Overview

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

1 About this Document ................................. 1
   1.1 Intended Audience ......................... 1
   1.2 Scope .................................. 1
   1.3 Related Documents ..................... 1

2 Getting Started ................................. 3
   2.1 What is REST? ............................ 3
   2.2 HTTP Response Codes .................... 4
   2.3 Setup ................................ 4
   2.4 Authentication and Access ............... 5
   2.5 Data Store ............................. 5

3 REST Resources ................................. 7
   3.1 Problem Resource ....................... 7
      3.1.1 Problem Lifecycle ................. 7
      3.1.2 Submit Problem ................... 9
      3.1.3 Cancel Multiple Problems .......... 12
      3.1.4 Cancel a Problem by ID .......... 14
      3.1.5 List Problems ..................... 15
      3.1.6 Get Problem ....................... 16
      3.1.7 Get Problem Answer ............... 17
      3.1.8 Get Problem Messages ............. 18
   3.2 Solvers Resource ......................... 19
      3.2.1 Solver Resource Properties ........ 19
      3.2.2 List Solvers ....................... 20
      3.2.3 Get Solver Configuration .......... 21

4 Data Encoding ................................. 23
   4.1 Data Encoding for Problems ............... 23
      4.1.1 JSON Encoding ..................... 23
      4.1.2 Text Encoding ..................... 24
   4.2 Data Encoding for Solver Parameters ....... 25
   4.3 Data Encoding for Solutions ............... 26
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*
The Solver API (SAPI) is a front-end for the D-Wave quantum processing unit (QPU) solver 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. See D-Wave Solver Properties and Parameters: Reference for more information.
2.2 HTTP Response Codes

Table 2.1: HTTP Response Codes

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

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.

2.5 Data Store

While some problem data is cached, the SAPI server itself is an I/O queue rather than a database. While it has a short-term Redis-based data store, the server does not store solutions long-term. Therefore, it is good practice to periodically retrieve the solutions you want to keep. Expiry times vary by system; plan to retrieve results within a couple of hours after your problem completes.
The Solver API primarily manages two types of resources: problem resources and solver resources. The following table provides an overview of the SAPI URLs that map to resources.

### Table 3.1: SAPI URLs

<table>
<thead>
<tr>
<th>URL</th>
<th>Method</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>/problems/</td>
<td>POST</td>
<td>Submit Problem</td>
</tr>
<tr>
<td>/problems/</td>
<td>DELETE</td>
<td>Cancel Multiple Problems</td>
</tr>
<tr>
<td>/problems/&lt;problem_id&gt;</td>
<td>DELETE</td>
<td>Cancel a Problem by ID</td>
</tr>
<tr>
<td>/problems/[?id=id1,id2]</td>
<td>GET</td>
<td>List Problems</td>
</tr>
<tr>
<td>/problems/&lt;problem_id&gt;/</td>
<td>GET</td>
<td>Get Problem</td>
</tr>
<tr>
<td>/problems/&lt;problem_id&gt;/answer/</td>
<td>GET</td>
<td>Get Problem Answer</td>
</tr>
<tr>
<td>/problems/&lt;problem_id&gt;/messages/</td>
<td>GET</td>
<td>Get Problem Messages</td>
</tr>
<tr>
<td>/solvers/remote/</td>
<td>GET</td>
<td>List Solvers</td>
</tr>
<tr>
<td>/solvers/remote/&lt;solver_id&gt;/</td>
<td>GET</td>
<td>Get Solver Configuration</td>
</tr>
</tbody>
</table>

### 3.1 Problem Resource

A problem resource represents a problem submitted by a user to a solver. This may be a problem submitted via the API or a web-based front-end. See also the Data Encoding chapter to understand the supported data types for problems and solutions.

