The Heliophysics Application Programmer's Interface Specification 3.0.0

Robert Weigel

Jon Vandegriff

Jeremy Faden

Aaron Roberts

Todd King

Robert Candey

Bernard Harris

2021-04-01

Table of Contents

1 Significant Changes to Specification 4
1.1 v2 to v3 API Changes 4
1.2 v2 to v3 Schema Changes 4
2 Introduction 4
2.1 Overview 4
2.2 Adoption 5
2.3 Limitations 6
3 Endpoints 6
3.1 Overview 6
3.2 `hapi` 8
3.3 `about` 8
3.4 `capabilities` 9
3.5 `catalog` 11
3.6 `info` 12
3.6.1 Request Parameters 13
3.6.2 Info Response Object 13
3.6.3 `unitsSchema` Details 15
3.6.4 Parameter Object 16
3.6.5 `size` Details 19
3.6.6 `fill` Details 19
3.6.7 Bins Object 20
3.6.8 Subsetting Parameters 23
3.6.9 JSON References 25
3.6.10 Time-Varying Bins 28
3.6.11 Time-Varying `size` 30
3.7 `data` 31
3.7.1 Request Parameters 31
3.7.2 Response 32
3.7.3 Examples 34
3.7.4 Response formats 35
3.7.5 Errors While Streaming 37
3.7.6 Representation of Time 38

4 Status Codes........41
4.1 `status` Object........42
4.2 `status` Error Codes........42
4.3 Client Error Handling........44
5 Cross-Origin Resource Sharing........44
6 Security Notes........45
7 References........45
8 Contact........45
9 Appendix........45
9.1 Sample Landing Page........45
9.2 JSON Object of Status Codes........46
9.3 Examples........46

1 Significant Changes to Specification

Details on the following are documented as issues in the 3.0 Milestone list.

1.1 v2 to v3 API Changes

Non-backward compatible changes 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

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)

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 using 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 (REST) 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-ordered 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 Adoption

In terms of adopting HAPI as a data delivery mechanism, data providers will likely not want to change or remove existing services, so a HAPI-compliant access mechanism could be added alongside existing services. Several demonstration servers were created while the specification was being written to validate decisions being made, but the code for those was incorporated into existing services and therefore is not generic. There are some generic tools to assist providers with adoption. There is a `node.js` HAPI front-end that requires providers to write only a simple data streaming script. There is also a validator mechanism to test a server for HAPI compliance. But there are not yet any other libraries to assist with server generation, although at least two are planned - one in Python and one in Java. The eventual goal is to create a reference implementation as a full-fledged example that providers could adapt. On the client-side, there are also demonstration-level capabilities, the most mature being a Python reader library. Some clients already support reading from HAPI servers (Autoplot and SPEDAS, for example). Simple libraries in several languages (Matlab, IDL) are available to assist in writing clients that extract data from HAPI servers. However, even without a library or example code, the HAPI specification is designed to be simple enough so that even small data providers could add HAPI compliant access to their holdings.

2.3 Limitations

Because HAPI requires a single time column to be the first column, this requires each record (one row of data) to be associated with one time value (the first value in the row). This has implications for serving files with multiple time arrays in them. Supposed 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 own 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 in the future 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.

The five required endpoints are REST-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 contained 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)
http://server/hapi/capabilities
http://server/hapi/about
http://server/hapi/catalog
http://server/hapi/info?dataset=...[&...]
http://server/hapi/data?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 additional 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 given 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
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, UniversityIowa/VanAllen, LASP/TSI, etc.
title	string	**Required** A short human-readable name for the server. The suggested maximum length is 40 characters.
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.
citation	string	**Optional** How to cite data server. An actionable DOI is preferred (e.g., https://doi.org/...). This `citation` differs from the `citation` in an `/info` response. Here the citation is for the entity that maintains the data server.

Example

`http://server/hapi/about`

Example Response:

```
{
  "HAPI": "3.0",
  "status": {"code": 1200, "message": "OK"},
  "id": "TestData3.0",
  "title": "HAPI 3.0 Test Data and Metadata",
  "contact": "examplel@example.org"
}
```

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.
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.

