Version 3.3.0-dev | Heliophysics Data and Model Consortium (HDMC) |

This is the development version of the HAPI Data Access Specification.

The most recent stable release is Version 3.3.0.

1 Significant Changes to Specification

Note that version 3.0 introduced some breaking changes. They are listed here and are documented as issues in the 3.0 Milestone list.

1.1 v2 to v3 API Changes

Non-backward compatible changes introduced to the request interface in HAPI 3.0:

1. The URL parameter `id` was replaced with `dataset`.

2. `time.min` and `time.max` were replaced with `start` and `stop`, respectively.

3. Addition of a new endpoint, "about", for server description metadata.

These changes were discussed in issue #77. HAPI 3 servers must accept both the old and these new parameter names, but the HAPI 2 specification requires an error response if the new URL parameter names are used. In a future version, the deprecated older names will no longer be valid.

1.2 v2 to v3 Schema Changes

This list captures new optional elements that can be included in the `info` response as of 3.0:

1. Ability to specify time-varying bins (#83)

2. Ability to use JSON references in `info` response (#82)

3. Ability to indicate a units schema (if one is being used for `units` strings) (#81)

1.3 3.X changes

3.0, 3.1, 3.2, and 3.3 changes are documented in the changelog. We follow Semantic Versioning conventions, so all 3.X versions are backward compatible with version 3.0.

2 Introduction

2.1 Overview

This document describes the Heliophysics Application Programmer's Interface (HAPI) specification, which is an API, metadata, and data streaming format specification for time-series data. The intent of this specification is to enhance interoperability among time series data providers. The objective of this specification is to capture features available from many existing data providers and to codify implementation details so that providers can use a common API. This will make it possible to obtain

time series science data content seamlessly from many sources and use a variety of programming languages.

This document is intended to be used by two groups of people: first by data providers who want to make time-series data available through a HAPI server, and second by data users who want to understand how data is made available from a HAPI server, or perhaps to write client software to obtain data from an existing HAPI server.

HAPI constitutes a minimum but complete set of capabilities needed for a server to allow access to the time series data values within one or more data collections. Because of its focus on data access, the HAPI metadata standard is not intended for complex search and discovery. However, the metadata schema allows ways to indicate where further descriptive details for any dataset could be found and the metadata contains enough information to enable its use by complex search and discovery software.

The HAPI API is based on REpresentational State Transfer (RESTful) principles, which emphasize that URLs are stable endpoints through which clients can request data. Because it is based on well-established HTTP request and response rules, a wide range of HTTP clients can be used to interact with HAPI servers.

Key definitions for terms used in this document include

- **parameter** – a measured science quantity or a related ancillary quantity at one instant in time; may be scalar or a multi-dimensional array as a function; must have units and must have a fill value that indicates no measurement was available or absent information
- **record** – all the parameters and an associated time value
- **dataset** – a collection of records with the same parameters; a HAPI service presents a dataset as a seamless collection of time- records such that it or a subset of it can be retrieved without knowledge of the actual storage details
- **catalog** - a collection of datasets
- **request parameter** – keywords that appear after the `?` in a URL

  For example, in the request (see change notes)

  `http://server/hapi/data?dataset=alpha&start=2016-07-13&stop=2016-07-14`

  the three request parameters are `dataset` (corresponding to the identifier of the dataset), `start`, and `stop`. They have values of `alpha`, `2016-07-13`, and `2016-07-14`, respectively. This document will always use the full phrase "request parameter" to refer to these URL elements to distinguish them from the parameters in a dataset.

  In the above URL, the segment represented as `server` captures the hostname for the HAPI server as well as any prefix path elements before the required `hapi` element. For example, in `http://example.com/public/data/hapi`, the `server` element is `example.com/public/data`.

2.2 Facilitating Adoption

The text from this section was taken from Weigel et al., 2021.

The HAPI specification is designed to provide a standardized way to provide streaming services that many time-series data providers already offer. The use of the common aspects of existing features of HP data providers means that data providers can, with minimal effort, modify existing streaming servers to make them HAPI compliant.

In addition, several tools have been developed to facilitate adoption.

1. Although the HAPI streaming formats are simple enough that only 10-20 lines of code are needed to read a basic HAPI response, there are error conditions and other optimizations to consider (such as caching and parallelization of requests); a robust and comprehensive client implementation is beyond what should be expected of science users or even software developers who want to create tools that use HAPI data. Mature HAPI server client libraries are available from open-source HAPI projects for Java, IDL, MATLAB, and Python, so most users can begin with these libraries. These clients read a HAPI response into a data structure appropriate for the given language (e.g., in Python, the response is read into a NumPy N-D array with timestamps converted to Python `datetime` objects). Work is underway to incorporate HAPI readers into existing popular analysis frameworks, such as SPEDAS in IDL and Python, Autoplot in Java, SunPy in Python, and SpacePy in Python.

2. To assist server developers with adoption, a cross-platform reference server has been developed. With this server, a data provider only needs to provide HAPI JSON metadata and a command-line program that emits HAPI CSV given inputs of a dataset identifier and start/stop times. The server handles HAPI error responses, request validation, and logging.

3. A validator/verifier has been developed that runs many tests on a server. The server executes a comprehensive suite of tests for compliance with the specification. By either downloading and running the test software locally, or by entering the URL of a HAPI server into a website, data providers can test most aspects of their HAPI implementation, with detailed results indicating warnings or errors. This has proven to be a very effective way to assist developers, and it helps ensure that new HAPI servers are fully functional. The tests include checks of JSON responses against the HAPI metadata schema, verifying that the server handles different allowed start/stop time representations (e.g, year, day-of-year and year, month day with varying amounts of additional time precision), error responses and message, timeouts, and testing that output is independent of the output format. Also, the validator/verifier makes recommendations. For example, if a server does not include cross-origin request sharing headers or respond with compressed data when gzip is specified in the `Accept-Encoding` request header, it emits a message noting their advantages and a link to information on how to enable or implement these features.

2.3 Limitations

Because HAPI requires a single time column to be the first column, each record (one row of data) must be associated with one time value (the first value in the row). This has implications for serving files with multiple time arrays. Suppose a file contains 1-second data, 3-second data, and 5-second data, all from the same measurement but averaged differently. A HAPI server could expose this data, but not as a single dataset. To a HAPI server, each time resolution could be presented as a separate dataset, each with its unique time array.

Note that there are only a few supported data types: `isotime`, `string`, `integer`, and `double`. This is intended to keep the client code simple in terms of dealing with the data stream. However, the spec may be expanded to include other types, such as 4-byte floating-point values (which would be called float), or 2-byte integers (which would be called short).

3 Endpoints

3.1 Overview

The HAPI specification has five required endpoints that give clients a precise way to first determine the data holdings of the server and then to request data. The functionality of the required endpoints is as follows:

1. `/hapi/capabilities` lists the output formats the server can stream (`csv`, `binary`, or `json`, described below).

2. `/hapi/about` lists the server id and title, contact information, and a brief description of the datasets served (this endpoint is new in the version 3 HAPI specification).

3. `/hapi/catalog` lists the catalog of available datasets; each dataset is associated with a unique id and may optionally have a title.

4. `/hapi/info` lists information about a dataset with a given id; a primary component of the description is the list of parameters in the dataset.

5. `/hapi/data` streams data for a dataset of a given id and over a given time range; a subset of parameters in a dataset may be requested (default is all parameters).

There is also an optional landing page endpoint `/hapi` that returns human-readable HTML. Although there is recommended content for this landing page, it is not essential to the functioning of the server.

Servers may have non-standard endpoints (an endpoint not given above) under `/hapi/` if they are prefixed by `x_`, e.g., `/hapi/x_custom_endpoint`. This will communicate to the user that this is not a standard endpoint and will prevent a name conflict if the HAPI standard adds `custom_endpoint` to the specification.

The five required endpoints are RESTful-style in that the resulting HTTP response is the complete response for each endpoint. In particular, the `/data` endpoint does not give URLs for file or links to where the data can be downloaded; instead, it streams the data in the HTTP response body. The full specification for each endpoint is described below.

All endpoints must have a `/hapi` path element in the URL, and only the `/info` and `/data` endpoints take query parameters:

```
http://server/hapi (Optional HTML landing page)                          1
http://server/hapi/capabilities                                          2
http://server/hapi/about                                                 3
http://server/hapi/catalog                                               4
http://server/hapi/info?dataset=...[&...]                                5
http://server/hapi/data?dataset=...&...                                  6
```

We recommend that any requests with a trailing slash are handled by sending a `301` redirect, e.g.,

```
http://server/hapi/ => 301 redirect to http://server/hapi
http://server/hapi/info/?dataset=... => 301 redirect to http://server/hapi/info?
dataset=...
```

Requests to a HAPI server must not change the server state. Therefore, all HAPI endpoints must respond only to HTTP HEAD and GET requests.

The request parameters and their allowed values must be strictly enforced by the server. **HAPI servers must not add request parameters beyond those in the specification.** If a request URL contains any unrecognized or misspelled request parameters, a HAPI server must respond with an error status (see HAPI Status Codes for more details). The principle being followed is that the server must not silently ignore unrecognized request parameters because this would falsely indicate to clients that the request

parameter was understood and was taken into account when creating the output. That is, if a server is given a request parameter that is not part of the HAPI specification, such as `averagingInterval=5s`, the server must report an error for two reasons: 1. additional request parameters are not allowed, and 2. the server will not do any averaging.

The outputs from a HAPI server to the `about`, `catalog`, `capabilities`, and `info` endpoints are JSON objects, the formats of which are described below in the sections detailing each endpoint. The `data` endpoint must be able to deliver Comma Separated Value (CSV) data following the RFC 4180 standard [1], but may optionally deliver data content in binary format or JSON format. The response stream formats are described in the Data Stream Content section.

The following is the detailed specification for the five main HAPI endpoints as well as the optional landing page endpoint.

3.2 `hapi`

This root endpoint is optional and should provide a human-readable landing page for the server. Unlike the other endpoints, there is no strict definition for the output, but if present, it should include a brief description of the data and other endpoints and links to documentation on how to use the server. An example landing page that can be easily customized for a new server is in the Appendix.

There are many options for landing page content, such as an HTML view of the catalog or links to commonly requested data.

Sample Invocation

> `http://server/hapi`

Request Parameters

None

Response

The response is in HTML format with a mime type of `text/html`. There is no specification for the content but should provide an overview that is useful for science users.

Example

Retrieve the landing page for this server.

> `http://server/hapi`

Example Response

See the Appendix.

3.3 `about`

Sample Invocation

> `http://server/hapi/about`

Request Parameters

None

Response

The server's response to this endpoint must be in JSON format [3] as defined by RFC-7159, and the response must indicate a mime type of `application/json`. Server attributes are described using keyword-value pairs, with the required and optional keywords described in the following table.

About Object

Name	Type	Description
HAPI	string	**Required** The version number of the HAPI specification this description complies with.
status	Status object	**Required** Server response status for this request.
id	string	**Required** A unique ID for the server. Ideally, this ID has the organization name in it, e.g., NASA/SPDF/SSCWeb, NASA/SPDF/CDAWeb, INTERMAGNET, UIowa/RPWG, LASP/TSI, etc. A list of known ids is available in the servers repository.
title	string	**Required** A short human-readable name for the server that defines an acronym in the `id` or provided additional context about data served (e.g., LASP Total Solar Irradiance (TSI) data). The suggested maximum length is 40 characters. We recommend against including "HAPI" and/or the HAPI version of the server in the title.
contact	string	**Required** Contact information or email address for server issues. HAPI clients should show this contact information when it is certain that an error is due to a problem with the server (as opposed to the client). Ideally, a HAPI client will recommend that the user check their connection and try again at least once before contacting the server contact.
description	string	**Optional** A brief description of the type of data the server provides.
contactID	string	**Optional** The identifier in the discovery system for information about the contact. For example, a SPASE ID of a person identified in the `contact` string.
resourceID	string	**Optional** An identifier associated with all datasets.
citation	string	**Optional** Deprecated; use `serverCitation`.
serverCitation	string	**Optional** How to cite the HAPI server. An actionable DOI is preferred (e.g., https://doi.org/…). This `serverCitation` differs from the `dataCitation` in an `/info` response. Here the

Name	Type	Description
		`serverCitation` is for the entity that maintains the data server.
note	string or array of strings	**Optional** General notes about the server that are not appropriate to include in `description`. For example, a change log that lists added datasets or parameters in datasets. If an array of strings is used to describe datestamped notes, we recommend prefixing the note with a HAPI restricted ISO 8601 format, e.g., `["2024-01-01T00: Note on this date/time", "2024-04-02T00: Note on this date/time"].`
warning	string or array of strings	**Optional** Temporary warnings about the data server, such as scheduled downtime and known general problems. Dataset-specific warnings should be placed in the `warning` element of the `/info` response.
dataTest	DataTest	**Optional** Information that a client can use to check that a server is operational. Data response should contain more than zero records. See below for the definition of this object.

`DataTest` Object

Name	Type	Description
query	Query	**Required** See the `Query` object definition below.
name	string	**Optional** Name of the test.

A server test query then has the following definition:

`Query` Object

Name	Type	Description
dataset	string	**Required** String identifier for the dataset to be used in the test.
start	string	**Required** String value of test data start time in restricted ISO 8601 format.
stop	string	**Required** String value of test data stop time in restricted ISO 8601 format.
parameters	string	**Optional** A comma-separated list of parameters to include in the response (allowed characters). If not provided, all parameters will be included.

Example

```
http://server/hapi/about
```

Example Response:

```
{                                                            1
  "HAPI": "3.3",                                             2
  "status": {"code": 1200, "message": "OK"},                 3
  "id": "TestData3.3",                                       4
  "title": "HAPI 3.3 Test Data and Metadata",                5
  "contact": "example1@example.org",                         6
  "dataTest": {                                              7
                "name": "Ping Test",                         8
                "query": {                                   9
                      "dataset": "dataset1",                 10
                      "start": "2023-01-01T12:00:00Z",       11
                      "stop": "2023-01-01T14:00:01Z",        12
                      "parameters": "parameter1,parameter2"  13
                }                                            14
        }                                                    15
}                                                            16
```

3.4 `capabilities`

This endpoint describes relevant implementation capabilities for this server. Currently, the only possible variability from server to server is the list of output formats that are supported.

A server must support the `csv` output format, but `binary` output format and `json` output may optionally be supported. Servers may support custom output formats, which would be advertised here. All custom formats listed by a server must begin with the string `x_` to indicate that they are custom formats and avoid naming conflicts with possible future additions to the specification.

Sample Invocation

```
http://server/hapi/capabilities
```

Request Parameters

None

Response

The server's response to this endpoint must be in JSON format [3] as defined by RFC 7159, and the response must indicate a mime type of `application/json`. Server capabilities are described using keyword-value pairs, with `outputFormats` being the only keyword currently in use.

Capabilities Object

Name	Type	Description
HAPI	string	**Required** The version number of the HAPI specification this description complies with.

Name	Type	Description
status	Status object	**Required** Server response status for this request.
outputFormats	string array	**Required** The list of output formats the server can emit. All HAPI servers must support at least `csv` output format, with `binary` and `json` output formats being optional. Any custom formats not in this list must begin with a prefix of `x_` as shown in the example below.
catalogDepthOptions	string array	**Optional** A list of options for `/catalog?depth=` requests. It can include one or more of `dataset` and `all`. See the catalog endpoint description.