### 3.1.1 Problem Lifecycle

A problem may have one of the following statuses:

### Table 3.2: Problem Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PENDING</td>
<td>Problem was submitted and is queued for processing.</td>
</tr>
<tr>
<td>IN_PROGRESS</td>
<td>Problem is currently being processed.</td>
</tr>
<tr>
<td>COMPLETED</td>
<td>Problem completed successfully.</td>
</tr>
<tr>
<td>FAILED</td>
<td>Problem processing failed.</td>
</tr>
<tr>
<td>CANCELLED</td>
<td>Problem was cancelled.</td>
</tr>
</tbody>
</table>
**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.
3.1.2 Submit Problem

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":"AAAAAAAA4L8AAAAAAADwPwAAAAAAAAAAAAAAAAAA+H+amZmZmZmJrPwAAAAAAAPh/"
AAAAAAAA\+H8A\+nAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/
AAAAAAAA\+H8AAAAAAD4fwAAAAAAPh/AAAAAAAA+H8AAAAAAD4fwAAAAAAPh/

,"quad":"AAAAAAAA8L8AAAAAAADgP5qZmZmZmJrPwAAAAAAAPh/",
"params":{"answer_mode":"histogram","num_reads":10}}
'}
```

Problem Data

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>solver</td>
<td>Solver to be used.</td>
</tr>
<tr>
<td>data</td>
<td>Problem data; see the Data Encoding section.</td>
</tr>
<tr>
<td>type</td>
<td>One of the supported problem types (ising or qubo).</td>
</tr>
<tr>
<td>params</td>
<td>Solver-specific parameters. See Data Encoding for Solver Parameters.</td>
</tr>
</tbody>
</table>
Note: The allowed problem types are listed in the solver’s properties.

Response

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

Error in problem (example):

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

Table 3.4: Key-Value Pairs for Submit Problem Response with Error

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_code</td>
<td>Error code describing the type of error encountered; 400 for missing parameters, 403 for no access to solver, 429 for too many requests.</td>
</tr>
<tr>
<td>error_msg</td>
<td>Message describing the error encountered</td>
</tr>
</tbody>
</table>

Successful completion (example):

```json
{
  "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": "GgAAAA==",
    "active_variables": "AAAAAAEAAAAAABAAAAA==",
    "solutions": "sA==",
    "timing": {}
  },
  "latest_estimated_completion": "2018-08-13T20:47:03.009543Z",
  "type": "ising",
  "id": "d8b0a7d0-01e5-4d27-bae2-93cd23f3204c"
}
```

Completion with errors (example):

```json
{
  "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-4e3acb831b",
"error_message": "Some error message"}

Table 3.5: Key-Value Pairs for Submit Problem Responses

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>answer</td>
<td>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.</td>
</tr>
<tr>
<td>earliest_estimated_completion</td>
<td>Estimated completion time: beginning of range.</td>
</tr>
<tr>
<td>error_message</td>
<td>Error message generated by the system if problem status is FAILED.</td>
</tr>
<tr>
<td>id</td>
<td>Unique identifier of the problem; can be used later to retrieve problem information, the answer, and the messages.</td>
</tr>
<tr>
<td>latest_estimated_completion</td>
<td>Estimated completion time: end of range.</td>
</tr>
<tr>
<td>solved_on</td>
<td>If this problem is in terminal state (COMPLETED, CANCELLED or FAILED), time when problem was solved. Empty otherwise.</td>
</tr>
<tr>
<td>solver</td>
<td>Name of the solver to which the problem was submitted.</td>
</tr>
<tr>
<td>status</td>
<td>One of the problem states as defined in Problem Lifecycle.</td>
</tr>
<tr>
<td>submitted_on</td>
<td>Time when problem was submitted.</td>
</tr>
<tr>
<td>type</td>
<td>Problem type as specified when the problem was submitted.</td>
</tr>
</tbody>
</table>
3.1.3 Cancel Multiple Problems

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
```

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

Table 3.6: Response Codes for Cancel Problem

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_code</td>
<td>Error code describing the type of error encountered</td>
</tr>
<tr>
<td>error_msg</td>
<td>Message describing the error encountered</td>
</tr>
</tbody>
</table>

Table 3.7: Key-Value Pairs for Cancel Problem Response with Error
Successful cancellation (example):

```json
{
    "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 3.8: Key-Value Pairs for Cancel Problem Response**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Unique identifier of the problem.</td>
</tr>
<tr>
<td>status</td>
<td>Problem state: CANCELLED.</td>
</tr>
<tr>
<td>submitted_on</td>
<td>The time when problem was submitted.</td>
</tr>
<tr>
<td>solved_on</td>
<td>The time when problem was cancelled.</td>
</tr>
<tr>
<td>type</td>
<td>Problem type as specified when the problem was submitted.</td>
</tr>
</tbody>
</table>
3.1.4 Cancel a Problem by ID

**Request**

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

$SAPI_HOME/problems/<problem_id>/

**Response**

The system returns the problem formatted as a JSON object.

