problem`

Response Code	Meaning
200	Success.
202	Cancellation request received. Relevant for in-progress jobs. The problem is cancelled if the request was received in time; otherwise, it will complete.
404	Problem with the specified ID does not exist.
409	Problem reached one of terminal states before the call.
429	Number of API requests exceeds the permitted limit.

Table 3.11: Key-Value Pairs for `Cancel Problem` Response with Error

Key	Value
error_code	Error code describing the type of error encountered
error_msg	Message describing the error encountered

Successful cancellation (example):

```
{
    "status": "CANCELLED",
    "solved_on": "2018-01-18T10:26:00.020954",
    "solver": "c4-sw_sample",
    "submitted_on": "2018-01-18T10:25:59.941674",
    "type": "ising",
    "id": "f004cea4-0f18-4b44-8aca-4e3acbb6831b"
}
```

Table 3.12: Key-Value Pairs for `Cancel Problem` Response

Key	Value
id	Unique identifier of the problem.
status	Problem state: CANCELLED.
submitted_on	The time when problem was submitted.
solved_on	The time when problem was cancelled.
type	Problem type as specified when the problem was submitted.

3.6 Cancel a Problem by ID

3.6.1 Request

To cancel a previously submitted problem, make an HTTP DELETE request to `$SAPI_HOME/problems/`.

```
$SAPI_HOME/problems/<problem_id>/
```

3.6.2 Response

The system returns the problem formatted as a JSON object.

Example of the output:

```
{
    "status": "CANCELLED",
    "solved_on": null,
    "solver": "C4_19091607.17_C6",
    "submitted_on": "2018-01-18T10:06:10.025064",
    "type": "ising",
    "id": "894fb8e7-4176-4b39-92f4-10ad56432f09"
}
```

Response codes are as follows:

Table 3.13: Response Codes for `Cancel Problem`

Response Code	Meaning
200	Success.
202	Cancellation request received. Relevant for in-progress jobs. The problem is cancelled if the request was received in time; otherwise, it will complete.
404	Problem with the specified ID does not exist.
409	Problem reached one of terminal states before the call.
429	Number of API requests exceeds the permitted limit.

3.7 List Problems

3.7.1 Request

To retrieve a list of previously submitted problems, make an HTTP GET request to `$SAPI_HOME/problems/`. You can filter the results using one or more of the following parameters:

Table 3.14: Key-Value Pairs for `List Problems`

Key	Value
id	comma-separated list of problem IDs
max_results	Maximum number of results to return. Returns up to 1000 if not specified.
status	Problem state: COMPLETED, IN_PROGRESS, PENDING, FAILED, CANCELLED.
solver	Solver name (e.g., hybrid_v1).

This example uses `curl`:

```
$ curl  -H "X-Auth-Token: $SAPI_TOKEN" -X GET $SAPI_HOME/problems/?solver=hybrid_v1&
 ↪max_results=5&status=COMPLETED
```

3.7.2 Response

The system returns problems as a JSON list.

Example of the output:

```
[
    {
        "status": "COMPLETED",
        "solved_on": "2012-12-05T19:15:04+00:00",
        "solver": "SR8",
        "submitted_on": "2012-12-05T19:06:57+00:00",
        "type": "ising",
        "id": "0b759042-726e-4665-b04b-3b42e1ed0b01"
    },
    {
        "status": "COMPLETED",
        "solved_on": "2012-12-05T19:15:07+00:00",
        "solver": "SR8",
        "submitted_on": "2012-12-05T19:06:57+00:00",
        "type": "ising",
        "id": "80387b7a-0976-43da-8ed0-e7a27821c177"
    }
]
```

Response codes are as follows:

Table 3.15: Response Codes for `List Problem`

Response Code	Meaning
200	Response contains valid problems list.
404	Problem with specified ID does not exist.
429	Number of API requests exceeds the permitted limit.

3.8 Get Problem

3.8.1 Request

To retrieve a previously submitted problem, make an HTTP GET request to `$SAPI_HOME/problems/`:

```
$SAPI_HOME/problems/<problem_id>/
```

When possible, if you want more than one problem, submit the request in a single query.

3.8.2 Response

When a problem is solved, the system includes the specified problem formatted as a JSON object in its response.