Example

Retrieve a listing of the capabilities of this server.

```
http://server/hapi/capabilities
```

Example Response:

```
{                                                               1
  "HAPI": "3.3",                                                2
  "status": {"code": 1200, "message": "OK"},                    3
  "outputFormats": ["csv", "binary", "json", "x_hdf"]           4
}                                                               5
```

Note the non-standard format of HDF data, and that it has the required `x_` prefix.

If a server only reports an output format of `csv`, then requesting `binary` data should cause the server to respond with an error status of `1409 "Bad request - unsupported output format"` with a corresponding HTTP response code of 400. See below for more about error responses.

3.5 `catalog`

This endpoint provides a list of datasets available from the server.

Sample Invocation

```
http://server/hapi/catalog
```

Request Parameters

Name	Description
depth	**Optional** Possible values are `dataset` (the default) and `all`. Servers may choose to implement the `all` option, which allows all of the metadata from a server to be obtained in a single request. If this request parameter is supported, the `/capabilites` end point must return `catalogDepthOptions=["dataset", "all"]`.

Name	Description
resolve_references	**Optional** If `false,` allow the server to send responses with references that need resolution by the client. Default is `true`. See 3.6.13 JSON References.
Response	

The response is in JSON format [3] as defined by RFC-7159 and has a MIME type of `application/json`. The catalog is a simple listing of identifiers for the datasets available from the server. Additional metadata about each dataset is available through the `info` endpoint (described below). The catalog takes no query parameters and always lists the full catalog.

Catalog Object

Name	Type	Description
HAPI	string	**Required** The version number of the HAPI specification this description complies with.
status	object	**Required** Server response status for this request. (see HAPI Status Codes)
catalog	array of Dataset	**Required** A list of datasets available from this server.

Dataset Object

Name	Type	Description
id	string	**Required** The computer-friendly identifier (allowed characters) that the host system uses to locate the dataset. Each identifier must be unique within the HAPI server where it is provided.
title	string	**Optional** A short label for the dataset usable in selecting a dataset from a list. If none is given, it defaults to the id. The suggested maximum length is 40 characters.
info	object	**Optional** but required when `depth=all` is use in request. This object should be identical in content to what is returned by a `/info?dataset=id` request. The `HAPI` and `status` nodes should be omitted inside the `info` objects. (The `status` does not belong there, and the `HAPI` version should be the same for all `info` elements, so it is redundant information.)

The identifiers must be unique within a single HAPI server. Also, dataset identifiers in the catalog should be stable over time. Including rapidly changing version numbers or other revolving elements (dates, processing ids, etc.) in the datasets identifiers should be avoided. The intent of the HAPI specification is to allow data to be referenced using RESTful URLs with a reasonable lifetime.

Identifiers must be limited to the set of characters, including upper and lower case letters, numbers, and the following characters: comma, colon, slash, minus, and plus. See 89 for a discussion of this.

Example

Retrieve a listing of datasets available from a server.

```
http://server/hapi/catalog
```

Example Response:

```
{                                                                    1
  "HAPI" : "3.3",                                                    2
  "status": {"code": 1200, "message": "OK"},                        3
  "catalog":                                                         4
          [                                                          5
             {"id": "ACE_MAG", title:"ACE Magnetometer data"},      6
             {"id": "data/IBEX/ENA/AVG5MIN"},                        7
             {"id": "data/CRUISE/PLS"},                              8
             {"id": "any_identifier_here"}                           9
          ]                                                         10
}                                                                   11
```

Example

Retrieve a listing of all the datasets and embed the `info` metadata for each in the dataset object.

First, verify that this is supported:

```
http://server/hapi/capabilities
```

```
{                                                                    1
  "HAPI": "3.3",                                                     2
  "status": {"code": 1200, "message": "OK"},                         3
  "outputFormats": ["csv", "binary", "json"],                        4
  "catalogDepthOptions": ["dataset", "all"]                          5
}                                                                    6
```

The `/capabilities` response indicates that the `/catalog` endpoint allows the `depth=all` option. Then, a subsequent request for the entire catalog will succeed:

```
http://server/hapi/catalog?depth=all
```

Example Response:

(where . . . is a placeholder for additional metadata that has been omitted for clarity)

```
{                                                                            1
   "HAPI" : "3.3",                                                           2
   "status": {"code": 1200, "message": "OK"},                               3
   "catalog":                                                                4
           [                                                                 5
               {                                                             6
                "id": "MAG1", title:"Magnetometer data",                     7
                "info": {                                                    8
                  "startDate": "2010-01-01T00:00:00Z",                       9
                  "stopDate": "2020-01-02T00:00:00Z",                       10
                  "parameters": [{"name": "Time", ...}]                     11
                  ...                                                       12
                 }                                                          13
               },                                                          14
               {                                                           15
                "id": "MAG2", title:"Magnetometer data",                   16
                "info": {                                                  17
                  "startDate": "2000-01-02T00:00:00Z",                     18
                  "stopDate": "2010-01-01T00:00:00Z",                      19
                  "parameters": [{"name": "Time", ...}]                    20
                  ...                                                      21
                 }                                                         22
               },                                                          23
               {                                                           24
                  ...                                                      25
                 }                                                         26
           ]                                                               27
}                                                                          28
```

3.6 `info`

This endpoint provides a data header for a given dataset. The header is expressed in JSON format [3] as defined by RFC-7159 and has a MIME type of `application/json`. The specification for the header is that it provides a minimal amount of metadata that allows for the automated reading by a client of the data content streamed via the `data` endpoint. The header must include a list of the parameters in the dataset and the date range covered by the dataset. There are also optional metadata elements for capturing other high-level information, such as a brief description of the dataset, the nominal cadence of the data, and ways to learn more about a dataset. The table below lists all required and optional dataset attributes in the header.

Servers may include additional custom (server-specific) keywords or keyword/value pairs in the header provided that the keywords begin with the prefix `x_`. While a HAPI server must check all request parameters (servers must return an error code given any unrecognized request parameter as described earlier), the JSON content output by a HAPI server may contain additional, user-defined metadata elements. All non-standard metadata keywords must begin with the prefix `x_` to indicate to HAPI clients

that these are extensions. Custom clients could use the additional keywords, but standard clients would ignore the extensions. By using the standard prefix, the custom keywords will not conflict with any future keywords added to the HAPI standard. Servers using these extensions may wish to include additional, domain-specific characters after the `x_` to avoid possible collisions with extensions from other servers.

Each parameter listed in the header must itself be described by specific metadata elements, and a separate table below describes the required and optional parameter attributes.

By default, the parameter list in the `info` response will include *all* parameters in the dataset. However, a client may request a header for just a subset of the parameters. The subset of interest is specified as a comma-separated list via the request parameter called `parameters`. (Note that the client would have to obtain the parameter names from a prior request.) There must not be any duplicates in the subset list, and the subset list must be arranged according to the ordering in the original, full list of parameters. The reduced header is useful because it is also possible to request a subset of parameters when asking for data (see the `data` endpoint), and a reduced header can be requested that would then match the subset of parameters in the data. This correspondence of reduced header and reduced data ensures that a data request for a subset of parameters can be properly interpreted even if additional subset requests are made with no header. (Although a way to write a client as safe as possible would be always to request the full header and rely on its parameter ordering to determine the data column ordering.)

Note that the `data` endpoint may optionally prepend the `info` header to the data stream when `include=header` is included in the request URL. In cases where the `data` endpoint response includes a header followed by `csv` or `binary` data, the header must always end with a newline. This enables the end of the JSON header to be more easily detected.

Sample Invocation

```
http://server/hapi/info?dataset=ACE_MAG
```

3.6.1 Request Parameters

Items with a * superscript in the following table have been modified from version 2 to 3; see change notes.

Name	Description
dataset *	**Required** The identifier for the dataset (allowed characters)
parameters	**Optional** A comma-separated list of parameters to include in the response (allowed characters). Default is all; `...&parameters=&...` in URL should be interpreted as meaning all parameters. See "subsetting parameters" in this subsection for additional examples.
resolve_references	**Optional** If `false`, allow the server to send responses with references that need resolution by the client. Default is `true`. See JSON References.

Response

The response is in JSON format [3] and provides metadata about one dataset.

Subsetting Parameters

Clients may request an `info` response that includes only a subset of the parameters or a data stream for a subset of parameters (via the `data` endpoint, described next). The logic on the server is the same for `info` and `data` requests in terms of what dataset parameters are included in the response. The primary time parameter (always required to be the first parameter in the list) is always included, even if not requested. These examples clarify the way a server must respond to various types of dataset parameter subsetting requests:

- **request:** do not ask for any specific parameters (i.e., there is no request parameter called `parameters`);
  example: `http://server/hapi/data?dataset=MY_MAG_DATA&start=1999Z&stop=2000Z`
  response: all columns

- **request:** ask for just the primary time parameter;
  example: `http://server/hapi/data?`
  `dataset=MY_MAG_DATA&parameters=Epoch&start=1999Z&stop=2000Z`
  response: just the primary time column

- **request:** ask for a single parameter other than the primary time column (like `parameters=Bx`);
  example: `http://server/hapi/data?`
  `dataset=MY_MAG_DATA&parameters=Bx&start=1999Z&stop=2000Z`
  response: primary time column and the one requested data column

- **request:** ask for two or more parameters other than the primary time column;
  example: `http://server/hapi/data?`
  `dataset=MY_MAG_DATA&parameters=Bx,By&start=1999Z&stop=2000Z`
  response: primary time column followed by the requested parameters in the order they occurred in the original, non-subsetted dataset header (note that the parameter ordering in the request must match the original ordering anyway - see just below)

Note that the order in which parameters are listed in the request must not differ from the order that they appear in the response. For a data set with parameters `Time,param1,param2,param3` this subset request

`?dataset=ID&parameters=Time,param1,param3`

is acceptable, because `param1` is before `param3` in the `parameters` array (as determined by the `/info` response). However, asking for a subset of parameters in a different order, as in

`?dataset=ID&parameters=Time,param3,param1`

is not allowed, and servers must respond with an error status. See HAPI Status Codes for more about error conditions and codes.

3.6.2 `info` Response Object

Dataset Attribute	Type	Description
HAPI	string	**Required** The version number of the HAPI specification with which this description complies.
status	object	**Required** Server response status for this request; see HAPI Status Codes.
format	string	**Required** (when the header is prefixed to data stream) Format of the data as `csv` or `binary` or `json`.

Dataset Attribute	Type	Description
parameters	array of Parameter	**Required** Description of the parameters in the data.
startDate	string	**Required** Restricted ISO 8601 date/time of first record of data in the entire dataset.
stopDate	string	**Required** Restricted ISO 8601 date/time of the last record of data in the entire dataset. For actively growing datasets, the end date can be approximate, but it is the server's job to report an accurate end date.
timeStampLocation	string	**Optional** Indicates the positioning of the timestamp within the measurement window. Must be one of `begin`, `center`, `end` or `other`. **If this attribute is absent, clients are to assume a default value of `center`,** which is meant to indicate the exact middle of the measurement window. A value of `other` indicates that the location of the time stamp in the measurement window is not known or these options cannot be used for an accurate description. See also HAPI convention notes. (Note: version 2.0 indicated that these labels were in all upper case. Starting with version 2.1, servers should use all lower case. Clients, however, should be able to handle both all upper case and all lower case versions of these labels.)
cadence	string	**Optional** Time difference between records as an ISO 8601 duration. This is meant as a guide to the nominal cadence of the data and not a precise statement about the time between measurements. See also HAPI convention notes.
sampleStartDate	string	**Optional** Restricted ISO 8601 date/time of the start of a sample time period for a dataset, where the time period must contain a manageable, representative example of valid, non-fill data. **Required** if `sampleStopDate` given.
sampleStopDate	string	**Optional** Restricted ISO 8601 date/time of the end of a sample time period for a dataset, where the time period must contain a manageable, representative example of valid, non-fill data. **Required** if `sampleStartDate` given.
maxRequestDuration	string	**Optional** An ISO 8601 duration indicating the maximum duration for a request. This duration should be interpreted by clients as a limit above which a request for all parameters will very likely be

Dataset Attribute	Type	Description
		rejected with a HAPI 1408 error; requests for fewer parameters and a longer duration may or may not be rejected.
description	string	**Optional** A detailed description of the dataset content, caveats, relationships to other data, references and links – the information users need to know for a basic interpretation of the data. Suggested length is a few lines of text, e.g., 1000 characters; extended details can be referenced with links.
unitsSchema	string	**Optional** The name of the units convention that describes how to parse all `units` strings in this dataset. Currently, the only allowed values are: `udunits2`, `astropy3`, and `cdf-cluster`. See `unitsSchema` Details for additional information about these conventions. The list of allowed unit specifications is expected to grow to include other well-documented unit standards.
coordinateSystemSchema	string	**Optional** The name of the schema or convention that contains a list of coordinate system names and definitions. If this keyword is provided, any `coordinateSystemName` keyword given in a parameter definition should follow this schema. See `coordinateSystemSchema` Details for additional information.
location	object	**Optional** A way to specify where a dataset's measurements were made. See location and geoLocation Details for an explanation of the two ways to indicate location: one for a fixed location and one for when the measurement location changes with time.
geoLocation	array of double	**Optional** A simplified, compact way of indicating a fixed location for a dataset in Earth-based coordinates. The array has 2 or 3 elements: [longitude in degrees, latitude in degrees, (optional) altitude in meters]. For information and examples, see location and geoLocation Details.
resourceURL	string	**Optional** URL linking to more detailed information about this dataset.
resourceID	string	**Optional** An identifier by which this data is known in another setting (e.g., DOI)
creationDate	string	**Optional** Restricted ISO 8601 date/time of the

Dataset Attribute	Type	Description
		dataset creation.
citation	string	**Optional** Deprecated; use `datasetCitation`.
datasetCitation	string	**Optional** How to cite the data set. An actionable DOI is preferred (e.g., https://doi.org/…). Note that there is a `serverCitation` in an `/about` response for citing the data server, but `datasetCitation` is for the dataset citation, which is typically different.
licenseURL	string or array of strings	**Optional** A URL or array of URLs to a license landing page. If the license is in the spdx.org list, provide a link to it. A list of licenses implies that users may choose any one from the list.
provenance	string	**Optional** A description of the provenance of this dataset.
modificationDate	string	**Optional** Restricted ISO 8601 date/time of the last modification of metadata and/or data has changed. (Use this to signal that any content of the dataset in the full time range of data has changed; more granularity may be possible in the future, see (issue 218)[https://github.com/hapi-server/data-specification/issues/218].)
contact	string	**Optional** Relevant contact person name (and possibly contact information) for science questions about the dataset.
contactID	string	**Optional** The identifier in the discovery system for information about the contact. For example, the SPASE ID or ORCID of the person.
additionalMetadata	object	**Optional** A way to include a block of other (non-HAPI) metadata. See below for a description of the object, which can directly contain the metadata or point to it via a URL.
definitions	object	**Optional** An object containing definitions that are referenced using a JSON reference
note	string or array of strings	**Optional** General notes about the dataset that are inappropriate to include in `description`. For example, a change log that lists added parameters. If an array of strings is used to describe datestamped notes, we recommend prefixing the note with a HAPI restricted ISO 8601 format, e.g., `["2024-01-01T00: Note on this date time", "2024-04-02T00: Note on this date time"]`.