Example of the output:

```json
{
    "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>
<thead>
<tr>
<th>Response Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Success.</td>
</tr>
<tr>
<td>202</td>
<td>Cancellation request received. Relevant for in-progress jobs. The problem is cancelled if the request was received in time; otherwise, it will complete.</td>
</tr>
<tr>
<td>404</td>
<td>Problem with the specified ID does not exist.</td>
</tr>
<tr>
<td>409</td>
<td>Problem reached one of terminal states before the call.</td>
</tr>
<tr>
<td>429</td>
<td>Number of API requests exceeds the permitted limit.</td>
</tr>
</tbody>
</table>
3.1.5 List Problems

Request

To retrieve a list of previously submitted problems, make an HTTP GET request to $SAPI_HOME/problems/ with a parameter id that contains a comma-separated list of problem IDs to SAPI URI:

```
$SAPI_HOME/problems/?id=0b759042-726e-4665-b04b-3b42e1ed0b01,
780387b7a-0976-43da-8ed0-e7a27821c177
```

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": "780387b7a-0976-43da-8ed0-e7a27821c177"
   }
]
```

Response codes are as follows:

<table>
<thead>
<tr>
<th>Response Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Response contains valid problems list.</td>
</tr>
<tr>
<td>404</td>
<td>Problem with specified ID does not exist.</td>
</tr>
<tr>
<td>429</td>
<td>Number of API requests exceeds the permitted limit.</td>
</tr>
</tbody>
</table>
# 3.1.6 Get Problem

## Request

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

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

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

## 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_06",
    "submitted_on": "2018-01-18T10:06:10.025064",
    "type": "ising",
    "id": "894fb8e7-4176-4b39-92f4-10ad56432f09"
}
```

Response codes are as follows:

<table>
<thead>
<tr>
<th>Response Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Success</td>
</tr>
<tr>
<td>404</td>
<td>Problem with specified ID does not exist.</td>
</tr>
<tr>
<td>429</td>
<td>Number of API requests exceeds the permitted limit.</td>
</tr>
</tbody>
</table>

---

Table 3.11: Response Codes for Get Problem
3.1.7 Get Problem Answer

Request

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

$SAPI_HOME/problems/<problem_id>/answer/

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": "AwAAAAIAAAAAACAAAAAwAAAA==",
        "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>
<thead>
<tr>
<th>Response Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Success.</td>
</tr>
<tr>
<td>404</td>
<td>Answer for the problem does not exist.</td>
</tr>
<tr>
<td>429</td>
<td>Number of API requests exceeds the permitted limit.</td>
</tr>
</tbody>
</table>
3.1.8 Get Problem Messages

Request

To retrieve messages for a problem, make an HTTP GET request to $\$SAPI\_HOME\/problems/.

$\$SAPI\_HOME\/problems\/<\$problem\_id\>/messages/

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>
<thead>
<tr>
<th>Response Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Success.</td>
</tr>
<tr>
<td>404</td>
<td>Problem does not exist.</td>
</tr>
<tr>
<td>429</td>
<td>Number of API requests exceeds the permitted limit.</td>
</tr>
</tbody>
</table>
3.2 Solvers Resource

A solver instance represents a logical solver that can be one of:

- QPAPI solver
- Subgraph solver

3.2.1 Solver Resource Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Description of the solver.</td>
</tr>
<tr>
<td>property</td>
<td>Solver properties such as supported problem types, active qubits, active couplers, total number of qubits, and so on.</td>
</tr>
<tr>
<td>id</td>
<td>Unique ID of the solver.</td>
</tr>
</tbody>
</table>

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*. 
3.2.2 List Solvers

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
```

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"
    }
]
```
3.2.3 Get Solver Configuration

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
```

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>
<thead>
<tr>
<th>Response Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Success</td>
</tr>
<tr>
<td>404</td>
<td>Solver with specified solver_id does not exist.</td>
</tr>
<tr>
<td>429</td>
<td>Number of API requests exceeds the permitted limit.</td>
</tr>
</tbody>
</table>
This section describes the data encoding for problems and solutions.