Example of the output:

```
{
    "status": "PENDING",
    "solved_on": null,
    "solver": "4.7_C4_19091607.17_C6",
    "submitted_on": "2018-01-18T10:06:10.025064",
    "type": "ising",
    "id": "894fb8e7-4176-4b39-92f4-10ad56432f09"
}
```

Response codes are as follows:

Table 3.16: Response Codes for `Get Problem`

Response Code	Meaning
200	Success
404	Problem with specified ID does not exist.
429	Number of API requests exceeds the permitted limit.

3.9 Get Problem Information

3.9.1 Request

To retrieve information about a problem, make an HTTP GET request to `$SAPI_HOME/problems/<problem ID>/info`:

This example uses `curl`:

```
$ curl  -H "X-Auth-Token: $SAPI_TOKEN" -X GET $SAPI_HOME/problems/b46176c1-4089-
 ↪4772-804f-2028cbf2e9a4/info
```

3.9.2 Response

Returns information about the problem including the problem ID, the problem or an ID for a large problem uploaded in multiple parts by *Upload Data in a Multi-Part Upload*, metadata about the submission, and the answer returned by the solver.

Below is an **edited** example of a JSON-encoded problem answer in a response message:

```
{"id": "b46176c1-4089-4772-804f-2028cbf2e9a4",
 "data":
     {"format": "bqm-ref",
      "data": "bd8d6ec0-4435-4f44-bed8-70d9e322f8b9"},
 "params":
     {"time_limit": 2},
 "metadata":
     {"submitted_by": "ISDE-3fthis99is99an99api99toek0667c04",
      "solver": "hybrid_v1",
      "type": "bqm",
      "submitted_on": "2020-01-21T18:12:41.860425Z",
      "solved_on": "2020-01-21T18:12:43.980222Z",
      "earliest_estimated_completion": "2020-01-21T18:12:43.860425Z",
      "latest_estimated_completion": "2020-01-21T18:13:10.660425Z",
      "status": "COMPLETED",
      "messages": []},
 "answer":
     {"format": "bq",
      "data":
           {"type": "SampleSet",
            "version":
                {"sampleset_schema": "3.0.0"},
            "num_variables": 300,
            "num_rows": 1,
            "sample_data":
                {"type": "array",
                 "data": [[2567045157, ... , 1140850688, 32]],
                 "data_type": "uint32",
                 "shape": [1, 10],
                 "use_bytes": false},
            "sample_type": "float64",
```

```
            "vectors":
                {"energy":
                    {"type": "array",
                     "data": [-40.0],
                     "data_type": "float64",
                     "shape": [1],
                     "use_bytes": false},
                 "num_occurrences":
                    {"type": "array",
                     "data": [1],
                     "data_type": "int64",
                     "shape": [1],
                     "use_bytes": false}},
                "variable_labels": [0, 1, 2, 3,... 297, 298, 299],
                "variable_type": "BINARY",
                "info": {}}}}
```

3.10 Get Problem Answer

3.10.1 Request

To retrieve an answer for the problem, make an HTTP GET request to `$SAPI_HOME/problems/`:

```
$SAPI_HOME/problems/<problem_id>/answer/
```

3.10.2 Response

When a problem is solved, the system returns the problem answer formatted as a JSON object, or `404` if none exists.

Below is an example of a JSON-encoded problem answer in a response message:


```
{
    "answer": {
        "format": "qp",
        "num_variables": 1152,
        "solutions": "AwEAAg==",
        "energies": "gBSuR+F6lL+AFK5H4XqUv4AUrkfhepS/gBSuR+F6lL8=",
        "active_variables": "AQAAAAQAAAA=",
        "num_occurrences": "AwAAAAIAAAACAAAAAwAAAA==",
        "timing": {
        }
    }
}
```


See the *Data Encoding for Solutions* section for a list of the key-value pairs returned in the response.

Response codes are as follows:

Table 3.17: Response Codes for `Get Problem Answer`

Response Code	Meaning
200	Success.
404	Answer for the problem does not exist.
429	Number of API requests exceeds the permitted limit.

3.11 Get Problem Messages

3.11.1 Request

To retrieve messages for a problem, make an HTTP GET request to `$SAPI_HOME/problems/`.

```
$SAPI_HOME/problems/<problem_id>/messages/
```

3.11.2 Response

The system returns messages formatted as a JSON object or `404` if an incorrect problem ID is specified. If there are no messages, an empty list is returned.