Example

Retrieve a listing of capabilities of this server.

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

Example Response:

```
{
  "HAPI": "2.1",
  "status": {"code": 1200, "message": "OK"},
  "outputFormats": ["csv", "binary", "json"]
}
```

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

None

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 (see below) 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 human-readable name for the dataset. If none is given, it defaults to the id. The suggested maximum length is 40 characters.

Example

Retrieve a listing of datasets shared by this server.

`http://server/hapi/catalog`

Example Response:

```
{
   "HAPI" : "2.1",
   "status": {"code": 1200, "message": "OK"},
   "catalog":
                [
                   {"id": "ACE_MAG", title:"ACE Magnetometer data"},
                   {"id": "data/IBEX/ENA/AVG5MIN"},
                   {"id": "data/CRUISE/PLS"},
                   {"id": "any_identifier_here"}
                ]
}
```

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 that have 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.

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 the minimal amount of metadata that allows for the automated reading by a client of the data content that is streamed via the `data` endpoint. The header must include a list of the parameters in the dataset, as well as 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 make use of 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 available 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 interpreted properly even if additional subset requests are made with no header. (Although a way to write a client as safe as possible would be to always 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

Name	Description
dataset	**Required** The identifier for the dataset (see change notes)
parameters	**Optional** A subset of the parameters to include in the header.

Response

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

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`.
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.
description	string	**Optional** A brief description of the dataset.
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 above for where to find out about each of these conventions. The list of

		allowed unit specifications is expected to grow to include other well-documented unit standards.
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, for example, the SPASE ID.
creationDate	string	**Optional** Restricted ISO 8601 date/time of the dataset creation.
citation	string	**Optional** How to cite the data set. An actionable DOI is preferred (e.g., https://doi.org/...). Note that there is a `citation` in an `/about` response that is focused on the server implementation, but this `citation` is focused on one dataset.
modificationDate	string	**Optional** Restricted ISO 8601 date/time of the modification of the any content in the dataset.
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.

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 that also have software available for parsing and interpreting unit strings. Note that only major version numbers (if available) are indicated in the convention name. It is expected that this list will grow over time as needed. Current locations of the official definitions and software tools for interpreting the various units 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

3.6.4 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 items and their allowed types.

Parameter Attribute	Type	Description
name	string	**Required** A short name for this parameter. 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`, `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` and `isotime` types.
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.
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 the way 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 below for more about array sizes.
units	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 below (the example responses to an `info` query) for examples of a single string and string array units.
fill	string	**Required** A fill value indicates no valid data is present. If a parameter has no fill present for any records in the dataset, this can be indicated by using a JSON null for this attribute as in `"fill": null` See below for more about fill values, **including the issues related to specifying numeric fill values as strings**. Note that 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). 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. 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. No `null` values or the empty string `""` values are allowed in an array of label strings. See below (the example responses to an `info` query) for examples of a single string and string array labels.
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 table below describes all required and optional attributes within each `bins` object. 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 (min to max) associated with each channel. At least one of these must be specified. The bins object has a required `units` keyword (any string value is allowed), and `name` is also required. See examples below for a parameter with bins describing an energy spectrum. Note that for 2-D or higher bins, each bin array is still a 1-D array -- having bins with 2-D

(or higher) dependencies is not currently supported.

3.6.5 `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, because for both of these formats, the array needs to be "unrolled" into individual columns. The mapping of 2-D array element to unrolled column index is done so that the later array elements change the fastest. This is illustrated with the following example. Given a 2-D array of `"size":[2,5]`, the 5 item index changes the most quickly. Items in each record will be ordered like this `[0,0] [0,1], [0,2] [0,3] [0,4] [1,0,] [1,1] [1,2] [1,3] [1,4]` and the ordering is similarly done 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] ] ]
```

3.6.6 `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 that is 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.

3.6.7 Bins Object

The bins attribute of a parameter is an array of JSON objects. These objects have the attributes described below. **NOTE: Even though `ranges` and `centers` are marked as required, only one of the two must be specified.**

Bins Attribute	Type	Description
name	string	**Required** Name for the dimension (e.g. "Frequency").
centers	array of n doubles	**Required** The centers of each bin.
ranges	array of n array of 2 doubles	**Required** The boundaries for each bin.
units	string	**Required** The units for the bin ranges and/or center values.
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.

Note that 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.

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:

```
centers = [2, null, 4],
ranges = [[1,3], null, [3,5]]
```

If the bin centers or ranges change with time, then having static values for the centers or ranges cannot in the `info` response is inadequate. See the section below on time varying bins for how to handle this situation.

Example

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

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

Example Response:

```
{  "HAPI": "3.0",
   "status": { "code": 1200, "message": "OK"},
   "startDate": "1998-001Z",
   "stopDate" : "2017-100Z",
   "parameters": [
       { "name": "Time",
         "type": "isotime",
         "units": "UTC",
         "fill": null,
         "length": 24 },
       { "name": "radial_position",
         "type": "double",
         "units": "km",
         "fill": null,
         "description": "radial position of the spacecraft",
         "label": "R Position"},
       { "name": "quality_flag",
         "type": "integer",
         "units": "none",
         "fill": null,
         "description ": "0=OK and 1=bad"},
       { "name": "mag_GSE",
         "type": "double",
         "units": "nT",
         "fill": "-1e31",
         "size" : [3],
         "description": "hourly average Cartesian magnetic field in nT in
GSE",
         "label": "B field in GSE"}
   ]
}
```

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

```
{  "HAPI": "3.0",
   "status": {"code": 1200, "message": "OK"},
   "startDate": "1998-001Z",
   "stopDate" : "2017-100Z",
   "parameters": [
       { "name": "Time",
         "type": "isotime",
         "units": "UTC",
         "fill": null,
         "length": 24 },
       { "name": "radial_position",
         "type": "double",
         "units": "km",
         "fill": null,
         "description": "radial position of the spacecraft",
         "label": "R Position"},
       { "name": "quality_flag",
         "type": "integer",
         "units": "none",
         "fill": null,
         "description ": "0=OK and 1=bad " },
       { "name": "mag_GSE",
         "type": "double",
         "units": ["nT","degrees", "degrees"],
         "fill": "-1e31",
         "size" : [3],
         "description": "B field as magnitude and two angles theta
(colatitude) and phi (longitude)",
         "label": ["B Magnitude", "theta", "phi"] }
   ]
}
```

This example is nearly the same as the previous `info` header, but the `mag_GSE` parameter is different. It is given as a magnitude and two direction angles, and it also illustrates the use of an array of strings for the `units` and `label`. Each element in the string array applies to the corresponding element in the `mag_GSE` data 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.

Here are some example fragments from a parameter definition showing what is allowed and not allowed for `units` and `label` values.

OK (scalar units applied to all 6 elements in the array; unique label for each element)

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

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

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

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

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

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

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

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

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

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

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

3.6.8 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)

- **request:** including the `parameters` option, but not specifying any parameter names;
  example:
  `http://server/hapi/data?dataset=MY_MAG_DATA&parameters=&start=1999Z&stop=2000Z`
  response: the is an error condition; server should report a user input error

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.

3.6.9 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 bins values are indeed 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 indeed the same.

JSON has a built-in mechanism for handling references. HAPI utilizes a subset of these features, focusing on the simple aspects that are 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`

```
{
  "definitions" : {
     "pitch_angle_bins": {
         "name" : "angle_bins",
         "ranges": [ [  0,   30 ],
                     [  30,  60 ],
                     [  60,  90 ],
                     [  90,  120 ],
                     [  120, 150 ],
                     [  150, 180 ]
                   ],
         "units": "degrees",
         "label": "Pitch Angle"
      }
   }
}
```

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

```
{
   "parameters": [
       { "name": "Time",
         "type": "isotime",
         "units": "UTC",
         "fill": null,
         "length": 24
       },
       { "name": "Protons_10_to_20_keV_pitch_angle_spectrogram",
         "type": "double",
         "units": "1/(cm^2 s^2 ster keV)",
         "fill": "-1.0e38",
         "size": [6],
         "bins": [
                   {"$ref" : "#/definitions/angle_bins"}
                  ]
       },
       { "name": "Uncert_for_Protons_10_to_20_keV_pitch_angle_spectrogram",
         "type": "double",
         "units": "1/(cm^2 s^2 ster keV)",
         "fill": "-1.0e38",
         "size": [6],
         "bins": [
                   {"$ref" : "#/definitions/angle_bins"}
                  ]

       }
   ]
}
```

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, it would not make sense to use both of these approaches in a single `info` response.