Dataset Attribute	Type	Description
`warning`	string or array of strings	**Optional** Temporary warnings about the dataset, such as "dataset stopDate is typically updated continuously, but …"

3.6.3 `unitsSchema` Details

One optional attribute is `unitsSchema`. This allows a server to specify, for each dataset, what convention is followed for the `units` strings in the parameters of the dataset. Currently, the only allowed values for `unitsSchema` are: `udunits2`, `astropy3`, and `cdf-cluster`. These represent the currently known set of unit conventions with software available for parsing and interpreting unit strings. Only major version numbers (if available) are indicated in the convention name. It is expected that this list will grow over time as needed. The current locations of the official definitions and software tools for interpreting the various unit conventions are in the following table:

Convention Name	Current URL	Description (context help if link is broken)
`udunits2`	https://www.unidata.ucar.edu/software/udunits	Unidata from UCAR; a C library for units of physical quantities
`astropy3`	https://docs.astropy.org/en/stable/units/	package inside `astropy` that handles defining, converting between, and performing arithmetic with physical quantities, such as meters, seconds, Hz, etc
`cdf-cluster`	https://caa.esac.esa.int/documents/DS-QMW-TN-0010.pdf which is referenced on this page: https://www.cosmos.esa.int/web/csa/documentation	conventions created and used by ESA's Cluster mission
`vounits1.1`	https://www.ivoa.net/documents/VOUnits/	IVOA Recommendation for Units in the VO

3.6.4 `coordinateSystemSchema` Details

The `parameter` object (described below) allows for a `coordinateSystemName` to be associated with any parameter. To allow a precise and computer-readable meaning to these coordinate system names, a dataset can have a `coordinateSystemSchema`. This schema should contain a computer-readable and publically available list of coordinate system names and definitions. Few such schemas exist. One suggested schema designation for Heliophysics is provided in the following table. If more community-approved schemas emerge for coordinate systems, then they could be included in the table, and if these schemas become widely adopted, it might make sense in the future to restrict `coordinateSystemSchema` to a set of enumerated values, but for now it is unconstrained.

Schema Name	URL	Description
spase2.4.1	https://spase-group.org/data/schema/spase-2.4.1.xsd	Schema containing the names and definitions of coordinate systems commonly used in Heliophysics. See also an HTML version of relevant section of schema.

This schema captures a Heliophysics-specific list of coordinate systems and is part of the larger SPASE metadata model (also Heliophysics-specific). It serves as an example of how a publically accessible list of coordinate frames can be referenced by the `coordinateSystemSchema` keyword. Different communities can reference their schemas.

Within the `parameter` object is the `coordinateSystemName` keyword, which contains the name of the coordinate system (must be from the schema). There is also the `vectorComponents` keyword to capture the details about which coordinate components are present in the parameter. See the description below (the `vectorComponents` section) for details on how to describe parameters that contain directional (i.e., vector) quantities.

3.6.5 `additionalMetadata` Object

HAPI allows for bulk inclusion of additional metadata that may exist for a dataset. Additional metadata keywords can be inserted by prefixing them with `x_` (which indicates that the element is not part of the HAPI standard), but this means any original metadata would have to modify its keywords.

The `additionalMetadata` object is a list of objects represented by the table below and allows for one or more sets of additional metadata, which can be isolated from each other and HAPI keywords.

```
{
  "additionalMetadata" : [ md1, md2, md3 ]
}
```

There can be one or more metadata objects (md1, md2, md3, etc above) in the list, and the keywords for these objects are as follows:

Keyword	Type	Description
name	string	**Optional** the name of the additional metadata
content	string or JSON object	**Required** (if no `contentURL`) either a string with the metadata content (for XML), or a JSON object representing the object tree for the additional metadata
contentURL	string	**Required** (if no `content`) URL pointing to additional metadata
schemaURL	string	**Optional** points to computer-readable schema for the additional metadata
aboutURL	string	**Optional** points to human-readable explanation for the metadata

The `name` is appropriate if the additional metadata follows a known standard that people know about. One of `content` or `contentURL` must be present. The `content` can be a string version of the actual metadata or a JSON object tree. If there is a schema reference embedded in the metadata (easy to do with XML and JSON), clients can figure that out, but if no internal schema is in the metadata, then the `schemaURL` can point to an external schema. The `aboutURL` is for humans to learn about the given type of additional metadata.

For the `name`, please use these if appropriate CF, FITS, ISTP, SPASE. Other fields beyond Heliophysics will likely have their metadata names, which could be listed here if requested.

Example

```
 {                                                                          1
  "additionalMetadata" : [                                                  2
     { "name": "SPASE",                                                     3
       "contentURL": "https://hpde.io/NASA/DisplayData/ACE/MAG/27-Day.xml", 4
       "aboutURL":  "http://spase-group.org"                               5
     },                                                                     6
     { "name": "CF",                                                        7
       "content":  { "keyword" : "value1", "keyword2": "value2" },          8
       "aboutURL": "https://cfconventions.org/"                             9
     },                                                                    10
     { "name": "ISTP",                                                     11
        "content":  { "json object (not shown) representing the tree of ISTP    12
keyword-value pairs" },
        "aboutURL":  "https://spdf.gsfc.nasa.gov/istp_guide/variables.html"    13
     },                                                                    14
     {  "name": "FITS",                                                    15
        "content":  { "fits_header_keyword1" : "value1", "fits_header_keyword2":   16
"value2" },
        "aboutURL": "http://fits.gsfc.nasa.gov"                            17
     }                                                                     18
   ]                                                                       19
 }                                                                         20
```

Note that no single dataset would likely have this variety of additional metadata – these are provided for illustration.

3.6.6 `stringType` Object

`stringType` is an optional element within each `parameter` object, and it allows servers to indicate that a string parameter has a special interpretation.

Currently, the only special `stringType` allowed is a URI, and it can be used to indicate that a string parameter contains a time series of links to resources (pointed to by the URIs).

The main use of HAPI is serving numeric data, but the ability to also serve URIs that point to data opens up two use cases for HAPI servers.

1. Serving image URIs. In this case, the images should be in a widely recognized format that could be easily interpreted by libraries available to many clients, such as JPG, PNG, etc.

2. Serving data file URIs to provide a list of files used to construct a HAPI numeric data response. For example, if a server has a dataset named `dataset1`, the files used to construct a request for `dataset1` could be provided in a dataset named `dataset1Files`. In this case, a user can request `dataset1` over a time range and determine what files `dataset1` came from using a request for `dataset1Files` over the same time range.

   It is emphasized that a HAPI server that provides only datasets with data file URIs that contain time series data that could be served as HAPI numeric data is not recommended. HAPI clients should only need to read a HAPI stream and not have to read and parse data in arbitrary file formats.

A recommended practice in both cases is also to include columns that provide metadata values.

The `stringType` attribute can either have a simple value that is just the string `uri`, or it can be an object that is a dictionary with `uri` as the key and a value that is another object with three optional elements: `mediaType`, `scheme`, and `base`. Thus a `stringType` will have one of the following forms:

```
"stringType": "uri"
```

or

```
"stringType": {                                                          1
    "uri": {                                                             2
            "mediaType": "image/png",                                    3
            "scheme": "https",                                           4
            "base":                                                      5
"https://cdaweb.gsfc.nasa.gov/pub/pre_generated_plots/de/pwi/"
    }                                                                    6
}                                                                        7
```

The `uri` object attributes are:

stringType Attribute	Type	Description
mediaType	string	**Optional** indicates content type behind the URI (also referred to as MIME type)
scheme	string	**Optional** the access protocol for the URI
base	string	**Optional** allows each URI string value to be relative to a base URI

The media type indicates the type of data each URI points to. HAPI places no constraints on the values for `mediaType`, but servers should only use standard values, such as `image/fits`, `image/png` or `application/x-cdf`. There are standard lists of media types available, and we do not repeat them in the HAPI specification.

The `scheme` describes the access protocol. Again, there are no restrictions, but there is an expectation that it should be a well-known protocol, such as `http` or `https` or `ftp` or `doi` or `s3` (used for Amazon

object stores).

The `base` allows the individual string values for the parameter to be relative to a base URI, typically a web-accessible location ending in a slash.

URIs should follow the syntax outlines in RFC 3986. The basic pattern is:

```
URI = scheme ":" ["//" authority] path ["?" query] ["#" fragment]
```

URI strings should not be encoded because this is what most clients expect, and clients typically do their own encoding of a URI before issuing a request to retrieve the content.

The units for a string parameter that is a URI should be `null`. The units value here should not be used to try and describe the contents behind the URIs. URI content is likely too variable to be uniformly handled by this simple units indicator.

Example:

```
 "parameters":                                                       1
    [ { "name": "Time",                                              2
        "type": "isotime",                                           3
        "units": "UTC",                                              4
        "fill": null,                                                5
        "length": 24                                                 6
      },                                                             7
      { "name": "solar_images",                                      8
        "description": "full-disk images of the Sun by SDO/AIA at wavelength of   9
304 angstroms",
        "type": "string",                                            10
        "length": 64,                                                11
        "stringType": {"uri": {"mediaType": "image/fits", "scheme": "https"}},    12
        "units": null,                                               13
        "fill": null                                                 14
      },                                                             15
      { "name": "cadence",                                           16
        "description": "time between images",                        17
        "type": "double",                                            18
        "units": "sec",                                              19
        "fill": "-999"                                               20
      },                                                             21
      { "name": "wavelength",                                        22
        "description": "which wavelength filter was active for this image",   23
        "type": "double",                                            24
        "units": "angstrom",                                         25
        "fill": "NaN"                                                26
      },                                                             27
      { "name": "contains_active_region",                            28
        "description": "boolean indicator of active region presence: 0=no,   29
1=yes",
        "type": "integer",                                           30
        "units": null,                                               31
        "fill": "NaN"                                                32
      }                                                              33
    ]                                                                34
```

This example shows what the `parameters` portion of a HAPI `info` response would look like for a set of solar images. The parameter name `solar_images` is given a `stringType` of `uri` and has a `mediaType` and `scheme` specified. No `base` is given, so the URIs must be fully qualified. There are also other parameters (`cadence`, `wavelength`, and `contains_active_region`) that could be used on the client side for filtering the images based on the values of those parameters.

The approach shown here emphasizes a useful way for HAPI to provide image lists. HAPI queries can only constrain a set of images by time, but if the response contains metadata values in other columns, clients can restrict the image list further by filtering on values in the metadata columns.

3.6.7 parameter Object

The focus of the header is to list the parameters in a dataset. The first parameter in the list must be a time value. This time column serves as the independent variable for the dataset. The time column parameter may have any name, but its type must be `isotime`, and there must not be any fill values in the data stream for this column. Note that the HAPI specification does not clarify if the time values given are the start, middle, or end of the measurement intervals. There can be other parameters of type `isotime` in the parameter list. The table below describes the `parameter` object attributes.

Parameter Attribute	Type	Description
name	string	**Required** A short name for this parameter. Each parameter must have a unique name. It is recommended that all parameter names start with a letter or underscore, followed by letters, underscores, or numbers. This allows the parameter names to become variable names in computer languages. Parameter names in a dataset must be unique, and names are not allowed to differ only by having a different case. Note that because parameter names can appear in URLs that can serve as permanent links to data, changing them will have negative implications, such as breaking links to data. Therefore, parameter names should be stable over time.
type	string	**Required** One of `string`, `double`, `integer`, or `isotime`. Binary content for `double` is always 8 bytes in IEEE 754 format, `integer` is 4 bytes signed little-endian. There is no default length for `string` or `isotime` types. String parameters may include UTF-8 encoded Unicode characters. Strings may be flagged as representing URIs - see the optional `stringType` attribute in this table.
length	integer	**Required** For type `string` and `isotime`; **not allowed for others**. The maximum number of bytes that the string may contain. If the response format is binary and a string has fewer than this maximum number of bytes, the string must be padded with ASCII null bytes. If the string parameter contains only ASCII characters, `length` means the maximum number of ASCII characters. If a string parameter contains UTF-8 encoded Unicode characters, `length` means the maximum number of bytes required to represent all of the characters. For example, if a string parameter can be `A` or α `length: 2` is required because α in Unicode requires two bytes when encoded as UTF-8. HAPI clients that read CSV output from a HAPI server will generally not need to use the `length` parameter. However, for HAPI binary, the `length` parameter is

Parameter Attribute	Type	Description
		needed for parsing the stream. See also the description of HAPI binary.
size	array of integers	**Required** For array parameters; **not allowed for others**. Must be a 1-D array whose values are the number of array elements in each dimension of this parameter. For example, `"size"=[7]` indicates that the value in each record is a 1-D array of length 7. For the `csv` and `binary` output, there must be 7 columns for this parameter – one column for each array element, effectively unwinding this array. The `json` output for this data parameter must contain an actual JSON array (whose elements would be enclosed by `[ ]`). For arrays 2-D and higher, such as `"size"=[2,3]`, the later indices are the fastest moving, so that the CSV and binary columns for such a 2 by 3 would be `[0,0]`, `[0,1]`, `[0,2]` and then `[1,0]`, `[1,1]`, `[1,2]`. Note that `"size": [1]` is allowed but discouraged because clients may interpret it as either an array of length 1 or as a scalar. Similarly, an array size of 1 in any dimension is discouraged because of ambiguity in how clients would treat this structure. Array sizes of arbitrary dimensionality are allowed, but from a practical view, clients typically support up to 3D or 4D arrays. See the `size` Details section for more about array sizes.
units	null OR string OR array of string	**Required** The units for the data values represented by this parameter. For dimensionless quantities, the value can be the literal string `"dimensionless"` or the special JSON value `null`. Note that an empty string `""` is not allowed. For `isotime` parameters, the units must be `UTC`. If a parameter is a scalar, the units must be a single string. For an array parameter, a `units` value that is a single string means that the same units apply to all elements in the array. If the elements in the array parameter have different units, then `units` can be an array of strings to provide specific units strings for each element in the array. Individual values for elements in the array can also be `"dimensionless"` or `null` (but not an empty string) to indicate no units for that element. The shape of such a `units` array must match the shape given by the `size` of the parameter, and the ordering of multi-dimensional arrays of unit strings is as discussed in the `size` attribute definition above. See the following example responses to an `info` query for examples of a single string and string array units and additional details in `units` and `label` Arrays.
fill	string	**Required** A fill value indicates no valid data. If a parameter has no fill present for any records in the dataset, this can be indicated by using a JSON null for

Parameter Attribute	Type	Description
		this attribute as in `"fill": null` See the `fill` Details section for more about fill values, **including the issues related to specifying numeric fill values as strings**. Since the primary time column cannot have fill values, it must specify `"fill": null` in the header.
`description`	string	**Optional** A brief, one-sentence description of the parameter.
`label`	string OR array of string	**Optional** A word or very short phrase that could serve as a label for this parameter (as on a plot axis or in a selection list of parameters). It is intended to be less cryptic than the parameter name. If the parameter is a scalar, this label must be a single string. If the parameter is an array, a single string label or an array of string labels are allowed. `null` and the empty string `""` are not allowed. A single label string will be applied to all elements in the array, whereas an array of label strings specifies a different label string for each element in the array parameter. The shape of the array of label strings must match the `size` attribute, and the ordering of multi-dimensional arrays of label strings is as discussed in the `size` attribute definition above. See also the following example responses to an `info` query for examples of a single string and string array labels and additional details in Units and Label Arrays.
`stringType`	string or object	**Optional** A string parameter can have a specialized type. Currently, the only supported specialized type is a URI. See The `stringType` Object for more details on syntax and allowed values for `stringType`.
`coordinateSystemName`	string	**Optional** Some parameters contain directional or position information, such as look direction, spacecraft location, or a measured vector quantity. This keyword specifies the name of the coordinate system for these quantities. If `coordinateSystemName` is given, `vectorComponents` should be given (but this is not required). If a `coordinateSystemSchema` was given for this dataset, then the `coordinateSystemName` must come from the schema. See the `vectorComponents` Object for more about coordinate systems.
`vectorComponents`	string or array of strings	**Optional** The name or list of names of the vector-related elements. For a scalar `parameter`, only a single string indicating its component type is allowed. For an array `parameter`, an array of strings corresponding to the component names is expected. If `vectorComponents` is given, `length(size)=1` and `size[0]=length(vectorComponents)` are required. If a

Parameter Attribute	Type	Description
		parameter has a `coordinateSystemName` and three elements, and they represent Cartesian (x,y,z) values, then as a convenience, you may omit the `vectorComponents`. For any other situation (not three elements, not Cartesian, not ordered as x,y,z), then you must specify the `vectorComponents` explicitly. See the `vectorComponents` Object for additional details.
`bins`	array of Bins object	**Optional** For array parameters, each object in the `bins` array corresponds to one of the dimensions of the array and describes values associated with each element in the corresponding dimension of the array. The bins object table below describes all required and optional attributes within each `bins` object. For example, if the parameter represents a 1-D frequency spectrum, the `bins` array will have one object describing the frequency values for each frequency bin; within that object, the `centers` attribute points to an array of values to use for the central frequency of each channel, and the `ranges` attribute specifies a range associated with each channel.

