HAPI Data Access Specification

Version 1.1.0 | Heliophysics Data and Model Consortium (HDMC) | May 16, 2017

Table of Contents

Introduction

This document describes the Heliophysics Application Programmer's Interface (HAPI) specification, which is an API and streaming format specification for delivering digital time series data. The intent of HAPI is to enhance interoperability among time series data providers. The HAPI specification describes a lowest common denominator of services that any provider of time series data could implement. In fact, many providers already offer access to their data holdings through some kind of API. The hope is that this specification captures what many providers are already doing, but just codifies the specific details so that providers could use the same exact API. This would make it possible to obtain time series science data content seamlessly from many sources.

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 this focus on access to data content, HAPI is very light on metadata and data discovery. Within the metadata offered by HAPI are optional ways to indicate where further descriptive details for any dataset could be found.

The API itself is built using REST principles that emphasize URLs as 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.

The following definitions are provided first to ensure clarity in ensuing descriptions.

parameter – a measured science quantity or a related ancillary quantity at one instant in time; may be scalar as a function of time, or an array at each time step; must have units; also must have a fill value that represents no measurement or absent information.

dataset – a collection with a conceptually uniform of set of parameters; one instance of all the parameters together with associated with a time value constitutes a data record. A HAPI service presents a dataset as a seamless collection of time ordered records, offering a way to retrieve the parameters while hiding actual storage details.

request parameter – keywords that appear after the '?' in a URL with a GET request.

Consider this example GET request:

The two request parameters are `id` and `time.min`. They are shown in bold and have values of `alpha` and `2016-07-13` respectively. This document will always use the full phrase "request parameter" to refer to these URL elements to draw a clear distinction from a parameter in a dataset.

Line Endings

HAPI servers must use a newline (ASCII code 10 in decimal, 0x0A in hexidecimal) for line endings.

Endpoints

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

1. describe the capabilities of the server; lists the output formats the server can emit (`csv`, `binary`, or `json`, described below)

2. list the catalog of datasets that are available; each dataset is associated with a unique id and may optionally have a title

3. show information about a dataset with a given id; a primary component of the description is the list of parameters in the dataset

4. stream data content for a dataset of a given id; the streaming request must have time bounds (specified by request parameters `time.min` and `time.max`) and may indicate a subset of parameters (default is all parameters)

There is also an optional landing page endpoint for the HAPI service 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 four required endpoints are REST-style services, in that the resulting HTTP response is the complete response for each endpoint. In particular, the `data` endpoint does not just give URLs or links to the data, but rather streams the data content in the HTTP response. The full specification for each endpoint is discussed below.

All endpoints must be directly below a `hapi` path element in the URL:

```
http://hapi-server.org/hapi/capabilities
http://hapi-server.org/hapi/catalog
http://hapi-server.org/hapi/info
http://hapi-server.org/hapi/data
```

All requests to a HAPI server are for retrieving resources and must not change the server state. Therefore, all HAPI endpoints must respond only to HTTP GET requests. POST requests should result in an error. This represents a RESTful approach in which GET requests are restricted to be read-only operations from the server. The HAPI specification does not allow any input to the server (which for RESTful services are often implemented using POST requests).

The input specification for each endpoint (the request parameters and their allowed values) must be strictly enforced by the server. HAPI servers are not allowed to 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 below for more details on how a HAPI server returns status information to clients.) The principle being followed here 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. For example, 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 parametrs are not allowed, and 2. the server will not be doing any averaging.

The outputs from a HAPI server to the `catalog`, `capabilities`, and `info` endpoints are JSON structures, 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, but may optionally deliver data content in binary format or JSON format. The structure of the response stream formats are described below.

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

hapi

This root endpoint is optional and serves as 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 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 available here: https://github.com/hapi-server/data-specification/blob/master/example_hapi_landing_page.html

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://hapi-server.org/hapi
```

Request Parameters

None

Response

The response is in HTML format with a mime type of `text/html`. The content for the landing page is not strictly defined, but should look something like the example below.

Example

Retrieve landing page for this server.