Example of the output:

```
[
    {
        "timestamp": "2011-06-06T14:24:40.393751",
        "message": "Internal SAPI error occurred.",
        "severity": "ERROR"
    }
]
```

Response codes are as follows:

Table 3.18: Response Codes for `Get Problem Messages`

Response Code	Meaning
200	Success.
404	Problem does not exist.
429	Number of API requests exceeds the permitted limit.

CHAPTER 4

SOLVERS RESOURCES

solver resources represent resources that run a problem.

4.1 SAPI Endpoints

The following table provides an overview of the SAPI URLs that map to solver resources.

Table 4.1: SAPI URLs

URL	Method	Reference
/solvers/remote/	GET	*List Solvers*
/solvers/remote/<solver_id>/	GET	*Get Solver Configuration*

4.2 Solver Properties

Table 4.2: Solver Resource Properties

Property	Description
description	Description of the solver.
property	Solver properties such as supported problem types, active qubits, active couplers, total number of qubits, and so on.
id	Unique ID of the solver.

For details on solver types, solver properties, and the problem-solving parameters that are supported by the different solver types, see the *D-Wave Solver Properties and Parameters: Reference Guide*.

4.3 List Solvers

4.3.1 Request

Retrieve the list of supported remote solvers by sending an HTTP GET request to `$SAPI_HOME/solvers/`. This example uses `curl` to retrieve the list of remote solvers:

```
curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/solvers/remote
```

4.3.2 Response

The system responds with `200 OK` and returns list of solvers formatted as a JSON list.

Example of the output:

```
[
    {
        "properties": {
            "supported_problem_types": ["qubo", "ising"],
            "qubits": [ ... ],
            "couplers": [ ... ],
            "num_qubits": 512
        }
        "id": "Solver 1",
        "description": "Solver #1"
    },
    {
        "properties": {
            "supported_problem_types": ["ising"],
            "qubits": [ ... ],
            "couplers": [ ... ],
            "num_qubits": 512
        }
        "id": "Solver 2",
        "description": "Solver #2"
    }
]
```

4.4 Get Solver Configuration

4.4.1 Request

Retrieve the configuration of a solver by sending an HTTP GET request to `$SAPI_HOME/solvers/`. This example uses `curl` to retrieve the configuration of a remote solver:

```
curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/solvers/remote/solver1
```

4.4.2 Response

The system returns solver configuration formatted as a JSON object.

Example of the output:

```
{
    "properties": {
        "supported_problem_types": ["qubo", "ising"],
        "qubits": [ ... ],
        "couplers": [ ... ],
        "num_qubits": 512
    }
    "id": "solver1",
    "description": "solver1"
}
```

The `properties` attribute holds the solver properties such as number of active qubits and couplers.

Response codes are as follows:

Table 4.3: Response Codes for `Get Solver Configuration`

Response Code	Meaning
200	Success
404	Solver with specified `solver_id` does not exist.
429	Number of API requests exceeds the permitted limit.

CHAPTER 5

DATA ENCODING

This section describes the data encoding for problems and solutions.

5.1 Data Encoding for Problems

The REST API supports two types of encoding for problems submitted to the system. Use the appropriate code in the `type` field of the submission message.

Problem Type	Code
Ising	`ising`
QUBO	`qubo`

5.1.1 JSON Encoding

Problems are encoded as JSON objects with the following properties:

Table 5.1: JSON Encoding Properties: Problems

Key	Value
format	String: `qp` or, for large (multi-part upload) problems, `bqm-ref`
lin	Linear coefficients (base64-encoded little-endian doubles). One value per working qubit in the same order as solver's *qubits* property; NaN indicates an inactive qubit.
quad	Quadratic coefficients (base64-encoded little-endian doubles). One value per active coupler in the same order as solver's *couplers* property. An *active coupler* means that both qubits the coupler is incident to are active. NaN values are not permitted.

An example of an HTTP POST request for submitting a problem using `curl` in JSON encoding is as follows:

```
curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/problems -X POST
-d '[{"type":"ising","solver":"c4-sw_sample","data":{"lin":
 ↪"AAAAAAAA4L8AAAAAAADwPwAAAAAAAAAAAAAAAAA+H+amZmZmZnJPwAAAAAAAPh/
 ↪AAAAAAAA+H8A\r\nAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAA\r\nAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/AAAA\r\nAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAA\r\nAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAA\r\nAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA\r\n+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4\r\nfwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/\r\nAAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8A\r\nAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAA\r\nAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAA\r\nAAAA+H8AAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAA\r\nAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAA\r\nAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/AAAAAAAA\r\n+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4\r\nfwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪\r\nAAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8A\r\nAAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAAD4fwAAAAAAPh/
 ↪AAAAAAAA+H8AAAAAAAD4fw==","quad":"AAAAAAAA8L8AAAAAAADgP5qZmZmZmem/","format":"qp
 ↪"},"params":{"answer_mode":"histogram","num_reads":10}}]'
```