Example

These examples show an `info` response for a hypothetical magnetic field dataset.

```
http://server/hapi/info?dataset=ACE_MAG
```

Example Response:

```json
{                                                                              1
    "HAPI": "3.3",                                                             2
    "status": { "code": 1200, "message": "OK"},                               3
    "startDate": "1998-001Z",                                                  4
    "stopDate" : "2017-100Z",                                                  5
    "coordinateSystemSchema" : { "schemaName": "SPASE", "schemaURI": "TBD"},   6
    "parameters": [                                                            7
        { "name": "Time",                                                      8
          "type": "isotime",                                                   9
          "units": "UTC",                                                      10
          "fill": null,                                                        11
          "length": 24 },                                                      12
        { "name": "radial_position",                                           13
          "type": "double",                                                    14
          "units": "km",                                                       15
          "fill": null,                                                        16
          "description": "radial position of the spacecraft",                 17
          "label": "R Position"},                                             18
        { "name": "quality_flag",                                              19
          "type": "integer",                                                   20
          "units": "none",                                                     21
          "fill": null,                                                        22
          "description ": "0=OK and 1=bad"},                                   23
        { "name": "mag_GSE",                                                   24
          "type": "double",                                                    25
          "units": "nT",                                                       26
          "fill": "-1e31",                                                     27
          "size" : [3],                                                        28
          "coordinateSystemName": "GSE",                                       29
          "description": "hourly average Cartesian magnetic field in nT in GSE",  30
          "label": "B field in GSE"}                                          31
    ]                                                                          32
}                                                                              33
```

3.6.8 size Details

The `size` attribute is required for array parameters and not allowed for others. The length of the `size` array indicates the number of dimensions, and each element in the `size` array indicates the number of elements in that dimension. For example, the size attribute for a 1-D array would be a 1-D JSON array of length one, with the one element in the JSON array indicating the number of elements in the data array. For a spectrum, this number of elements is the number of wavelengths or energies in the spectrum. Thus `"size": [9]` refers to a data parameter that is a 1-D array of length 9, and in the `csv` and `binary` output formats, there will be 9 columns for this data parameter. In the `json` output for this data parameter, each record will contain a JSON array of 9 elements (enclosed in brackets `[ ]`).

For arrays of size 2-D or higher, the column orderings need to be specified for the `csv` and `binary` output formats. For both of these formats, the array needs to be "unrolled" into individual columns. The mapping of 2-D array element to an unrolled column index is done so that the later array elements change the fastest. For example, given a 2-D array of `"size": [2, 5]`, the 5-item index changes the most quickly. Items in each record will be like this: `[0,0] [0,1] [0,2] [0,3] [0,4] [1,0] [1,1] [1,2] [1,3] [1,4]`. The ordering is similar for higher dimensions.

No unrolling is needed for JSON arrays because JSON syntax can represent arrays of any dimension. The following example shows one record of data with a time parameter and a single data parameter `"size":[2,5]` (of type double):

```
["2017-11-13T12:34:56.789Z", [ [0.0, 1.1, 2.2, 3.3, 4.4], [5.0, 6.0, 7.0, 8.0, 9.0] ] ]
```

Time-Varying `size`

If the size of a dimension in a multi-dimensional parameter changes over time, the only way to represent this in HAPI is to define the parameter as having the largest potential `size`, and then use a `fill` value for any data elements which are no longer actually being provided.

If this size-changing parameter has bins, then the number of bins would also presumably change over time. Servers can indicate the absence of one or more bins by using the time-varying bin mechanism described above and then providing all fill values for the `ranges` and `centers` of the records where those bins are absent.

The following example shows a conceptual data block (not in HAPI format) where there is an array parameter whose values are in columns `d0` through `d3`. The corresponding bin centers are in the columns `c0` through `c3`. The data block shows what happens in the data if the size of the parameter changes from 4 to 3 after the third time step. The data values change to fill (`-1.0e31` in this case), and the values for the centers also change to fill to indicate that the corresponding array elements are no longer valid elements in the array.

```
 time                   data0 data1 data2 data3    center0 center1 center1      1
center3
 2019-01-01T14:10:30.5   1.2   3.4   5.4   8.9      10.0    20.0    30.0    40.0   2
 2019-01-01T14:10:31.5   1.1   3.6   5.8   8.4      10.0    20.0    30.0    40.0   3
 2019-01-01T14:10:32.5   1.4   3.8   5.9   8.3      10.0    20.0    30.0    40.0   4
 2019-01-01T14:10:33.5   1.3   3.1   5.3   -1.0e31  15.0    25.0    35.0           5
-1.0e31
 2019-01-01T14:10:34.5   1.2   3.0   5.4   -1.0e31  15.0    25.0    35.0           6
-1.0e31
 2019-01-01T14:10:35.5   1.2   3.0   5.4   -1.0e31  15.0    25.0    35.0           7
-1.0e31
```

Note that the fill value in the bin centers column indicates that this `data3` array element is gone more permanently than just finding a fill value in `data3`. Finding some fill values in an array parameter would not necessarily indicate that the column was permanently gone, while the bin center being fill indicates that the array size has effectively changed. If a bin center is fill, the corresponding data column should also be fill, even though this is duplicate information (since having a fill `center3` in a record already indicates a non-usable `data3` in that record.)

Recall that the static `centers` and `ranges` objects in the JSON `info` header cannot contain null or fill values.

3.6.9 fill Details

Note that fill values for all types must be specified as a string (not just as ASCII within the JSON, but as a literal JSON string inside quotes). For `double` and `integer` types, the string should correspond to a numeric value. In other words, using a string like `invalid_int` would not be allowed for an integer fill value. Care should be taken to ensure that the string value given will have an exact numeric representation and special care should be taken for `double` values, which can suffer from round-off problems. For integers, string fill values must correspond to an integer value small enough to fit into a 4-byte signed integer. For `double` parameters, the fill string must parse to an exact IEEE 754 double representation. One suggestion is to use large negative integers, such as `-1.0E30`. The string `NaN` is allowed, in which the case `csv` output should contain the string `NaN` for fill values. For `binary` data output with double NaN values, the bit pattern for quiet NaN should be used, as opposed to the signaling NaN, which should not be used (see [6]). For `string` and `isotime` parameters, the string `fill` value is used at face value, and it should have a length that fits in the length of the data parameter.

3.6.10 units and label Array

When a scalar `units` value is given for an array parameter, the scalar is assumed to apply to all elements in the array – a kind of broadcast application of the single value to all values in the array. For multi-dimensional arrays, the broadcast applies to all elements in every dimension. A partial broadcast to only one dimension in the array is not allowed. Either a full set of unit strings are given to describe every element in the multi-dimensional array, or a single value is given to apply to all elements. This allows for the handling of special cases while keeping the specification simple. The same broadcast rules govern labels.

The previous example included the optional `label` attribute for some parameters. Using a single string for the `units` and `label` of the array parameter `mag_GSE` indicates that all array elements have the same units and label. The next example shows an `info` response for a magnetic field dataset where the vector components are assigned distinct units and labels.

Example

```
{                                                                          1
   "HAPI": "3.3",                                                          2
   "status": {"code": 1200, "message": "OK"},                             3
   "startDate": "1998-001Z",                                               4
   "stopDate" : "2017-100Z",                                               5
   "parameters": [                                                         6
       { "name": "Time",                                                   7
         "type": "isotime",                                                8
         "units": "UTC",                                                   9
         "fill": null,                                                    10
         "length": 24 },                                                  11
       { "name": "radial_position",                                       12
         "type": "double",                                                13
         "units": "km",                                                   14
         "fill": null,                                                    15
         "description": "radial position of the spacecraft",             16
         "label": "R Position"},                                          17
       { "name": "quality_flag",                                          18
         "type": "integer",                                               19
         "units": "none",                                                 20
         "fill": null,                                                    21
         "description ": "0=OK and 1=bad " },                             22
       { "name": "mag_GSE",                                               23
         "description": "B field as magnitude and two angles theta (colatitude)   24
and phi (longitude)",
         "type": "double",                                                25
         "size" : [3],                                                    26
         "coordinateSystemName": "GSE",                                   27
         "vectorComponents": [ "r", "colatitude", "longitude" ],          28
         "units": ["nT","degrees", "degrees"],                            29
         "label": ["B Magnitude", "theta", "phi"],                        30
         "fill": "-1e31" }                                                31
   ]                                                                      32
}                                                                         33
```

Each element in the string array applies to the corresponding element in the `mag_GSE` data array.

The following are examples for `units` and `label` values for a parameter of `size=[2,3]`. The parameter is vector velocity from two separate instruments. In a JSON response, the parameter at a given time would have the form

```
["2017-11-13T12:34:56.789Z", [ [1, 2, 3] [4, 5, 6] ] ]
```

The `[1,2,3]` are measurements from the first instrument and the `[4, 5, 6]` are measurements from the second instrument.

Allowed (scalar for `units` and for `label` applies to all elements in the array)

```
"size": [2,3]
"units": "m/s",
"label": "velocity"
```

Allowed (scalar units applied to all six elements in the array; unique label for each element)

```
"size": [2,3],
"units": "m/s",
"label": [["V1x","V1y","V1z"],["V2x","V2y","V2z"]]
```

Allowed (array of length 1 is treated like scalar; not preferred but allowed)

```
"size": [2,3]
"units": ["m/s"],
"label": [["V1x","V1y","V1z"],["V2x","V2y","V2z"]]
```

Allowed (all elements are properly given their own `units` string)

```
"size": [2,3],
"units": [["m/s","m/s","km/s"],["m/s","m/s","km/s"]],
"label": [["V1x","V1y","V1z"],["V2x","V2y","V2z"]]
```

Not Allowed (array size does not match parameter size – must specify all `units` elements if not just giving a scalar)

```
"size": [2,3],
"units": ["m/s","m/s","km/s"],
"label": [["V1x","V1y","V1z"],["V2x","V2y","V2z"]]
```

Not Allowed (`units` array size does not match parameter size)

```
"size": [2,3]
"units": ["m/s",["m/s","m/s","km/s"]],
"label": [["V1x","V1y","V1z"],["V2x","V2y","V2z"]]
```

3.6.11 `vectorComponents` Object

For a `parameter` that describes a vector-related quantity (position of spacecraft relative to a body, location of ground station, direction of measured vector quantity, detector look direction), the `vectorComponents` keyword indicates the vector-related components present in the data. For a scalar `parameter` that is associated with a vector component, this keyword contains a single string describing the component. For non-scalar parameters, this keyword contains an array of strings naming the coordinate values present in the parameter.

The names represent the mathematical components typically found in vector or positional quantities.

Possible component names are constrained to be one of the following:

Component Name	Meaning
x	Cartesian x component of vector
y	Cartesian y component of vector
z	Cartesian z component of vector
r	Magnitude of vector
rho	Magnitude of radial component (perpendicular distance from z-axis) in a Cylindrical coordinate representation
latitude	Angle relative to x-y plane from -90° to 90°, or -π to π (90° corresponds to +z axis)
colatitude	Angle relative to +z axis, from 0 to 180°, or 0 to π
longitude	Angle relative to +x axis of a projection of the vector into the x-y plane, from -180° to 180°, or -π to π (90° corresponds to +y axis; this is also known as "East longitude")
longitude0	Angle relative to +x axis of a projection of the vector into the x-y plane, from 0° to 360°, or 0 to 2π (270° corresponds to -y axis; this is also known as "East longitude")
altitude	Altitude above a reference sphere or surface as identified in `coordinateSystemName`. For example, Earth's locations are often described by longitude, latitude, and altitude.
other	Any parameter component that cannot be described by a name in this list

Note that `vectorComponents` should be interpreted as describing vector-related elements. In some instances, the `vectorComponents` will describe a proper vector (e.g., if `vectorComponents = ["x", "y", "z"]`).

Many angular quantities in datasets have different names than the ones here (azimuth instead of longitude, or elevation or inclination instead of latitude), but most quantities directly map to these commonly used component names, which come from spherical or cylindrical coordinate systems. A future version of HAPI may offer a way to represent other angular components.