```
{
    "HAPI": "3.0",
    "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_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_centers"}
            }]
        },
        {
            "name": "proton_spectrum2",
            "type": "double",
            "size": [4],
            "units": {"$ref": "#/definitions/spectrum_units"},
            "bins": [
                     {"$ref": "#/definitions/spectrum_bins"}
                    ]
        }
    ]
}
```

3.6.10 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 then to be treated as 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.

This kind of complex data structure for binned data will require some corresponding complexity on clients reading the data, but that it outside the scope of this specification.

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.

```
{
    "HAPI": "3.0",
    "status": {"code": 1200, "message": "OK"},
    "startDate": "2016-01-01T00:00:00.000Z",
    "stopDate": "2016-01-31T24:00:00.000Z",
    "parameters":
        [ { "name": "Time",
            "type": "isotime",
            "units": "UTC",
            "fill": null,
            "length": 24
          },
          { "name": "proton_spectrum",
            "type": "double",
            "size": [16,3],
            "units": "particles/(sec ster cm^2 keV)",
            "fill": "-1e31",
            "bins":
            [
                { "name": "energy",
                  "units": "keV",
                  "centers": "energy_centers",
                  "ranges":  "energy_ranges"
                },
                { "name": "pitch_angle",
                  "units": "degrees",
                  "centers": "pitch_angle_centers",
                  "ranges":  "pitch_angle_ranges"
                }
            ]
          },
          {
            "name": "energy_centers",
            "type": "double",
            "size": [16], // Must match product of elements in
#/proton_spectrum/size
            "units": "keV", // Should match #/proton_spectrum/units
            "fill": "-1e31" // Clients should interpret as meaning no
measurement made in bin
          },
          { "name": "energy_ranges",
            "type": "double",
            "size": [16,2],
            "units": "keV", // Should match #/proton_spectrum/units
            "fill": "-1e31" // Clients should interpret as meaning no
measurement made in bin
          },
          { "name": "pitch_angle_centers",
            "type": "double",
            "size": [3], // Must match product of elements in
#/proton_spectrum/size
            "units": "degrees", // Should match #/proton_spectrum/units
            "fill": "-1e31" // Clients should interpret as meaning no
measurement made in bin
          },
          { "name": "pitch_angle_ranges",
            "type": "double",
            "size": [3,2],
            "units": "degrees", // Should match #/proton_spectrum/units
            "fill": "-1e31" // Clients should interpret as meaning no
measurement made in bin
          }
        ]
}
```

3.6.11 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 using 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 center3
2019-01-01T14:10:30.5   1.2   3.4   5.4  8.9      10.0    20.0     30.0     40.0
2019-01-01T14:10:31.5   1.1   3.6   5.8  8.4      10.0    20.0     30.0     40.0
2019-01-01T14:10:32.5   1.4   3.8   5.9  8.3      10.0    20.0     30.0     40.0
2019-01-01T14:10:33.5   1.3   3.1   5.3 -1.0e31  15.0    25.0     35.0     -1.0e31
2019-01-01T14:10:34.5   1.2   3.0   5.4 -1.0e31  15.0    25.0     35.0     -1.0e31
2019-01-01T14:10:35.5   1.2   3.0   5.4 -1.0e31  15.0    25.0     35.0     -1.0e31
```

Note that the fill value in the bin centers column indicates that this `data3` array element is gone in a more permanent sense than just finding a fill value in `data3`. Just 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.

3.7 `data`

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 thought of as a stream of records, where each record contains one value for each of the dataset parameters. 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.
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. Default is 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.
format	**Optional** The desired format for the data stream. Possible values are "csv", "binary", and "json".