```
http://hapi-server.org/hapi
```

Example Response:

```
<html>
<head> </head>
<body>
<h2> HAPI Server</h2>
<p> This server supports the HAPI 1.0 specification for delivery of time series
   data. The server consists of the following 4 REST-like endpoints that will
   respond to HTTP GET requests.
</p>
<ol>
<li> <a href="capabilities">capabilities</a> describe the capabilities of the server; this lists the output formats the server can emit (CSV and binary)</li>
<li><a href="catalog">catalog</a> list the datasets that are available; each dataset is associated with a unique id</li>
<li><a href="info">info</a> obtain a description for dataset of a given id; the description defines the parameters in every dataset record</li>
<li><a href="data">data</a> stream data content for a dataset of a given id; the streaming request must have time bounds (specified by request parameters time.min and time.max) and may indicate a subset of parameters (default is all parameters)</li>
</ol>
<p> For more information, see <a href="http://spase-group.org/hapi">this HAPI description</a> at the SPASE web site.  </p>
</body>
<html>
```

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 `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 collisions with possible future additions to the specificaiton.

Sample Invocation

```
http://hapi-server.org/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 serve 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://hapi-server.org/hapi/capabilities
```

Example Response:

```
{
  "HAPI": "1.1",
  "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. There is a specific HAPI error code for this, namely 1409 "Bad request - unsupported output format" with a corresponding HTTP response code of 400. See below for more about error responses.

catalog

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

Sample Invocation

```
http://hapi-server.org/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 through the server providing the catalog. 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 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://hapi-server.org/hapi/catalog
```

Example Response:


```
{
  "HAPI" : "1.1",
  "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.

Also, note that the identifiers can have slashes in them.

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 focus of the header is to provide enough metadata to allow automated reading 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 about ten optional metadata elements for capturing other high level information such as a brief description of the dataset, the typical cadence of the data, and ways to learn more about a dataset. A 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, but any non-standard keywords must begin with the prefix `x_`.

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, all the parameters available in the dataset are listed in the header. 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.) This reduced header is potentially 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. The server must ignore duplicates in the subset list, and the server's response must order the subset of parameters according to the ordering in the original, full list of parameters. This ensures that a data request for a subset of parameters can be interpreted properly even if no header is requested. (Although a way to write a client as safe as possible would be to always request the header, and rely on the parameter ordering in the header to guide interpretation of the data column ordering.)

Note that the `data` endpoint may optionally prepend the info header to the data stream. 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 when it is in front of a binary data response. One good way to detect the end of the header is calculate the number of open braces minus the number of closed braces. The last character in the header is the newline following the closing brace that makes open braces minus closed braces equal to zero. For `json` output, the header and data are all withing a single JSON entity, and so newlines are not necessary.

Sample Invocation

```
http://hapi-server.org/hapi/info?id=ACE_MAG
```

Request Parameters

Name	Description
id	**Required** The identifier for the dataset.
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.

Info 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 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** ISO 8601 date of first record of data in the entire dataset.
stopDate	string	**Required** ISO 8601 date for 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.
sampleStartDate	string	**Optional** ISO 8601 date giving 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.
sampleStopDate	string	**Optional** ISO 8601 date giving 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.
description	string	**Optional** A brief description of the dataset.
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** ISO 8601 date and time of the dataset creation.
modificationDate	string	**Optional** Last modification time of the data content in the dataset as an ISO 8601 date.
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.
contact	string	**Optional** Relevant contact person and possibly contact information.
contactID	string	**Optional** The identifier in the discovery system for information about the contact. For example, the SPASE ID of the person.

Parameter

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 measurment 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. Parameter names in a dataset must be unique, and names are not allowed to differ only by having different case.
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 little-endian. There is no default length for string and isotime types. See below for more information on data types.
length	integer	**Required** for type string and isotime; **not allowed for others** The number of bytes or characters that contain the value, including the required null terminator character. Relevant only when data is streamed in binary format.
units	string	**Required** The units for the data values represented by this parameter. For dimensionless quantities, the value can be 'dimensionless' or null. For isotime parameters, the type must be UTC.
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]. See below for more about array sizes. **NOTE: array sizes of 2-D or higher are experimental at this point, and future versions of this specification may update the way 2-D or higher data is described.**
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 description of the parameter.
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. A 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 an optional units keyword (any string value is allowed), and name is required. See below for an example showing a parameter that holds a proton energy spectrum. The use of bins to describe values associated with 2-D or higher arrays is currently supported but should be considered experimental.

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	**Optional** the units for the bins
description	string	**Optional** brief comment explaining what the bins represent

Example