The following parameter object of an `/info` response (but not a full `/info` response) demonstrates the use of directional quantities.

```json
 "parameters": [
     { "name": "spacecraftLocationXYZ",
       "description": "S/C location in X,Y,Z; vector direction from center of
Earth to spacecraft",
       "size": [3],
       "units": "km",
       "coordinateSystemName": "GSM",
       "vectorComponents": ["x", "y", "z"] },

     { "name": "spacecraftLocationSpherical",
       "description": "position in R, MLT and magnetic latitude",
       "size": [3],
       "units": ["Re", "hours", "degrees" ],
       "coordinateSystemName": "GSM",
       "vectorComponents": ["r", "longitude", "latitude"] },

     { "name": "magnetic_field",
       "description": "mag vector in Cartesian coords; components are x,y,z
which is the assumed default for size [3]",
       "size": [3],
       "units": "nT",
       "coordinateSystemName": "GSM" },

     { "name": "magnetic_field_cylindrical",
       "description": "mag vector in cylindrical coords",
       "size": [3],
       "units": ["nT","degrees", "nT"],
       "coordinateSystemName": "GSM",
       "vectorComponents": ["rho", "longitude", "z"] },

     {  "name": "Bx",
       "description": "mag X component",
       "units": "nT",
       "coordinateSystemName": "GSE",
       "vectorComponents": "x" },

     {  "name": "By",
       "description": "mag Y component",
       "units": "nT",
       "coordinateSystemName": "GSE",
       "vectorComponents": "y" },
```

```
      {  "name": "Bz",                                                          41
         "description": "mag Z component",                                      42
         "units": "nT",                                                         43
         "coordinateSystemName": "GSE",                                         44
         "vectorComponents": "z" },                                            45
                                                                                46
      ]                                                                         47
```

In this example, the spacecraft position is provided in two different ways, Cartesian and spherical. The default value for `vectorComponents` is `["x", "y", "z"]`, and so it may be included if desired, as is the case with the first position parameter, `spacecraftLocationXYZ`, or omitted as is shown for the `magnetic_field` parameter. If the `units` provided is a scalar, it applies to all components of the parameter (this is true regardless of whether the `parameter` is a vector). The parameter `magnetic_field_cylindrical` is indeed in cylindrical coordinates, so it does list specific vector components of `["rho", "longitude", "z"]`, and since the units of each component are not the same, those are listed in an array as `["nT", "degrees", "nT"]`.

Note that the `longitude` units of the `spacecraftLocationSpherical` parameter are `hours`. This must be a floating point value to capture fractions of hours, not a string with hours, minutes, and seconds. So, within the data values for this parameter component, 20.5 would be an interpretable value for 20 hours 30 minutes, but 20:30:00 would not be ok.

The last three parameters in this example (`Bx`, `By`, and `Bz`) illustrate how a vector quantity can be spread across multiple scalar parameters. The `vectorComponents` attribute identifies which component is in the parameter. All relevant components would need to be manually combined to create the vector quantity.

3.6.12 location and geoLocation Details

Some datasets have parameters whose measurements are associated with a fixed location. Other measurements are made from locations that change with time. The `location` attribute can express the location of measurements either as a single, fixed location, or as location values that change with time. An alternative attribute, `geoLocation`, can be used instead as a compact way of describing an Earth-based position for a dataset's measurements. Only `location` or `geoLocation` should be used, not both.

For the case of a fixed location on Earth, the `geoLocation` attribute concisely represents a point location with an array of longitude, latitude, and optionally altitude.

```
"geoLocation": [longitude, latitude]
   -OR-
"geoLocation": [longitude, latitude, altitude]
```

Angles in `geoLocation` must be in `deg` and altitude in `m` and the coordinate system is WGS84 (see Geodetic Coordinate Systems). This makes `geoLocation` consistent with the GeoJSON specification for a point location.

As an example, the location of measurements made by a hypothetical magnetometer at the peak of Sugarloaf Mountain in Maryland, USA, could be represented as:

```
"geoLocation": [-77.395, 39.269, 391.0]
```

For a fixed location expressed in a different coordinate system, or with other vector components for the location, use a `location` object with these four attributes:

location Attribute	Type	Description
`point`	double or double array	**Required** values to specify the location
`vectorComponents`	string or string array	**Required** the kind of `vectorComponents` values in the `point` array. Number of values should match what is in `point`.
`units`	string or string array	**Required** units string or strings for the `vectorComponents` values. To see how `units` are applied to scalars or arrays, see the section about units and arrays
`coordinateSystemName`	string	**Required** the name of the coordinate system for the `vectorComponents` quantities

If a `unitsSchema` has been specified for this dataset, any string given for the `units` must be consistent with that schema. Similarly, if a `coordinateSystemSchema` has been specified for this dataset, any string given for the `coordinateSystemName` must be consistent with that schema.

Examples

A verbose version of `"geoLocation": [-77.395, 39.269, 391.0]`:

```
"location": {                                               1
   "point": [-77.395, 39.269, 391.0],                       2
   "components": ["longitude", "latitude", "altitude"],     3
   "units": ["deg", "deg", "m"],                            4
   "coordinateSystemName": "wgs84"                          5
}                                                           6
```

Location in non-WGS84 coordinate system and with cartesian vector components:

```
"location": {                                               1
   "point": [-4.1452e3, 1.2050e3, 0.10201e3],               2
   "components": ["x", "y", "z"],                           3
   "units": "km",                                           4
   "coordinateSystemName": "GSE"                            5
}                                                           6
```

Note that in the second example, the units value of km applies to all components elements.

Time-Varying Locations

If a dataset has parameters where the measurement location changes over time, the `location` object can indicate the names of other parameters in the dataset that contain the corresponding time-varying locations. If the time-varying position is present in more than one coordinate system, each can be referenced. Therefore, the `location` attribute is an array (outer array) consisting of a list of inner arrays of string parameter names. If a parameter containing location info has all the `vectorComponents` in it to fully represent the location, then the inner array will have just one element: the name of the fully sufficient parameter. If the position info for one coordinate system is spread over multiple parameters, then each parameter name needs to be in the inner array.

```
"location": {
   "parameters": [ ["param_name_for_location_using_coord_sys_A"],
["param_name_for_location_using_coord_sys_B"] ]
      # each parameter here must be a vector and have in its attributes a full set
of `vectorComponents` to describe the vector
 }
```

Examples help illustrate:

```
"location": {
   "parameters": [ ["Location_GEO"], ["Location_GSE"] ]
   }
```

In this first example, two parameters provide position info, each in a different coordinate system. Within the definition of each location parameter, there must be a `vectorComponent` description.

```
location: {                                                               1
   "parameters": [ ["Location_GEO_X", "Location_GEO_Y", "Location_GEO_Z],   2
                   ["Location_J2000_X", "Location_J2000_Y", "Location_J2000_Z"]   3
               ]                                                           4
}                                                                         5
```

In this second example, there are also two coordinate systems, but each one is expressed across three parameters, one each for the x, y, and z values of the position.

3.6.13 bins Object

The bins attribute of a parameter is an array of JSON objects with the following attributes.

Bins Attribute	Type	Description
name	string	**Required** Name for the dimension (e.g. "Frequency").
centers	array of n doubles	**Required if no `ranges`** The centers of each bin range.
ranges	array of n arrays of 2 doubles	**Required if no `centers`** The boundaries for each bin (array of `[min, max]`).
units	string	**Required** The units for the bin ranges and/or center values.

Bins Attribute	Type	Description
label	string	**Optional** A label appropriate for a plot (use if `name` is not appropriate)
description	string	**Optional** Brief comment explaining what the bins represent.

Notes:

- At least one of `ranges` and `centers` must be given.
- Some dimensions of a multi-dimensional parameter may not represent binned data. Each dimension must be described in the `bins` object, but any dimension not representing binned data should indicate this by using `"centers": null` and not including the `'ranges'` attribute.
- Bin centers and/or ranges may depend on time (but the number of centers and/or ranges may not change), see the description of time-varying bins later in this section.
- Bin centers and/or ranges may not depend on non–time dimensions. For example, suppose the parameter dimensions are time, energy and pitch angle. The pitch angle bin centers and/or ranges cannot depend on the energy bins.

Example

```
"parameters":                                                              1
[                                                                          2
    {                                                                      3
        "name": "Time",                                                    4
        "type": "isotime",                                                 5
        "units": "UTC",                                                    6
        "fill": null,                                                      7
        "length": 24                                                       8
    },                                                                     9
    {                                                                      10
        "name": "Protons_10_to_20_keV_pitch_angle_spectrogram",           11
        "type": "double",                                                  12
        "units": "1/(cm^2 s^2 ster keV)",                                 13
        "fill": "-1.0e31",                                                 14
        "size": [6],                                                       15
        "bins": [{                                                         16
            "name": "angle_bins",                                          17
            "ranges": [                                                    18
                        [0.0,    30.0],                                    19
                        [30.0,   60.0],                                    20
                        [60.0,   90.0],                                    21
                        [90.0,   120.0],                                   22
                        [120.0, 150.0],                                    23
                        [150.0, 180.0]                                     24
            ],                                                             25
            "units": "degrees",                                           26
            "label": "Pitch Angle"                                        27
        }]                                                                 28
    }                                                                     29
]                                                                         30
```

The data given for `centers` and `ranges` must not contain any `null` or missing values. The number of valid numbers in the `centers` array and the number of valid min/max pairs in the `ranges` array must match the size of the parameter dimension being described. So this is not allowed:

Invalid

```
centers = [2.0, 3.0, 4.0],
ranges = [[1.0, 3.0], null, [3.0, null]]
```

Invalid

```
centers = [2.0, 3.0, null]
```

Valid

```
centers = [2.0, 3.0, 4.0],
ranges = [[1.0, 3.0], [2.0, 4.0], [3.0, 5.0]]
```

Time-Varying Bins

In some datasets, the bin centers and/or ranges may vary with time. The static values in the `bins` object definition for `ranges` or `centers` are fixed arrays and therefore cannot represent bin boundaries that change over time. As of HAPI 3.0, the `ranges` and `centers` objects can be, instead of a numeric array, a string value that is the name of another parameter in the dataset. This allows the `ranges` and `centers` objects to point to a parameter that is the source of numbers for the bin `centers` or `ranges`. The size of the target parameter must match that of the bins being represented. And, of course, each record of data can contain a different value for the parameter, effectively allowing the bin `ranges` and `centers` to change potentially at every time step.

Due to its complexity, plotting data with time-varying bins may not be supported by all clients. We recommend that if not supported, the client communicates this to the user.

The following example shows a dataset of multi-dimensional values: proton intensities over multiple energies and at multiple pitch angles. The data parameter name is `proton_spectrum`, and it has bins for both an energy dimension (16 different energy bins) and a pitch angle dimension (3 different pitch angle bins). For the bins in both of these dimensions, a parameter name is given instead of numeric values for the bin locations. The parameter `energy_centers` contains an array of 16 values at each time step, and these are to be interpreted as the time-varying centers of the energies. Likewise, there is a `pitch_angle_centers` parameter, which serves as the source of numbers for the centers of the other bin dimension. There are also `ranges` parameters that are two-dimensional elements since each range consists of a high and low value.

Note that the comments embedded in the JSON (with a prefix of `//`) are for human readers only since comments are not supported in JSON.

```
{                                                                      1
    "HAPI": "3.3",                                                     2
    "status": {"code": 1200, "message": "OK"},                         3
    "startDate": "2016-01-01T00:00:00.000Z",                           4
    "stopDate": "2016-01-31T24:00:00.000Z",                            5
    "parameters":                                                      6
        [ { "name": "Time",                                            7
            "type": "isotime",                                         8
            "units": "UTC",                                            9
            "fill": null,                                             10
            "length": 24                                              11
          },                                                          12
          { "name": "proton_spectrum",                                13
            "type": "double",                                         14
            "size": [16,3],                                           15
            "units": "particles/(sec ster cm^2 keV)",                 16
            "fill": "-1e31",                                          17
            "bins":                                                   18
            [                                                         19
                { "name": "energy",                                   20
                  "units": "keV",                                     21
                  "centers": "energy_centers",                        22
                  "ranges":  "energy_ranges"                          23
                },                                                    24
                { "name": "pitch_angle",                              25
                  "units": "degrees",                                 26
                  "centers": "pitch_angle_centers",                   27
                  "ranges":  "pitch_angle_ranges"                     28
                }                                                     29
            ]                                                         30
          },                                                          31
          {                                                           32
            "name": "energy_centers",                                 33
            "type": "double",                                         34
            "size": [16],   // 16 matches size[0] in #/proton_spectrum/size   35
            "units": "keV", // Should match #/proton_spectrum/units    36
            "fill": "-1e31" // Clients should interpret as meaning no  37
measurement made in bin
          },                                                          38
          { "name": "energy_ranges",                                  39
            "type": "double",                                         40
```

```
            "size": [16,2], // 16 matches size[0] in #/proton_spectrum/size;     41
size[1] must be 2
            "units": "keV", // Should match #/proton_spectrum/units              42
            "fill": "-1e31" // Clients should interpret as meaning no             43
measurement made in bin
        },                                                                        44
        { "name": "pitch_angle_centers",                                          45
          "type": "double",                                                       46
          "size": [3],        // 3 matches size[1] in #/proton_spectrum/size      47
          "units": "degrees", // Should match #/proton_spectrum/units             48
          "fill": "-1e31"     // Clients should interpret as meaning no           49
measurement made in bin
        },                                                                        50
        { "name": "pitch_angle_ranges",                                           51
          "type": "double",                                                       52
          "size": [3,2],      // 3 matches size[1] in #/proton_spectrum/size;     53
size[1] must be 2
          "units": "degrees", // Should match #/proton_spectrum/units             54
          "fill": "-1e31"     // Clients should interpret as meaning no           55
measurement made in bin
        }                                                                         56
    ]                                                                             57
}                                                                                 58
```

3.6.14 JSON References

If the same information appears more than once within the info response, it is better to represent this in a structured way rather than to copy and paste duplicate information. Consider a dataset with two parameters – one for the measurement values and one for the uncertainties. If the two parameters both have bins associated with them, the bin definitions would likely be identical. Having each bins entity refer back to a pre-defined, single entity ensures that the bin values are identical, and it also more readily communicates the connection to users, who otherwise would have to do a value-by-value comparison to see if the bin values are the same.

JSON has a built-in mechanism for handling references. HAPI utilizes a subset of these features, focusing on the simple aspects implemented in many existing JSON parsers. Also, using only simple features makes it easier for users to implement custom parsers. Note that familiarity with the full description of JSON references (5), is helpful in understanding the use of references in HAPI described below.

JSON objects placed within a definitions block can be pointed to using the $ref notation. We begin with an example. This definitions block contains this object called angle_bins that can be referred to using the JSON syntax #/definitions/angle_bins

```json
{                                                                        1
  "definitions": {                                                       2
    "pitch_angle_bins": {                                                3
      "name": "angle_bins",                                              4
      "ranges": [ [0.0,    30.0],                                        5
                  [30.0,   60.0],                                        6
                  [60.0,   90.0],                                        7
                  [90.0,   120.0],                                       8
                  [120.0, 150.0],                                        9
                  [150.0, 180.0]                                         10
              ],                                                         11
      "units": "degrees",                                               12
      "label": "Pitch Angle"                                            13
    }                                                                    14
  }                                                                      15
}                                                                        16
```

Here is a parameter fragment showing the reference used in two places:

```
 {                                                                                   1
    "parameters": [                                                                  2
        { "name": "Time",                                                            3
          "type": "isotime",                                                         4
          "units": "UTC",                                                            5
          "fill": null,                                                              6
          "length": 24                                                               7
        },                                                                           8
        { "name": "Protons_10_to_20_keV_pitch_angle_spectrogram",                    9
          "type": "double",                                                          10
          "units": "1/(cm^2 s^2 ster keV)",                                          11
          "fill": "-1.0e38",                                                         12
          "size": [6],                                                               13
          "bins": [                                                                  14
                  {"$ref" : "#/definitions/angle_bins"}                              15
                  ]                                                                  16
        },                                                                           17
        { "name": "Uncert_for_Protons_10_to_20_keV_pitch_angle_spectrogram",         18
          "type": "double",                                                          19
          "units": "1/(cm^2 s^2 ster keV)",                                          20
          "fill": "-1.0e38",                                                         21
          "size": [6],                                                               22
          "bins": [                                                                  23
                  {"$ref" : "#/definitions/angle_bins"}                              24
                  ]                                                                  25
                                                                                     26
        }                                                                            27
    ]                                                                                28
 }                                                                                   29
```

The following rules govern the use of JSON references a HAPI info response.