3.7.2 Response

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 just 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 `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 an interaction 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. But the `info` endpoint is useful in that it allows clients to learn about a dataset without having to make a data request.

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 must not rearrange the order of any parameters. Duplicates are not allowed.

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:

```
{  "HAPI": "3.0",
   "status": { "code": 1200, "message": "OK"},
   "startDate": "2005-01-21T12:05:00.000Z",
   "stopDate" : "2010-10-18T00:00:00Z",
   "parameters": [
       { "name": "Time",
         "type": "isotime",
         "units": "UTC",
         "fill": null,
         "length": 24 },
       { "name": "Bx", "type": "double", "units": "nT", "fill": "-1e31"},
       { "name": "By", "type": "double", "units": "nT", "fill": "-1e31"},
       { "name": "Bz", "type": "double", "units": "nT", "fill": "-1e31"},
    ]
}
```

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:

```
{  "HAPI": "3.0",
   "status": { "code": 1200, "message": "OK"},
   "startDate": "2005-01-21T12:05:00.000Z",
   "stopDate" : "2010-10-18T00:00:00Z",
   "parameters": [
       { "name": "Time",
         "type": "isotime",
         "units": "UTC",
         "fill": null,
         "length": 24 },
       { "name": "Bx", "type": "double", "units": "nT", "fill": "-1e31" },
    ]
}
```

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

```
#{
#  "HAPI": "3.0",
#  "status": { "code": 1200, "message": "OK"},
#  "format": "csv",
#  "startDate": "1998-001Z",
#  "stopDate" : "2017-001Z",
#  "parameters": [
#       { "name": "Time",
#         "type": "isotime",
#         "units": "UTC",
#         "fill": null,
#         "length": 24
#       },
#       { "name": "radial_position",
#         "type": "double",
#         "units": "km",
#         "fill": null,
#         "description": "radial position of the spacecraft"
#       },
#       { "name": "quality flag",
#         "type": "integer",
#         "units ": null,
#         "fill": null,
#         "description ": "0=OK and 1=bad "
#       },
#       { "name": "mag_GSE",
#         "type": "double",
#         "units": "nT",
#         "fill": "-1e31",
#         "size" : [3],
#         "description":"hourly average Cartesian magnetic field nT in GSE"
#       }
#   ]
#}
2016-01-01T00:00:00.000Z,6.848351,0,0.05,0.08,-50.98
2016-01-01T01:00:00.000Z,6.890149,0,0.04,0.07,-45.26
        ...
        ...
2016-01-01T02:00:00.000Z,8.142253,0,2.74,0.17,-28.62
```

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
2016-01-01T01:00:00.000Z,6.890149,0,0.04,0.07,-45.26
...
...
2016-01-01T02:00:00.000Z,8.142253,0,2.74,0.17,-28.62
```

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 having to name individual columns representing array elements within an array parameter. Recall that an array parameter has only a single name. The place HAPI specifies parameter names is 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.

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` (which are just 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 very 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": "3.0",
   "status": { "code": 1200, "message": "OK"},
   "startDate": "2005-01-21T12:05:00.000Z",
   "stopDate" : "2010-10-18T00:00:00Z",
   "parameters": [
       { "name": "Time", "type": "isotime", "units": "UTC", "fill": null,
"length": 24 },
       { "name": "quality_flag", "type": "integer", "description": "0=ok;
1=bad", "fill": null },
       { "name": "mag_GSE", "type": "double", "units": "nT",  "fill": "-
1e31", "size" : [3],
           "description": "hourly average Cartesian magnetic field in nT in
GSE" },
       { "name": "region", "type": "string", "length": 20, "fill": "???",
"units" : null}
   ],
"format": "json",
"data" : [
["2010-001T12:01:00Z",0,[0.44302,0.398,-8.49],"sheath"],
["2010-001T12:02:00Z",0,[0.44177,0.393,-9.45],"sheath"],
["2010-001T12:03:00Z",0,[0.44003,0.397,-9.38],"sheath"],
["2010-001T12:04:00Z",1,[0.43904,0.399,-9.16],"sheath"]
]

}
```

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 start streaming the data as soon as records are available. In other words, the JSON format can be read and written without first having to hold all the records in memory. This may require some custom elements in the JSON parser, but preserving this streaming capability is important for keeping the HAPI spec scalable. Note that if pulling all the data content into memory is not a problem, then ordinary JSON parsers will also have no trouble with this JSON arrangement.