```
http://hapi-server.org/hapi/info?id=ACE_MAG
```

Example Response:


```
{  "HAPI": "1.1",
   "status": { "code": 1200, "message": "OK"},
   "startDate": "1998-001",
   "stopDate" : "2017-100",
   "parameters": [
      { "name": "Time",
        "type": "isotime",
        "units": "UTC",
        "length": 24 },
      { "name": "radial_position",
        "type": "double",
        "units": "km",
        "description": "radial position of the spacecraft" },
      { "name": "quality flag",
        "type": "integer",
        "units ": "none ",
        "description ": "0=OK and 1=bad " },
      { "name": "mag_GSE",
        "type": "double",
        "units": "nT",
        "size" : [3],
        "description": "hourly average Cartesian magnetic field in nT in GSE" }
   ]
}
```


Subsetting the Parameters

Clients may request a response that includes only a subset of the parameters in a dataset. When creating a header for a subset of parameters (via the info endpoint), or a data stream for a subset of parameters (via the data endpoint, described next), the logic on the server is the same 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'); response: all columns

- request: ask for just the primary time parameter; response: just the primary time column
- request: ask for a single parameter other than the primary time column (like 'parameters=Bx'); response: primary time column and the one requested data column
- request: ask for two or more parameters other than the primary time column; response: primary time column followed by the requested parameters in the order they occurred in the original, non-subsetted dataset header (not in the order of the subset request)

data

Provides access to a dataset and allows for selecting time ranges and parameters to return. Data is returned as a stream in CSV[2], binary, or JSON format. 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.

Request Parameters

Name	Description
id	**Required** The identifier for the dataset
time.min	**Required** The inclusive begin time for the data to include in the response
time.max	**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".

Response

Response is in one of three formats: CSV format as defined by RFC-4180 with a mime type of "text/csv"; binary format where floating points number are in IEEE 754[5] 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 section on Data Stream Content 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 `time.min` request parameter represents an inclusive lower bound and `time.max` 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 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. Consider the following dataset header for a fictional dataset with the identifier MY_MAG_DATA.

An `info` request like this:

```
http://hapi-server.org/hapi/info?id=MY_MAG_DATA
```

would result in a header listing of all the dataset parameters:


```
{  "HAPI": "1.1",
   "status": { "code": 1200, "message": "OK"},
   "startDate": "2005-01-21T12:05:00.000",
   "stopDate" : "2010-10-18T00:00:00",
   "parameters": [
      { "name": "Time",
        "type": "isotime",
        "units": "UTC",
        "length": 24 },
      { "name": "Bx", "type": "double", "units": "nT" },
      { "name": "By", "type": "double", "units": "nT" },
      { "name": "Bz", "type": "double", "units": "nT" },
```

```
   ]
}
```

An info request like this:

```
http://hapi-server.org/hapi/info?id=MY_MAG_DATA&parameters=Bx
```

would result in a header listing only the one dataset parameter:

```
{  "HAPI": "1.1",
   "status": { "code": 1200, "message": "OK"},
   "startDate": "2005-01-21T12:05:00.000",
   "stopDate" : "2010-10-18T00:00:00",
   "parameters": [
      { "name": "Time",
        "type": "isotime",
        "units": "UTC",
        "length": 24 },
      { "name": "Bx", "type": "double", "units": "nT" },
   ]
}
```

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

Data Stream Content

The three possible output formats are csv, binary, and json. A HAPI server must support csv, while binary and json are optional.

In the CSV stream, each record is one line of text, with commas between the values for each dataset parameter. 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.

The binary data output is best described as a binary translation of the CSV stream, with full numerical precision and no commas. 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 in the previous paragraph). All numeric values are little endian (LSB), integers are always 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) must have in their header a length element. All strings in the binary stream must be null terminated, and so the length element in the header must include the null terminator as part of the length for that string parameter.

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 illustrate what the JSON format looks like. Consider a dataset with four parameters: time, a scalar value, an 1-D array value with array length of 3, and a string value. The header with the data object might look like this:

```
{  "HAPI": "1.1",
   "status": { "code": 1200, "message": "OK"},
   "startDate": "2005-01-21T12:05:00.000",
   "stopDate" : "2010-10-18T00:00:00",
   "parameters": [
      { "name": "Time", "type": "isotime", "units": "UTC", "length": 24 },
      { "name": "quality_flag", "type": "integer", "description": "0=ok; 1=bad" },
      { "name": "mag_GSE", "type": "double", "units": "nT", "size" : [3],
         "description": "hourly average Cartesian magnetic field in nT in GSE" },
      { "name": "region", "type": "string", "length": 20, "units" : null}
   ],
"format": "json",
"data" : [
["2010-001T12:01:00",0,[0.44302,0.398,-8.49],"sheath"],
["2010-001T12:02:00",0,[0.44177,0.393,-9.45],"sheath"],
["2010-001T12:03:00",0,[0.44003,0.397,-9.38],"sheath"],
["2010-001T12:04:00",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 avaialble. In other words, the JSON format can be read and written without first having to hold all the records in memory. This may rquire some custom elements in the JSON parser, but preserving this streaming capabliity 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.

Errors While Streaming Data

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.

Examples

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

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://hapi-server.org/hapi/data?id=path/to/ACE_MAG&time.min=2016-01-01&time.max=2016-02-01&include=header
```

Example Response: Data with Header


```
#{
#  "HAPI": "1.1",
#  "status": { "code": 1200, "message": "OK"},
#  "format": "csv",
#  "startDate": "1998-001",
#  "stopDate" : "2017-001",
#  "parameters": [
#      { "name": "Time",
#        "type": "isotime",
#        "units": "UTC",
#        "length": 24
#      },
#      { "name": "radial_position",
#        "type": "double",
#        "units": "km",
#        "description": "radial position of the spacecraft"
#      },
#      { "name": "quality flag",
#        "type": "integer",
#        "units ": null,
#        "description ": "0=OK and 1=bad "
#      },
#      { "name": "mag_GSE",
#        "type": "double",
#        "units": "nT",
```

```
#       "size" : [3],
#       "description": "hourly average Cartesian magnetic field in nT in GSE"
#      }
#   ]
#}
2016-01-01T00:00:00.000,6.848351,0,0.05,0.08,-50.98
2016-01-01T01:00:00.000,6.890149,0,0.04,0.07,-45.26
      ...
      ...
2016-01-01T02:00:00.000,8.142253,0,2.74,0.17,-28.62
```

Data Only

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