1. Anything referenced must appear in a top-level node named `definitions` (this is a JSON Schema convention [5] but a HAPI requirement).

2. Objects in the `definitions` node may not contain references (JSON Schema [5] allows this, HAPI does not)

3. Referencing by `id` is not allowed (JSON Schema [5] allows this, HAPI does not)

4. `name` may not be a reference (names must be unique anyway - this would make HAPI `info` potentially very confusing).

By default, a server resolves these references and excludes the definitions node. Stated more directly, a server should not return a `definitions` block unless the request URL includes

```
resolve_references=false
```

in which case the response metadata should contain references to items in the `definitions` node. Note these constraints on what can be in the `definitions`:

1. Any element found in the `definitions` node must be used somewhere in the full set of metadata. Note that this full metadata can be obtained via an `info` request or by a `data` request to which the header is prepended (using `include=header`).

2. If an `info` request or a `data` request with `include=header` is for a *subset* of parameters (e.g., `/hapi/info?id=DATASETID&parameters=p1,p2`), the `definitions` node may contain objects that are not referenced in the metadata for the requested subset of parameters; removal of unused definitions is optional in this case.

Here, then, is a complete example of an info response with references unresolved, showing a `definitions` block and the use of references. The `units` string is commonly used, so it is captured as a reference, as is the full bins definition. Note that this example shows how just a part of the `bins` object could be represented – the `centers` object in this case. The example is valid HAPI content, but normally, using both approaches in a single `info` response would not make sense.

```json
{
    "HAPI": "3.3",
    "status": {
        "code": 1200,
        "message": "OK"
    },
    "startDate": "2016-01-01T00:00:00.000Z",
    "stopDate": "2016-01-31T24:00:00.000Z",
    "definitions": {
        "spectrum_units": "particles/(sec ster cm^2 keV)",
        "spectrum_units_explicit": ["particles/(sec ster cm^2
keV)","particles/(sec ster cm^2 keV)","particles/(sec ster cm^2
keV)","particles/(sec ster cm^2 keV)"],
        "spectrum_bins_centers": [15, 25, 35, 45],
        "spectrum_bins": {
            "name": "energy",
            "units": "keV",
            "centers": [15, 25, 35, 45]
        }
    },
    "parameters": [{
            "name": "Time",
            "type": "isotime",
            "units": "UTC",
            "fill": null,
            "length": 24
        },
        {
            "name": "proton_spectrum",
            "type": "double",
            "size": [4],
            "units": {"$ref": "#/definitions/spectrum_units"},
            "fill": "-1e31",
            "bins": [{
                "name": "energy",
                "units": "keV",
                "centers": {"$ref": "#/definitions/spectrum_bins_centers"}
            }]
        },
        {
            "name": "proton_spectrum2",
```

```
                "type": "double",                                        40
                "size": [4],                                             41
                "units": {"$ref": "#/definitions/spectrum_units"},       42
                "bins": [                                                43
                    {"$ref": "#/definitions/spectrum_bins"}              44
                    ]                                                    45
        },                                                               46
        {                                                                47
                "name": "proton_spectrum3",                              48
                "type": "double",                                        49
                "size": [4],                                             50
                "units": {"$ref": "#/definitions/spectrum_units_explicit"}, 51
                "bins": [                                                52
                    {"$ref": "#/definitions/spectrum_bins"}              53
                    ]                                                    54
        }                                                                55
    ]                                                                    56
}                                                                        57
```

3.7 `data`

This endpoint provides access to a dataset and allows for selecting time ranges and parameters to return. Data is returned as a CSV [2], binary, or JSON stream. The Data Stream Content section describes the stream structure and layout for each format.

The resulting data stream can be considered a stream of records, where each record contains one value for each dataset parameter. Each data record must contain a data value or a fill value (of the same data type) for each parameter.

3.7.1 Request Parameters

Items with a * superscript in the following table have been modified from version 2 to 3; see change notes.

Name	Description
dataset [*]	**Required** The identifier for the dataset (allowed characters).
start [*]	**Required** The inclusive begin time for the data to include in the response.
stop [*]	**Required** The exclusive end time for the data to include in the response.
parameters	**Optional** A comma-separated list of parameters to include in the response (allowed characters). Default is all; `...&parameters=&...` in URL should be interpreted as meaning all parameters.
include	**Optional** Has one possible value of "header" to indicate that the info header should precede the data. The header lines will be prefixed with the "#" character.

Name	Description
`format`	**Optional** The desired format for the data stream. Possible values are "csv", "binary", and "json".

3.7.2 Response

The `data` endpoint response is in one of three formats: CSV format as defined by [2] with a mime type of `text/csv`; binary format where floating points number are in IEEE 754 [4] format and byte order is LSB and a mime type of `application/octet-stream`; JSON format with the structure as described below and a mime type of `application/json`. The default data format is CSV. See the Data Stream Content section for more details.

If the header is requested, then for binary and CSV formats, each line of the header must begin with a hash (#) character. For JSON output, no prefix character should be used because the data object will be another JSON element within the response. Other than the possible prefix character, the contents of the header should be the same as returned from the info endpoint. When a data stream has an attached header, the header must contain an additional "format" attribute to indicate if the content after the header is `csv`, `binary`, or `json`. Note that when a header is included in a CSV response, the data stream is not strictly in CSV format.

The first parameter in the data must be a time column (type of `isotime`) and this must be the independent variable for the dataset. If a subset of parameters is requested, the time column is always provided, even if it is not requested.

Note that the `start` request parameter represents an inclusive lower bound, and the `stop` request parameter is the exclusive upper bound. The server must return data records within these time constraints, i.e., no extra records outside the requested time range. This enables the concatenation of results from adjacent time ranges.

There is a relationship between the `info` endpoint and the `data` endpoint because the header from the `info` endpoint describes the record structure of data emitted by the `data` endpoint. Thus, after a single call to the `info` endpoint, a client could make multiple calls to the `data` endpoint (for multiple time ranges, for example) with the expectation that each data response would contain records described by the single call to the `info` endpoint. The `data` endpoint can optionally prefix the data stream with header information, potentially obviating the need for the `info` endpoint.

Both the `info` and `data` endpoints take an optional request parameter (recall the definition of request parameter in the introduction) called `parameters` that allows users to restrict the dataset parameters listed in the header and data stream, respectively. This enables clients (that already have a list of dataset parameters from a previous info or data request) to request a header for a subset of parameters that will match a data stream for the same subset of parameters. The parameters in the subset request must be ordered according to the original order of the parameters in the metadata, i.e., the subset can contain fewer parameters, but their order must be the same as in the `parameter` array in the `/info` response.

Consider the following dataset header for a fictional dataset with the identifier `MY_MAG_DATA`.

An `info` request for this dataset[*]

```
http://server/hapi/info?dataset=MY_MAG_DATA
```

results in a header listing of all the dataset parameters:

```
{                                                                                    1
   "HAPI": "3.3",                                                                    2
   "status": { "code": 1200, "message": "OK"},                                       3
   "startDate": "2005-01-21T12:05:00.000Z",                                          4
   "stopDate" : "2010-10-18T00:00:00Z",                                              5
   "parameters": [                                                                   6
       { "name": "Time",                                                             7
         "type": "isotime",                                                          8
         "units": "UTC",                                                             9
         "fill": null,                                                              10
         "length": 24 },                                                            11
       { "name": "Bx", "type": "double", "units": "nT", "fill": "-1e31"},            12
       { "name": "By", "type": "double", "units": "nT", "fill": "-1e31"},            13
       { "name": "Bz", "type": "double", "units": "nT", "fill": "-1e31"},            14
   ]                                                                                 15
}                                                                                    16
```

An `info` request for a single parameter has the form

```
http://server/hapi/info?dataset=MY_MAG_DATA&parameters=Bx
```

and would result in the following header:

```
{                                                                                    1
   "HAPI": "3.3",                                                                    2
   "status": { "code": 1200, "message": "OK"},                                       3
   "startDate": "2005-01-21T12:05:00.000Z",                                          4
   "stopDate" : "2010-10-18T00:00:00Z",                                              5
   "parameters": [                                                                   6
       { "name": "Time",                                                             7
         "type": "isotime",                                                          8
         "units": "UTC",                                                             9
         "fill": null,                                                              10
         "length": 24 },                                                            11
       { "name": "Bx", "type": "double", "units": "nT", "fill": "-1e31" },           12
   ]                                                                                 13
}                                                                                    14
```

Note that the time parameter is included even though it was not requested.

In this request[*],

```
http://server/hapi/info?dataset=MY_MAG_DATA&parameters=By,Bx
```

the parameters are out of order. So the server should respond with an error code. See HAPI Status Codes for more about error conditions.

3.7.3 Examples

Two examples of data requests and responses are given – one with the header and one without.

3.7.3.1 Data with Header

Note that in the following request, the header is to be included, so the same header from the `info` endpoint will be prepended to the data but with a '#' character as a prefix for every header line.

```
http://server/hapi/data?dataset=path/to/ACE_MAG&start=2016-01-01Z&stop=2016-02-01Z&include=header
```

Response

```
#{                                                                            1
#  "HAPI": "3.3",                                                             2
#  "status": { "code": 1200, "message": "OK"},                               3
#  "format": "csv",                                                          4
#  "startDate": "1998-001Z",                                                 5
#  "stopDate" : "2017-001Z",                                                 6
#  "parameters": [                                                           7
#       { "name": "Time",                                                    8
#         "type": "isotime",                                                 9
#         "units": "UTC",                                                   10
#         "fill": null,                                                     11
#         "length": 24                                                      12
#       },                                                                  13
#       { "name": "radial_position",                                        14
#         "type": "double",                                                 15
#         "units": "km",                                                    16
#         "fill": null,                                                     17
#         "description": "radial position of the spacecraft"               18
#       },                                                                  19
#       { "name": "quality flag",                                           20
#         "type": "integer",                                                21
#         "units ": null,                                                   22
#         "fill": null,                                                     23
#         "description ": "0=OK and 1=bad "                                 24
#       },                                                                  25
#       { "name": "mag_GSE",                                                 26
#         "type": "double",                                                 27
#         "units": "nT",                                                    28
#         "fill": "-1e31",                                                  29
#         "size" : [3],                                                     30
#         "description": "hourly average Cartesian magnetic field in nT in GSE"   31
#       }                                                                   32
#   ]                                                                       33
#}                                                                          34
2016-01-01T00:00:00.000Z,6.848351,0,0.05,0.08,-50.98                        35
2016-01-01T01:00:00.000Z,6.890149,0,0.04,0.07,-45.26                        36
        ...                                                                 37
        ...                                                                 38
2016-01-01T02:00:00.000Z,8.142253,0,2.74,0.17,-28.62                        39
```

3.7.3.2 Data Only

The following example is the same, except it lacks the request to include the header.

```
http://server/hapi/data?dataset=path/to/ACE_MAG&start=2016-01-01&stop=2016-02-01
```

Response

Consider a dataset that contains a time field, two scalar fields, and one array field of length 3. The response will have the form:

```
2016-01-01T00:00:00.000Z,6.848351,0,0.05,0.08,-50.98                              1
2016-01-01T01:00:00.000Z,6.890149,0,0.04,0.07,-45.26                              2
...                                                                               3
...                                                                               4
2016-01-01T02:00:00.000Z,8.142253,0,2.74,0.17,-28.62                              5
```

Note that there is no leading row with column names. The RFC 4180 CSV standard [2] indicates that such a header row is optional. Leaving out this row avoids the complication of naming individual columns representing array elements within an array parameter. Recall that an array parameter has only a single name. HAPI specifies parameter names via the `info` endpoint, which also provides size details for each parameter (scalar or array, and array size if needed). The size of each parameter must be used to determine how many columns it will use in the CSV data. By not specifying a row of column names, HAPI avoids the need to have a naming convention for columns representing elements within an array parameter.

3.7.4 Response formats

The three possible output formats are `csv`, `binary`, and `json`. A HAPI server must support `csv`, while `binary` and `json` are optional. We emphasize that this is a simple and ephemeral streaming transport format and should not be considered or used in the same way as standard file formats such as FITS, HDF, CDF, and netCDF, which were not designed for streaming.

3.7.4.1 CSV

The format of the CSV stream should follow the guidelines for CSV data as described by RFC 4180 [2]. Each CSV record is one line of text, with commas between the values for each dataset parameter. Any value containing a comma must be surrounded with double quotes, and any double-quote within a value must be escaped by a preceding double quote. An array parameter (i.e., the value of a parameter within one record is an array) will have multiple columns resulting from placing each element in the array into its own column. For 1-D arrays, the ordering of the unwound columns is just the index ordering of the array elements. For 2-D arrays or higher, the right-most array index is the fastest moving index when mapping array elements to columns.

It is up to the server to decide how much precision to include in the ASCII values when generating CSV output.

Clients programs interpreting the HAPI CSV stream are encouraged to use existing CSV parsing libraries to be able to interpret the full range of possible CSV values, including quoted commas and escaped quotes. However, it is expected that a simple CSV parser would probably handle more than 90% of known cases.

3.7.4.2 Binary

The binary data output is best described as a binary translation of the CSV stream, with full numerical precision and no commas or newlines. Recall that the dataset header provides type information for each

dataset parameter, and this definitively indicates the number of bytes and the byte structure of each parameter and, thus, of each binary record in the stream. Array parameters are unwound in the same way for binary as for CSV data as described above. All numeric values are little-endian (LSB), integers are always signed and four-byte and floating-point values are always IEEE 754 double-precision values.

Dataset parameters of type `string` and `isotime` (strings of ISO 8601 dates) have a maximum length specified in the info header. This length indicates how many bytes to read for each string value. If the string content is less than the length, the remaining bytes must be padded with ASCII null bytes. If a string uses all the bytes specified in the `length`, no null terminator or padding is needed.

3.7.4.3 JSON

For the JSON output, an additional `data` element added to the header contains the array of data records. These records are similar to the CSV output, except that strings must be quoted and arrays must be delimited with array brackets in standard JSON fashion. An example helps to illustrate what the JSON format looks like. Consider a dataset with four parameters: time, a scalar value, a 1-D array value with an array length of 3, and a string value. The header with the data object might look like this:

```
{   "HAPI": "3.3",                                                              1
    "status": { "code": 1200, "message": "OK"},                                 2
    "startDate": "2005-01-21T12:05:00.000Z",                                    3
    "stopDate" : "2010-10-18T00:00:00Z",                                        4
    "parameters": [                                                             5
        { "name": "Time", "type": "isotime", "units": "UTC", "fill": null,      6
"length": 24 },
        { "name": "quality_flag", "type": "integer", "description": "0=ok; 1=bad",   7
"fill": null },
        { "name": "mag_GSE", "type": "double", "units": "nT",   "fill": "-1e31",   8
"size" : [3],
            "description": "hourly average Cartesian magnetic field in nT in GSE"   9
},
        { "name": "region", "type": "string", "length": 20, "fill": "???",      10
"units" : null}
    ],                                                                          11
 "format": "json",                                                             12
 "data" : [                                                                    13
 ["2010-001T12:01:00Z",0,[0.44302,0.398,-8.49],"sheath"],                     14
 ["2010-001T12:02:00Z",0,[0.44177,0.393,-9.45],"sheath"],                     15
 ["2010-001T12:03:00Z",0,[0.44003,0.397,-9.38],"sheath"],                     16
 ["2010-001T12:04:00Z",1,[0.43904,0.399,-9.16],"sheath"]                      17
 ]                                                                            18
                                                                              19
 }                                                                            20
```

The data element is a JSON array of records. Each record is itself an array of parameters. The time and string values are in quotes, and any data parameter in the record that is an array must be inside square brackets. This data element appears as the last JSON element in the header.