3.7.5 Errors While Streaming

If the server encounters an error while streaming the data and can no longer continue, it will have to terminate the stream. The `status` code (both HTTP and HAPI) and message will already have been set in the header and is unlikely to represent the error. Clients will have to be able to detect an abnormally terminated stream and should 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 properly interpret such time stamps.

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

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 in the range from years 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 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 and so padding of time strings is not needed.

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.

Note that the ISO 8601 time format allows arbitrary precision on the time values. HAPI servers should therefore also accept time values with high precision. As a practical limit, servers should at least handle time values down to the nanosecond or picosecond level.

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 set 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. If servers have no data, the OK header

```
HTTP/1.1 200 OK
```

is acceptable, but a more specific header

```
HTTP/1.1 200 OK HAPI 1201 - no data in the interval
```

is preferred if the server can detect in time that there is no data. This allows clients to verify that the empty body was intended.

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 status 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 the following 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`. Note that HAPI code numbers are 1000 higher than the HTTP codes to avoid collisions. For these three simple subtracting 1000. The following table summarizes the minimum required status response categories.

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 is allowed to be different (or in another language, for example). If the server is also including 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 just 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 a more specific error code for the following common types of input processing problems. For convenience, a JSON object with these error codes 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 - error in start time
400	1403	Bad request - error in stop time
400	1404	Bad request - start time equal to or after stop time
400	1405	Bad request - time outside valid range
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
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 properly fulfilled, but that no data was found. This can be very useful feedback to clients and users, who may otherwise suspect server problems if no data is returned.

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 let clients know 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 basically 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": "3.0",
  "status": { "code": 1401, "message": "Bad request - unknown request
parameter"}
}
```

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 to interact 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 servers for the exact values of the seconds and third digits. HAPI servers are allowed to 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 is allowed to 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 truly 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 requested CSV or binary data with no header. Here is a sample HTTP response for this case:

```
HTTP/1.1 204 OK - no content; HAPI 1201 OK - no data for the time range
```

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 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. For servers with only public data, enabling CORS is fairly common, and not implementing CORS limits the type of clients that can interface with a HAPI server. Server implementors are strongly encouraged to pursue a deeper understanding before proceeding with CORS. 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
```

6 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`.

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

7 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

8 Contact

- Jon Vandegriff (jon.vandegriff@jhuapl.edu)
- Robert Weigel (rweigel@gmu.edu)
- Aaron Roberts (aaron.roberts@nasa.gov)
- Jeremy Faden (faden@cottagesystems.com)
- Todd King (tking@igpp.ucla.edu)
- Robert Candey (Robert.M.Candey@nasa.gov)
- Bernard Harris (bernard.t.harris@nasa.gov)
- Nand Lal (nand.lal-1@nasa.gov)

9 Appendix

9.1 Sample Landing Page

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

9.2 JSON Object of Status Codes

```
{
"1200": {"status":{"code": 1200, "message": "HAPI 1200: OK"}},
"1201": {"status":{"code": 1201, "message": "HAPI 1201: OK - no data"}},
"1400": {"status":{"code": 1400, "message": "HAPI error 1400: user input error"}},
"1401": {"status":{"code": 1401, "message": "HAPI error 1401: unknown API parameter name"}},
"1402": {"status":{"code": 1402, "message": "HAPI error 1402: error in start"}},
"1403": {"status":{"code": 1403, "message": "HAPI error 1403: error in stop"}},
"1404": {"status":{"code": 1404, "message": "HAPI error 1404: start equal to or after stop"}},
"1405": {"status":{"code": 1405, "message": "HAPI error 1405: time outside valid range"}},
"1406": {"status":{"code": 1406, "message": "HAPI error 1406: unknown dataset id"}},
"1407": {"status":{"code": 1407, "message": "HAPI error 1407: unknown dataset parameter"}},
"1408": {"status":{"code": 1408, "message": "HAPI error 1408: too much time or data
requested"}},
"1409": {"status":{"code": 1409, "message": "HAPI error 1409: unsupported output format"}},
"1410": {"status":{"code": 1410, "message": "HAPI error 1410: unsupported include value"}},
"1411": {"status":{"code": 1411, "message": "HAPI error 1411: out-of-order or duplicate
parameters"}},
"1412": {"status":{"code": 1412, "message": "HAPI error 1412: unsupported resolve_references
value"}},
"1500": {"status":{"code": 1500, "message": "HAPI error 1500: internal server error"}},
"1501": {"status":{"code": 1501, "message": "HAPI error 1501: upstream request error"}}
}
```