## 4.1 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.

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ising</td>
<td>ising</td>
</tr>
<tr>
<td>QUBO</td>
<td>qubo</td>
</tr>
</tbody>
</table>

### 4.1.1 JSON Encoding

Problems are encoded as JSON objects with the following properties:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>String: <code>qp</code></td>
</tr>
<tr>
<td>lin</td>
<td>Linear coefficients (base64-encoded little-endian doubles). One value per working qubit in the same order as solver’s <code>qubits</code> property; NaN indicates an inactive qubit.</td>
</tr>
<tr>
<td>quad</td>
<td>Quadratic coefficients (base64-encoded little-endian doubles). One value per active coupler in the same order as solver’s <code>couplers</code> property. An active coupler means that both qubits the coupler is incident to are active. NaN values are not permitted.</td>
</tr>
</tbody>
</table>
An example of an HTTP POST request for submitting a problem using curl in JSON encoding is as follows:

```bash
curl -H "X-Auth-Token: $SAPI_TOKEN" $SAPI_HOME/problems -X POST -d '{"type":"ising","solver":"c4-sw_sample","data":
  [{"type":"ising","solver":"c4-sw_sample","data":{"lin":"AAAAAAAA4L8AAAAAAADwPwAAAAAAAAAAAAAAAAAA+H+amZmZmZmJwAAAAAAAPh/"},
  "params":{"answer_mode":"histogram","num_reads":10}}]
```
4.2 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>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>anneal_offsets</code></td>
<td>JSON array of length equal to the solver property <code>anneal_offset_ranges</code></td>
</tr>
<tr>
<td><code>anneal_schedule</code></td>
<td>JSON array of two element arrays of floating-point numbers</td>
</tr>
<tr>
<td><code>annealing_time</code></td>
<td>Integer</td>
</tr>
<tr>
<td><code>answer_mode</code></td>
<td>[“raw”</td>
</tr>
<tr>
<td><code>auto_scale</code></td>
<td>[true</td>
</tr>
<tr>
<td><code>beta</code></td>
<td>Float</td>
</tr>
<tr>
<td><code>chains</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>flux_biases</code></td>
<td>JSON array of floating point numbers with length equal to the <code>num_qubits</code> property of the solver. Use 0 for unused or unavailable devices.</td>
</tr>
<tr>
<td><code>flux_drift_compensation</code></td>
<td>[true</td>
</tr>
<tr>
<td><code>initial_state</code></td>
<td>JSON array of length equal to <code>num_qubits</code>, 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.</td>
</tr>
<tr>
<td><code>max_answers</code></td>
<td>Positive integer</td>
</tr>
<tr>
<td><code>num_reads</code></td>
<td>Positive integer</td>
</tr>
<tr>
<td><code>num_spin_reversal_transforms</code></td>
<td>Zero or a positive integer</td>
</tr>
<tr>
<td><code>postprocess</code></td>
<td>[“”</td>
</tr>
<tr>
<td><code>programming_thermalization</code></td>
<td>Integer bounded by solver property <code>programming_thermalization_range</code></td>
</tr>
<tr>
<td><code>readout_thermalization</code></td>
<td>Integer bounded by solver property <code>readout_thermalization_range</code></td>
</tr>
<tr>
<td><code>reduce_intersample_correlation</code></td>
<td>[true</td>
</tr>
<tr>
<td><code>reinitialize_state</code></td>
<td>[true</td>
</tr>
</tbody>
</table>
4.3 Data Encoding for Solutions

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>String: <code>qp</code></td>
</tr>
<tr>
<td>num_variables</td>
<td>Total number of variables (active or otherwise) that the solver has. JSON integer.</td>
</tr>
<tr>
<td>solutions</td>
<td>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.</td>
</tr>
<tr>
<td>energies</td>
<td>Base-64–encoded string of energies, each a little-endian 8-byte floating-point number (doubles).</td>
</tr>
<tr>
<td>active_variables</td>
<td>Base-64–encoded string of the indices of the problem’s active variables. The indices are 4-byte little-endian integers.</td>
</tr>
<tr>
<td>num_occurrences</td>
<td>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.</td>
</tr>
<tr>
<td>timing</td>
<td>Solver-specific JSON object describing the reported time that the solver took to handle the problem.</td>
</tr>
</tbody>
</table>

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.