The record-oriented arrangement of the JSON format is designed to allow a streaming client reader to begin reading (and processing) the JSON data stream before it is complete. Note also that servers can stream the data when records are available. In other words, the JSON format can be read and written without holding all the records in memory. This may require a custom JSON formatter, but this streaming capability is important for large responses.

3.7.5 Errors While Streaming

If the server encounters an error after the data has begun and can no longer continue, it must terminate the stream. As a result, clients must detect an abnormally terminated stream and treat this aborted condition the same as an internal server error. See HAPI Status Codes for more about error conditions.

3.7.6 Representation of Time

Time values are always strings, and the HAPI Time format is a subset of the ISO 8601 date and time format [1].

The restriction on the ISO 8601 standard is that time must be represented as

```
yyyy-mm-ddThh:mm:ss.sssZ
```

or

```
yyyy-dddThh:mm:ss.sssZ
```

and the trailing `z` is required. Strings with less precision are allowed as per ISO 8601, e.g., `1999-01Z` and `1999-001Z`. The HAPI JSON schema lists a series of regular expressions that codifies the intention of the HAPI Time specification. The schema allows leap seconds and hour=24, but it should be expected that not all clients will be able to correctly interpret such time stamps.

The name of the time parameter is not constrained. However, the time column name is strongly recommended to be "Time" or "Epoch" or some easily recognizable label.

Note that the ISO 8601 time format allows arbitrary precision on the time values. HAPI servers and clients should be able to interpret times with at least nanosecond precision.

3.7.6.1 Incoming time values

Servers must require incoming time values from clients (i.e., the `start` and `stop` values on a data request) to be valid ISO 8601 time values. The full ISO 8601 specification allows many esoteric options, but servers must only accept a subset of the full ISO 8601 specification, namely one of either year-month-day (`yyyy-mm-ddThh:mm:ss.sssZ`) or day-of-year (`yyyy-dddThh:mm:ss.sssZ`). Any date or time elements missing from the string are assumed to take on their smallest possible value. For example, the string `2017-01-15T23:00:00.000Z` could be given in truncated form as `2017-01-15T23Z`. Servers should be able to parse and properly interpret these truncated time strings. When clients provide a date at day resolution only, the `T` must not be included, so servers should be able to parse day-level time strings without the `T`, as in `2017-01-15Z`.

Note that in the ISO 8601 specification, a trailing `z` on the time string indicates that no time zone offset should be applied (so the time zone is GMT+0). If a server receives an input value without the trailing `z`, it should still interpret the time zone as GMT+0 rather than a local time zone. This is true for time strings with all fields present and for truncated time strings with some fields missing.

Example time range request	comments
`start=2017-01-15T00:00:00.000Z&stop=2017-01-16T00:00.000Z`	OK - fully specified time value with proper trailing Z
`start=2017-01-15Z&stop=2017-01-16Z`	OK - truncated time value that assumes 00:00.000 for the time
`start=2017-01-15&stop=2017-01-16`	OK - truncated with missing trailing Z, but GMT+0 should be assumed

There is no restriction on the earliest date or latest date a HAPI server can accept, but as a practical limit, clients are likely to be written to handle dates only from 1700 to 2100.

3.7.6.2 Outgoing time values

Time values in the outgoing data stream must be ISO 8601 strings. A server may use one of either the `yyyy-mm-ddThh:mm:ss.sssZ` or the `yyyy-dddThh:mm:ss.sssZ` form, but must use one format and length within any given dataset. The time values must not have any local time zone offset, and they must indicate this by including the trailing `z`. Time or date elements may be omitted from the end to indicate that the missing time or date elements should be given their lowest possible value. For date values at day resolution (i.e., no time values), the `T` must be omitted, but the `z` is still required. Note that this implies that clients must be able to parse potentially truncated ISO strings of both Year-Month-Day and Year-Day-of-year styles.

For `binary` and `csv` data, the length of the time string, truncated or not, is indicated with the `length` attribute for the time parameter, which refers to the number of printable characters in the string. Every time string must have the same length, so padding of time strings is unnecessary.

The data returned from a request should strictly fall within the limits of `start` and `stop`, i.e., servers should not pad the data with extra records outside the requested time range. Furthermore, note that the `start` value is inclusive (data at or beyond this time can be included), while `stop` is exclusive (data at or beyond this time shall not be included in the response).

The primary time column is not allowed to contain any fill values. Each record must be identified with a valid time value. If other columns contain parameters of type `isotime` (i.e., time columns that are not the primary time column), there may be fill values in these columns. Note that the `fill` definition is required for all types, including `isotime` parameters. The fill value for a (non-primary) `isotime` parameter does not have to be a valid time string - it can be any string, but it must be the same length string as the time variable.

HAPI metadata (in the `info` header for a dataset) allows a server to specify where timestamps fall within the measurement window. The `timeStampLocation` attribute for a dataset is an enumeration with possible values of `begin`, `center`, `end`, or `other`. This attribute is optional, but the default value is `center`, which refers to the exact middle of the measurement window. If the location of the timestamp is not known or is more complex than any of the allowed options, the server can report `other` for the `timeStampLocation`. Clients are likely to use `center` for `other`, simply because there is not much else they can do. Note that the optional `cadence` attribute is not meant to be accurate enough to use as a way to compute an alternate time stamp location. In other words, given a `timeStampLocation` of `begin` and a `cadence` of 10 seconds, it may not always work to just add 5 seconds to get to the center of the measurement interval for this dataset. This is because the `cadence` provides a nominal duration, and the actual duration of each measurement may vary significantly throughout the dataset. Some datasets may have specific parameters devoted to accumulation time or other measurement window parameters, but

HAPI metadata does not capture this level of measurement window details. Suggestions on handling the issues discussed in this paragraph are given on the implementation notes page.

3.7.6.3 Time Range With No Data

If a request is made for a time range in which there are no data, the server must respond with an HTTP 200 status code. The HAPI status-code must be either `HAPI 1201` (the explicit no-data code) or `HAPI 1200` (OK). While the more specific `HAPI 1201` code is preferred, servers may have a difficult time recognizing the lack of data before issuing the header, in which case the issuing of `HAPI 1200` and the subsequent absence of any data records communicates to clients that everything worked but no data was present in the given interval. Any response that includes a header (JSON always does, and CSV and binary when requested) must have this same HAPI status in the header. For CSV or binary responses without a header, the message body should be empty to indicate no data records.

This example clarifies the ideal case where a user has not requested the header. If servers have no data, the HTTP header

```
HTTP/1.1 200 OK
```

is acceptable, but a more specific header

```
HTTP/1.1 200 OK; HAPI 1201 OK - no data for time range
```

is preferred if the server can detect that there is no data in time and can modify the HTTP status message. This allows clients to verify that the empty body was intended without making a second request that includes the header containing a HAPI status object.

3.7.6.4 Time Range With All Fill Values

If a request is made with a time range in which the response will contain all fill values, the server must respond with all fill values and not an empty body.

4 Status Codes

There are two ways that HAPI servers must report errors, and these must be consistent. Because every HAPI server response is an HTTP response, an appropriate HTTP status and message must be set for each response. The HTTP integer status codes to use are the standard ones (200 means OK, 404 means not found, etc), and these are listed below.

The text message in the HTTP status should not just be the standard HTTP message but should include a HAPI-specific message, and this text should include the HAPI integer code along with the corresponding HAPI status message for that code. These HAPI codes and messages are also are described below. Note the careful use of "must" and "should" here. The use of the HTTP header message to include HAPI-specific details is optional, but the setting of the HTTP integer code status is required.

As an example, it is recommended that a status message such as

```
HTTP/1.1 404 Not Found
```

is modified to include the HAPI error code and error message (as described below)

```
HTTP/1.1 404 Not Found; HAPI 1402 Bad request - error in start time
```

Although the HTTP header mechanism is robust, it is more difficult for some clients to access – a HAPI client using a high-level URL retrieving mechanism may not have easy access to HTTP header content. Therefore the HAPI response itself must also include a status indicator. This indicator appears as a `status` object in the HAPI header. The two status indicators (HAPI and HTTP) must be consistent, i.e., if one indicates success, so must the other. Note that some HAPI responses do not include a header, and in these cases, the HTTP header is the only place to obtain the status.

4.1 `status` Object

The HAPI `status` object is described as follows:

Name	Type	Description
`code`	integer	Specific value indicating the category of the outcome of the request - see HAPI Status Codes.
`message`	string	Human readable description of the status - must conceptually match the intent of the integer code.

HAPI servers must categorize the response status using at least three status codes: `1200 - OK`, `1400 - Bad Request`, and `1500 - Internal Server Error`. These are intentionally analogous to the similar HTTP codes `200 - OK`, `400 - Bad Request`, and `500 - Internal Server Error`.

HTTP code	HAPI status `code`	HAPI status `message`
200	1200	OK
400	1400	Bad request - user input error
500	1500	Internal server error

The exact wording in the HAPI message does not need to match what is shown here. The conceptual message must be consistent with the status, but the wording can differ (or in another language, for example). If the server also includes the HAPI error message in the HTTP status message (recommended, not required), the HTTP status wording should be as similar as possible to the HAPI message wording.

The `about`, `capabilities`, and `catalog` endpoints need to indicate `1200 - OK` or `1500 - Internal Server Error` since they do not take any request parameters. The `info` and `data` endpoints do take request parameters, so their status response must include `1400 - Bad Request` when appropriate.

A response of `1400 - Bad Request` must also be given when the user requests an endpoint that does not exist.

4.2 `status` Error Codes

Servers may optionally provide the more specific codes in the table below. For convenience, a JSON object with these codes and messages is given in the Appendix. It is recommended but not required that a server implement this more complete set of status responses. Servers may add their own codes, but must use numbers outside the `1200`s, `1400`s, and `1500`s to avoid collisions with possible future HAPI codes.

HTTP code	HAPI status code	HAPI status message
200	1200	OK
200	1201	OK - no data for time range
400	1400	Bad request - user input error
400	1401	Bad request - unknown API parameter name
400	1402	Bad request - syntax error in start time
400	1403	Bad request - syntax error in stop time
400	1404	Bad request - start equal to or after stop
400	1405	Bad request - `start` < `startDate` and/or `stop` > `stopDate`
404	1406	Bad request - unknown dataset id
404	1407	Bad request - unknown dataset parameter
400	1408	Bad request - too much time or data requested
400	1409	Bad request - unsupported output format
400	1410	Bad request - unsupported include value
400	1411	Bad request - out-of-order or duplicate parameters
400	1412	Bad request - unsupported resolve_references value
400	1413	Bad request - unsupported depth value
500	1500	Internal server error
500	1501	Internal server error - upstream request error

Note that there is an OK status to indicate that the request was fulfilled correctly but that no data was found. This can be useful feedback to clients and users, who may otherwise suspect server problems if no data is returned.

Error `1405` implies that a HAPI server should not send data outside of the time range of availability indicated by `startDate` and `stopDate` from an `/info` response. The motivation for this is consistency between metadata and data responses and complexities that could result if servers did not enforce this error condition (see a related issue discussion). For a frequently updated dataset, we recommend that the server set the `stopDate` to a future time if there is a desire not to update the `stopDate` in the `/info` response at the same frequency. We also recommend that the allowed start and stop dates are included in the error message associated with a `1405` error.

Note also the response `1408` indicating that the server will not fulfill the request since it is too large. This gives a HAPI server a way to inform clients about internal limits within the server.

For errors that prevent any HAPI content from being returned (such as a `400 - not found` or `500 - internal server error`), the HAPI server should return a JSON object that is a HAPI header with just the status information. The JSON object should be returned even if the request was for non-JSON data. Returning server-specified content for an error response is also how HTTP servers handle error messages – think about custom HTML content that accompanies the `404 - not found` response when asking a server for a data file that does not exist.

In cases where the server cannot create a full response (such as an `info` request or `data` request for an unknown dataset), the JSON header response must include the HAPI version and a HAPI status object indicating that an error has occurred.

```
{
  "HAPI": "3.3",
  "status": { "code": 1406, "message": "Bad request - unknown dataset id"}
}
```

For the `data` endpoint, clients may request data with no JSON header, and in the case of an error, the HTTP status is the only place a client can determine the response status.

4.3 Client Error Handling

Because web servers are not required to limit HTTP return codes to those in the above table, HAPI clients should be able to handle the full range of HTTP responses. Also, the HAPI server code may not be the only software that interacts with a URL-based request from a HAPI server. There may be a load balancer or upstream request routing or caching mechanism in place. Therefore, it is a good client-side practice to be able to handle any HTTP errors.

Also, recall that in a three-digit HTTP error code, the first digit is the main one client code should examine for determining the response status. Subsequent digits give a finer nuance to the error, but there may be variability between HTTP server implementations for the exact values of the seconds and third digits. HAPI servers can use more specific values for these second and third digits but must keep the first digit consistent with the table above.

Consider the HTTP `204` error code, which represents "No data." A HAPI server can return this code when no data was present over the time range indicated, but (per HTTP rules) it must only do so in cases where the HTTP body contains no data. A HAPI header would count as HTTP data, so the HTTP 204 code can only be sent by a server when the clients request CSV or binary data with no header. Here is a sample HTTP response for this case:

```
HTTP/1.1 204 OK - no data for time range
```

(Note: it may be difficult to change a server's response code message, so this is optional.)

Regardless of whether the server uses a more specific HTTP code, the HAPI code embedded in the HTTP message must properly indicate the HAPI status.