9.3 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.

```
{
   "HAPI": "3.0",
   "status": { "code": 1200, "message": "OK"},
   "startDate": "2016-01-01T00:00:00.000Z",
   "stopDate": "2016-01-31T24:00:00.000Z",
   "parameters": [
      {"name" : "timestamp", "type": "isotime", "units": "UTC", "fill": null,
                 "length": 24},
      {"name" : "bx", "type": "double", "units": "nT", "fill": "-1e31"},
      {"name" : "by", "type": "double", "units": "nT", "fill": "-1e31"},
      {"name" : "bz", "type": "double", "units": "nT", "fill": "-1e31"}
   ]
}
```

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.

```
{
   "HAPI": "3.0",
   "status": { "code": 1200, "message": "OK"},
   "startDate": "2016-01-01T00:00:00.000Z",
   "stopDate": "2016-01-31T24:00:00.000Z",
   "parameters": [
      { "name" : "timestamp", "type": "isotime", "units": "UTC", , "fill":
null, "length": 24 },
      { "name" : "b_field", "type": "double", "units": "nT",, "fill": "-
1e31", "size": [3] }
   ]
}
```

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.

```
{"HAPI": "3.0",
 "status": { "code": 1200, "message": "OK"},
 "startDate": "2016-01-01T00:00:00.000Z",
 "stopDate": "2016-01-31T24:00:00.000Z",
 "parameters": [
   { "name": "Time",
     "type": "isotime",
     "units": "UTC",
     "fill": null,
     "length": 24
   },
   { "name": "qual_flag",
     "type": "int",
     "units": null,
     "fill": null
   },
   { "name": "maglat",
     "type": "double",
     "units": "degrees",
     "fill": null,
     "description": "magnetic latitude"
   },
   { "name": "MLT",
     "type": "string",
     "length": 5,
     "units": "hours:minutes",
     "fill": "??:??",
     "description": "magnetic local time in HH:MM"
   },
   { "name": "proton_spectrum",
     "type": "double",
     "size": [3],
     "units": "particles/(sec ster cm^2 keV)",
     "fill": "-1e31",
     "bins": [ {
         "name": "energy",
         "units": "keV",
         "centers": [ 15, 25, 35 ],
          } ],
   { "name": "proton_spectrum_uncerts",
     "type": "double",
     "size": [3],
     "units": "particles/(sec ster cm^2 keV)",
     "fill": "-1e31",
     "bins": [ {
         "name": "energy",
         "units": "keV",
         "centers": [ 15, 25, 35 ],
          } ]
   }

  ]
}
```

This shows how "ranges" can specify the bins:

```
{
    "HAPI": "3.0",
    "status": { "code": 1200, "message": "OK"},
    "startDate": "2016-01-01T00:00:00.000Z",
    "stopDate": "2016-01-31T24:00:00.000Z",
    "parameters": [
        {
            "length": 24,
            "name": "Time",
            "type": "isotime",
            "fill": null,
            "units": "UTC"
        },
        {
            "bins": [{
                "ranges": [
                    [  0,  30 ],
                    [  30,  60 ],
                    [  60,  90 ],
                    [  90,  120 ],
                    [  120,  150 ],
                    [  150,  180 ]
                ],
                "units": "degrees"
            }],
            "fill": "-1e31",
            "name": "pitchAngleSpectrum",
            "size": [6],
            "type": "double",
        "units": "particles/sec/cm^2/ster/keV"
        }
    ]
}
```