An example of an HTTP POST request for submitting a large problem uploaded as a multipart problem, using `curl` in JSON encoding is as follows:

```
curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/problems -X POST
-d '[{"type":"ising","solver":"hybrid-solver1","data":{'format': 'bqm-ref', 'data':
 ↪'9b515c01-e311-46c3-b696-ded3fdfe5d0b'},"params":{"answer_mode":"histogram","num_
 ↪reads":1}}]'
```

5.1.2 Text Encoding

Note: While the REST API still supports text encoding, support for this will be deprecated in an upcoming release.

Problem data is passed via SAPI as text, where each line represents a linear or quadratic coefficient:

```
128 3
1 1 -1
2 2 1
1 5 -3
```

The first line consists of the number of qubits followed by the number of variables used for

the problem (that is, the number of qubits and couplers whose values we are setting). Each subsequent line consists of two variable indices, followed by the strength of the coupler that connects them. If the first two variable indices are the same, the third number in the line is the weight of the qubit of the previous indices.

An example of an HTTP POST request for submitting a problem with the above data using `curl` in text encoding is as follows:

```
curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/problems -X POST
-d '[{"solver": "c4-sw_sample", "data": "128 3\n1 1 -1\n2 2 1\n1 5 -3",
"type": "ising", "params": {"num_reads": 123}}]'
```

5.2 Data Encoding for Solver Parameters

Solver parameters, contained in the `params` JSON object, contain key/value pairs; see the table below. The parameters and their default values are described in *Solver Properties and Parameters Reference*.

Table 5.2: JSON Encoding Properties: Solver Parameters (params)

Key	Value
anneal_offsets	JSON array of length equal to the solver property *anneal_offset_ranges*
anneal_schedule	JSON array of two element arrays of floating-point numbers
annealing_time	Integer
answer_mode	[”raw” | “histogram”]
auto_scale	[true | false]
beta	Float
chains	JSON list of lists of qubit indices. The lists are the chains, and they must be disjoint. This parameter is used only by post-processing.
flux_biases	JSON array of floating point numbers with length equal to the *num_qubits* property of the solver. Use `0` for unused or unavailable devices.
flux_drift_compensation	[true | false]
initial_state	JSON array of length equal to *num_qubits,* containing `{1,0,3}` for QUBO problems and `{1,-1,3}` for Ising problems. Use `3` for inactive qubits. This format allows you to take an answer from the SAPI clients and send it directly as this parameter.
max_answers	Positive integer
num_reads	Positive integer
num_spin_reversal_transforms	Zero or a positive integer
postprocess	[””|”sampling”|”optimization”]
programming_thermalization	Integer bounded by solver property *programming_thermalization_range*
readout_thermalization	Integer bounded by solver property *readout_thermalization_range*
reduce_intersample_correlation	[true | false]
reinitialize_state	[true | false]

5.3 Data Encoding for Solutions

The solution encoding for submitted problems is a JSON object with the following properties:

Table 5.3: JSON Encoding Properties: Solutions

Key	Value
format	String: `qp`
num_variables	Total number of variables (active or otherwise) that the solver has. JSON integer.
solutions	Base-64–encoded string of bit-packed solutions (with 0 = -1 for Ising problems). Bits are in little-endian order. Each solution is padded to end on a byte boundary and contains values for active qubits only.
energies	Base-64–encoded string of energies, each a little-endian 8-byte floating-point number (doubles).
active_variables	Base-64–encoded string of the indices of the problem's active variables. The indices are 4-byte little-endian integers.
num_occurrences	Base-64–encoded string of the number of occurrences of each solution. The numbers are 4-byte little-endian integers. Answers are ordered by energy.
timing	Solver-specific JSON object describing the reported time that the solver took to handle the problem.

Note: If the problem is submitted with the backwards-compatible formatting, the answer is also backwards-compatible: all the attributes are JSON arrays instead of base-64–encoded bytes, except the `solutions` field, which is a base-64 encoding of the bit-packed solutions without padding between them.