```
http://hapi-server.org/hapi/data?id=path/to/ACE_MAG&time.min=2016-01-01&time.max=2016-02-01
```

Example Response: Data Only

Consider a dataset that contains a time field, two scalar fields and one array field of length 3. The response will look something like:

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

Note that there is no leading row with column names. The 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.

Implications of the HAPI data model

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.

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) https://www.w3.org/TR/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 deeper understanding before proceeding with CORS. For testing purposes, the following headers have been sufficent for browser clients to HAPI servers:

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

HAPI Status Codes

There are two levels of error reporting a HAPI server must perform. Because every HAPI server response is an HTTP response, an appropriate HTTP status must be set for each response. Although the HTTP codes are robust, they are more difficult for clients to extract -- a HAPI client using a high-level URL retrieving mechanism may not have easy access to HTTP header content. Therefore, every HAPI response with a header must also include a status object indicating if the request succeeded or not. The two status indicators (HAPI and HTTP) must be consistent, i.e., if one indicates success, so must the other.

The status information returned from an endpoint is 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 intentional analgous 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 status categorizations, the HTTP code can be derived from the HAPI code by just 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 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).

The 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.

Servers may optionally provide a more specific error code for the following common types of input processing problems. 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 1200s, 1400s, and 1500s 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 request parameter
400	1402	Bad request - error in start time
400	1403	Bad request - error in stop time
400	1404	Bad request - start time 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
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.

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": "1.1",
  "status": { "code": 1401, "message": "Bad request - unknown request parameter"}
}
```

If no JSON header was requested, then the HTTP error will be the only indicator of a problem. Similarly, for the `data` endpoint, clients may request data with no JSON header, and in this case, the HTTP status is the only place a client can determine the response status.

HAPI 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 good client-side practice to be able to handle any HTTP errors.

Representation of Time

The HAPI specification is focused on access to time series data, so understanding how the server parses and emits time values is important.

When making a request to the server, the time range (`time.min` and `time.max`) values must each be valid time strings according to the ISO 8601 standard. Only two flavors of ISO 8601 time strings are allowed, namely those formatted at year-month-day (yyyy-mm-ddThh:mm:ss.sss) or day-of-year (yyyy-dddThh:mm:ss.sss). Servers should be able to handle either of these time string formats, but do not need to handle some of the more esoteric ISO 8601 formats, such as year + week-of-year. Any date or time elements missing from the string are assumed to take on their smallest possible value. For example, the string `2017-01-10T12` is the same as `2017-01-10T12:00:00.000`. Servers should be able to parse and properly interpret these types of truncated time strings.

Time values in the outgoing data stream must be ISO 8601 strings. A server may use either the yyyy-mm-ddThh:mm:ss or the yyyy-dddThh:mm:ss form, but should use just one format within any given dataset. Emitting truncated time strings is allowed, and again missing date or time elments are assumed to have the lowest value. Therefore, clients must be able to transparently handle truncated ISO strings of both flavors. For `binary` and `csv` data, a truncated time string is indicated by setting the `length` attribute for the time parameter. See https://en.wikipedia.org/wiki/ISO_8601.

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

Additional Keyword / Value Pairs

While the HAPI server strictly checks 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.

More About

Data Types

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).

The ‘size’ Attribute

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. In both cases, the later indices are faster moving, so that if you have a 2-D array of "size":[2,5] then 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.

'fill' Values

Note that fill values for all types must be specified as a string. 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 shoudl be taked 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 an 4 byte integer. For double parameters, the fill string must parse to an exact IEEE 754 double representation. One suggestions is to use large negative integers, such as -1.0E30. The string NaN is allowed, in which case csv output should contain the string NaN for fill values. For double NaN values, the bit pattern for quiet NaN should be used, as opposed to the signaling NaN, which should not be used (see reference [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.