5 Implementation Details

5.1 Cross-Origin Resource Sharing

Because of the increasing importance of JavaScript clients that use AJAX requests, HAPI servers are strongly encouraged to implement Cross-Origin Resource Sharing CORS. This will allow AJAX requests by browser clients from any domain. Enabling CORS is common for servers with only public

data, and not implementing CORS limits the type of clients that can interface with a HAPI server. For testing purposes, the following headers have been sufficient for browser clients to HAPI servers:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET
Access-Control-Allow-Headers: Content-Type
```

5.2 Security Notes

When the server sees a request parameter that it does not recognize, it should throw an error.

So given this query

```
http://server/hapi/data?dataset=DATA&start=T1&stop=T2&fields=mag_GSE&avg=5s
```

the server should throw an error with a status of `1400 - Bad Request` with an HTTP status of `400`. The server could optionally be more specific with `1401 - misspelled or invalid request parameter` with an HTTP code of `404 - Not Found`.

Following general security best practices, HAPI servers should screen incoming request parameter name values. Unknown request parameters and values, including incorrectly formatted time values, should **not** be echoed in the error response.

5.3 HEAD Requests and Efficiency

Although HEAD requests are allowed (and required by the HTTP specification), the HAPI specification does not define any additional or new aspects to the response of a HEAD request. Note that many server frameworks will respond to a HEAD request by making a GET request and then omitting the body in the response since this is a simple way to guarantee that the meta-information in the HEAD request is the same as that in the GET request (as is required by the HTTP specification). As a result, HAPI server developers may want to modify this default behavior to prevent unnecessary processing for HEAD requests.

6 References

1. ISO 8601:2019 Date Time Format Standard, https://www.iso.org/standard/70908.html
2. CSV format, https://tools.ietf.org/html/rfc4180
3. JSON Format, https://tools.ietf.org/html/rfc7159; http://json-schema.org/
4. IEEE Standard for Floating-Point Arithmetic, http://doi.org/10.1109/IEEESTD.2008.4610935
5. Understanding JSON Schema - Structuring a Complex Schema, https://json-schema.org/understanding-json-schema/structuring.html

7 Contact

- Jon Vandegriff (jon.vandegriff@jhuapl.edu)
- Robert Weigel (rweigel@gmu.edu)
- Jeremy Faden (faden@cottagesystems.com)
- Sandy Antunes (sandy.antunes@jhuapl.edu)
- Doug Lindholm (doug.lindholm@lasp.colorado.edu)

- Robert Candey (Robert.M.Candey@nasa.gov)

- Bernard Harris (bernard.t.harris@nasa.gov)

8 Appendix

8.1 Sample Landing Page

See https://github.com/hapi-server/server-ui

8.2 Allowed Characters in `id, dataset, and parameters`

HAPI allows the use of UTF-8 encoded Unicode characters for `id`, `dataset`, and `parameters`. (`id` is used in the `/catalog` request and `dataset` and `parameter` are used in `/info` and `/data` requests.)

Not allowed

- Comma (ASCII decimal code 44)

Recommended

1. strings that match the regular expression `[_a-zA-Z][_a-zA-Z0-9]{0,30}` so that URL encoding is not required and names can be mapped directly to a variable name in most programming languages and a file name on modern operating systems;

2. strings with any of `a-z`, `A-Z`, `-`, `.`, `_`, and `~` so that URL encoding is not required; and

3. short strings - the number of bytes required to write a comma-separated list of all parameters in a dataset and all other parts of a request URL (i.e., `http://.../hapi/data?dataset=...`) should be less than 2048 bytes, which is a limitation on a URL length for most web browsers.

Allowed

- Any non-control Unicode characters (but we recommend against using Unicode characters that look like a comma)

8.3 JSON Object of Status Codes

```
  {                                                                              1
    "1200": {"status":{"code": 1200, "message": "OK"}},                         2
    "1201": {"status":{"code": 1201, "message": "OK - no data for time range"}}, 3
    "1400": {"status":{"code": 1400, "message": "Bad request - user input error"}}, 4
    "1401": {"status":{"code": 1401, "message": "Bad request - unknown API        5
  parameter name"}},
    "1402": {"status":{"code": 1402, "message": "Bad request - syntax error in    6
  start time"}},
    "1403": {"status":{"code": 1403, "message": "Bad request - syntax error in stop 7
  time"}},
    "1404": {"status":{"code": 1404, "message": "Bad request - start equal to or  8
  after stop"}},
    "1405": {"status":{"code": 1405, "message": "Bad request - start < startDate  9
  and/or stop > stopDate"}},
    "1406": {"status":{"code": 1406, "message": "Bad request - unknown dataset    10
  id"}},
    "1407": {"status":{"code": 1407, "message": "Bad request - unknown dataset    11
  parameter"}},
    "1408": {"status":{"code": 1408, "message": "Bad request - too much time or   12
  data requested"}},
    "1409": {"status":{"code": 1409, "message": "Bad request - unsupported output 13
  format"}},
    "1410": {"status":{"code": 1410, "message": "Bad request - unsupported include 14
  value"}},
    "1411": {"status":{"code": 1411, "message": "Bad request - out-of-order or    15
  duplicate parameters"}},
    "1412": {"status":{"code": 1412, "message": "Bad request - unsupported        16
  resolve_references value"}},
    "1413": {"status":{"code": 1413, "message": "Bad request - unsupported depth  17
  value"}},
    "1500": {"status":{"code": 1500, "message": "Internal server error"}},        18
    "1501": {"status":{"code": 1501, "message": "Internal server error - upstream 19
  request error"}}
  }                                                                              20
```

8.4 Examples

The following two examples illustrate two different ways to represent a magnetic field dataset. The first lists a time column and three scalar data columns, Bx, By, and Bz for the Cartesian components.

```
{                                                                          1
   "HAPI": "3.3",                                                          2
   "status": { "code": 1200, "message": "OK"},                            3
   "startDate": "2016-01-01T00:00:00.000Z",                               4
   "stopDate": "2016-01-31T24:00:00.000Z",                                5
   "parameters": [                                                         6
      {"name" : "timestamp", "type": "isotime", "units": "UTC", "fill": null,    7
"length": 24},
      {"name" : "bx", "type": "double", "units": "nT", "fill": "-1e31"},         8
      {"name" : "by", "type": "double", "units": "nT", "fill": "-1e31"},         9
      {"name" : "bz", "type": "double", "units": "nT", "fill": "-1e31"}         10
   ]                                                                       11
}                                                                          12
```

This example shows a header for the same conceptual data (time and three magnetic field components), but with the three components grouped into a one-dimensional array of size 3.

```
{                                                                          1
   "HAPI": "3.3",                                                          2
   "status": { "code": 1200, "message": "OK"},                            3
   "startDate": "2016-01-01T00:00:00.000Z",                               4
   "stopDate": "2016-01-31T24:00:00.000Z",                                5
   "parameters": [                                                         6
      { "name" : "timestamp", "type": "isotime", "units": "UTC", "fill": null,    7
"length": 24 },
      { "name" : "b_field", "type": "double", "units": "nT", "fill": "-1e31",     8
"size": [3] }
   ]                                                                       9
}                                                                          10
```

These two different representations affect how a subset of parameters could be requested from a server. The first example, by listing Bx, By, and Bz as separate parameters, allows clients to request individual components:

```
http://server/hapi/data?dataset=MY_MAG_DATA&start=2001Z&stop=2010Z&parameters=Bx
```

This request would just return a time column (always included as the first column) and a Bx column. But in the second example, the components are all inside a single parameter named `b_field` and so a request for this parameter must always return all the components of the parameter. There is no way to request individual elements of an array parameter.

The following example shows a proton energy spectrum and illustrates the use of the 'bins' element. Note also that the uncertainty of the values associated with the proton spectrum is a separate variable. There is currently no way in the HAPI spec to explicitly link a variable to its uncertainties.

```json
{"HAPI": "3.3",                                                                    1
 "status": { "code": 1200, "message": "OK"},                                       2
 "startDate": "2016-01-01T00:00:00.000Z",                                          3
 "stopDate": "2016-01-31T24:00:00.000Z",                                           4
 "parameters": [                                                                   5
   { "name": "Time",                                                               6
     "type": "isotime",                                                            7
     "units": "UTC",                                                               8
     "fill": null,                                                                 9
     "length": 24                                                                  10
   },                                                                              11
   { "name": "qual_flag",                                                          12
     "type": "int",                                                                13
     "units": null,                                                                14
     "fill": null                                                                  15
   },                                                                              16
   { "name": "maglat",                                                             17
     "type": "double",                                                             18
     "units": "degrees",                                                           19
     "fill": null,                                                                 20
     "description": "magnetic latitude"                                            21
   },                                                                              22
   { "name": "MLT",                                                                23
     "type": "string",                                                             24
     "length": 5,                                                                  25
     "units": "hours:minutes",                                                     26
     "fill": "??:??",                                                              27
     "description": "magnetic local time in HH:MM"                                 28
   },                                                                              29
   { "name": "proton_spectrum",                                                    30
     "type": "double",                                                             31
     "size": [3],                                                                  32
     "units": "particles/(sec ster cm^2 keV)",                                     33
     "fill": "-1e31",                                                              34
     "bins": [ {                                                                   35
         "name": "energy",                                                         36
         "units": "keV",                                                           37
         "centers": [ 15, 25, 35 ],                                               38
          } ],                                                                     39
   { "name": "proton_spectrum_uncerts",                                           40
     "type": "double",                                                             41
```

```
    "size": [3],                                        42
    "units": "particles/(sec ster cm^2 keV)",           43
    "fill": "-1e31",                                    44
    "bins": [ {                                         45
        "name": "energy",                               46
        "units": "keV",                                 47
        "centers": [ 15, 25, 35 ],                      48
         } ]                                            49
  }                                                     50
                                                        51
  ]                                                     52
}                                                       53
```

This shows how "ranges" can specify the bins:

```json
{                                                                          1
    "HAPI": "3.3",                                                         2
    "status": { "code": 1200, "message": "OK"},                           3
    "startDate": "2016-01-01T00:00:00.000Z",                              4
    "stopDate": "2016-01-31T24:00:00.000Z",                               5
    "parameters": [                                                        6
        {                                                                  7
            "length": 24,                                                  8
            "name": "Time",                                                9
            "type": "isotime",                                            10
            "fill": null,                                                 11
            "units": "UTC"                                                12
        },                                                                13
        {                                                                 14
            "bins": [{                                                    15
                "ranges": [                                               16
                    [  0,   30 ],                                         17
                    [  30,   60 ],                                        18
                    [  60,   90 ],                                        19
                    [  90,  120 ],                                        20
                    [  120,   150 ],                                      21
                    [  150,   180 ]                                       22
                ],                                                        23
                "units": "degrees"                                       24
            }],                                                           25
            "fill": "-1e31",                                             26
            "name": "pitchAngleSpectrum",                                27
            "size": [6],                                                 28
            "type": "double",                                            29
        "units": "particles/sec/cm^2/ster/keV"                           30
        }                                                                31
    ]                                                                    32
}                                                                        33
```

8.5 Robot clients

Processes that introduce traffic to servers which does not represent immediate use for science purposes should be identifiable by the servers. For example, an indexing service that queries a server for metadata or a process that verifies that a server is responsive. Robot clients making such requests should set the `User-Agent` property. This property should identify both an `id` and a URL that describes the bot. For example, `hapibot-a` pings all HAPI servers each hour to see each is responsive and sets the `User-Agent` agent to

```
  "hapibot-a/1.0; https://github.com/hapi-server/data-specification/wiki/hapi-
bots.md#hapibot-a".
```

Note that the use of the wiki page to describe bots is encouraged.

8.6 FAIR

For each of the elements of FAIR listed below (copied from https://www.go-fair.org/fair-principles/), we describe their relationship with the HAPI specification.

Some aspects of HAPI, which is an API and metadata standard, directly address FAIR; however, some FAIR principles must be addressed by an external service or the data provider. As such, HAPI supports FAIR principles to the extent that the principles are within its scope.

8.6.1 Findable

The first step in (re)using data is to find them. Metadata and data should be easy to find for both humans and computers. Machine-readable metadata are essential for automatic discovery of datasets and services, so this is an essential component of the FAIRification process.

1. *(Meta)data are assigned a globally unique and persistent identifier*

The `resourceID` attribute can be used for a globally unique and persistent identifier.

Alternatively,

- If each HAPI dataset does not have a globally unique and persistent identifier, but the server does, a data provider can use the `resourceID` in the `/about` response (but data providers are encouraged to have dataset-level `resourceID`s).
- If a dataset is associated with more than one identifier, a data provider can create a dataset of identifiers and serve this dataset as a time series.

2. *Data are described with rich metadata (defined by Reusable, item 1. below)*

Reusable, item 1: *(Meta)data are richly described with a plurality of accurate and relevant attributes*

The HAPI metadata specification has accurate and relevant attributes; the data provider needs to ensure the attribute values accurately describe the data and include information required for interpretation.

3. *Metadata clearly and explicitly include the identifier of the data they describe*

The HAPI metadata specification requires an identifier (`id`) for every dataset. The list of all `id`s is returned in a `catalog/` request. `id` is also used in the `dataset` request parameter in the URL for an `info/` or `data/` request. (The HAPI `/info` response does not contain the `id` because we have generally avoided the duplication of metadata in responses from different endpoints.)

4. *(Meta)data are registered or indexed in a searchable resource*

This is outside the scope of the HAPI specification. However, there is a way to explore all known HAPI servers at https://hapi-server.org/servers/. We also work with other projects that address registration, indexing, and searching.

8.6.2 Accessible

Once the user finds the required data, they need to know how they can be accessed, possibly including authentication and authorization.

1. *(Meta)data are retrievable by their identifier using a standardized communications protocol*

All HAPI endpoints use the HTTP protocol; HAPI metadata is JSON. The `info/` endpoint takes the dataset `id` and returns JSON metadata. The `data/` endpoint also takes the `id` and returns data in CSV, JSON, or binary.

2. *The protocol is open, free, and universally implementable*

HAPI delivers JSON metadata and well-structured data over HTTP(S). The schema for the data and metadata is free, open, and can be implemented in any programming language.

3. *The protocol allows for an authentication and authorization procedure, where necessary*

This is out of scope for HAPI, which was designed to access open data. The HAPI specification explicitly does not include an option for authentication. Access restrictions can still be implemented using authentication mechanisms outside or independent of HAPI.

4. *Metadata are accessible, even when the data are no longer available*

This is outside the scope of the HAPI specification. However, an affiliated HAPI project caches metadata from all known HAPI servers nightly.

8.6.3 Interoperable

The data usually needs to be integrated with other data. In addition, the data needs to interoperate with applications or workflows for analysis, storage, and processing.

1. *(Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.*

HAPI metadata are in JSON, and JSON schemas are available for validation. HAPI data is transmitted as JSON or Comma Separated Values (CSV), both widely used. (HAPI servers may use a simple binary format, which uses IEEE standards for binary numbers, and the layout mimics the CSV output.)

2. *(Meta)data use vocabularies that follow FAIR principles*

HAPI metadata does not use vocabularies directly, but links can be made to external metadata that uses vocabularies (see next item).

3. *(Meta)data include qualified references to other (meta)data*

Other metadata can be referenced using `additionalMetadata`.

8.6.4 Reusable

The ultimate goal of FAIR is to optimize the reuse of data. To achieve this, metadata and data should be well-described so that they can be replicated and/or combined in different settings.

1. *(Meta)data are richly described with a plurality of accurate and relevant attributes*

This is satisfied by the HAPI specification.

2. *(Meta)data are released with a clear and accessible data usage license*

This can be satisfied by using the `license` attribute.

3. *(Meta)data are associated with detailed provenance*

This can be satisfied using the `provenance` attribute.

4. (Meta)data meet domain-relevant community standards

HAPI is built using the widely adopted RESTful approach to web-accessible resources. We use JSON in a way that is common in the community. The time standard is a subset of the ISO8601 standard for time strings. The design of the HAPI API for requesting and receiving data followed from an analysis of the API of many time series data providers, and HAPI is a standard that provides a common set of features.