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": "1.1",
  "status": { "code": 1200, "message": "OK"},
  "startDate": "2016-01-01T00:00:00.000",
  "stopDate": "2016-01-31T24:00:00.000",
  "parameters": [
    {"name" : "timestamp", "type": "isotime", "units": "UTC", "length": 24},
    {"name" : "bx", "type": "double", "units": "nT"},
    {"name" : "by", "type": "double", "units": "nT"},
    {"name" : "bz", "type": "double", "units": "nT"}
  ]
}
```

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": "1.1",
  "status": { "code": 1200, "message": "OK"},
  "startDate": "2016-01-01T00:00:00.000",
  "stopDate": "2016-01-31T24:00:00.000",
  "parameters": [
    { "name" : "timestamp", "type": "isotime", "units": "UTC", "length": 24 },
    { "name" : "b_field", "type": "double", "units": "nT","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://hapi-server.org/hapi/data?id=MY_MAG_DATA&time.min=2001&time.max=2010&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 are a separate variable. There is currently no way in the HAPI spec to explicitly link a variable to its uncertainties.

```
{"HAPI": "1.1",
 "status": { "code": 1200, "message": "OK"},
 "startDate": "2016-01-01T00:00:00.000",
 "stopDate": "2016-01-31T24:00:00.000",
 "parameters": [
  { "name": "Time",
    "type": "isotime",
    "units": "UTC",
    "length": 24
  },
  { "name": "qual_flag",
    "type": "int",
    "units": null
  },
  { "name": "maglat",
```

```
    "type": "double",
    "units": "degrees",
    "description": "magnetic latitude"
  },
  { "name": "MLT",
    "type": "string",
    "length": 5,
    "units": "hours:minutes",
    "description": "magnetic local time in HH:MM"
  },
  { "name": "proton_spectrum",
    "type": "double",
    "size": [3],
    "units": "particles/(sec ster cm^2 keV)"
    "bins": [ {
        "name": "energy",
        "units": "keV",
        "centers": [ 15, 25, 35 ],
         } ],
  { "name": "proton_spectrum_uncerts",
    "type": "double",
    "size": [3],
    "units": "particles/(sec ster cm^2 keV)"
    "bins": [ {
        "name": "energy",
        "units": "keV",
        "centers": [ 15, 25, 35 ],
         } ]
  }

 ]
}
```

This shows how "ranges" can specify the bins:

```
{
   "HAPI": "1.1",
   "status": { "code": 1200, "message": "OK"},
   "startDate": "2016-01-01T00:00:00.000",
   "stopDate": "2016-01-31T24:00:00.000",
   "parameters": [
      {
         "length": 24,
         "name": "Time",
         "type": "isotime",
         "units": "UTC"
```

```
    },
    {
      "bins": [{
        "ranges": [
          [ 0, 30 ],
          [ 30, 60 ],
          [ 60, 90 ],
          [ 90, 120 ],
          [ 120, 150 ],
          [ 150, 180 ]
        ],
        "units": "degrees"
      }],
      "fill": -1.0E38,
      "name": "pitchAngleSpectrum",
      "size": [6],
      "type": "double",
    "units": "particles/sec/cm^2/ster/keV"
    }
  ]
}
```


Security Notes

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

So given this query

```
http://hapi-server.org/hapi/data?id=DATA&time.min=T1&time.max=T2&fields=mag_GSE&avg=5s
```

the server should throw an error with a status of "1400 - Bad Request" with 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 names values. Unknown request parameters and values, including incorrectly formatted time values, should **not** be echoed in the error response.

Adoption

In terms of adopting HAPI as a data delivery mechanism, data providers will likely not want to change existing services, so a HAPI compliant access mechanism could be added alongside existing services. Several demonstration servers exist, but there are not yet any libraries or tools available for providers to use or adapt. These will be made available as they are created. The 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, and Autoplot currently can access HAPI compliant servers. Eventually, libraries in several languages will be made available to assist in writing clients that extract data from HAPI servers. However, even without 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.

References

[1] ISO 8601:2004, http://dotat.at/tmp/ISO_8601-2004_E.pdf
[2] CSV format, https://tools.ietf.org/html/rfc4180
[3] JSON Format, https://tools.ietf.org/html/rfc7159
[4] "JSON Schema", http://json-schema.org/
[5] EEE Computer Society (August 29, 2008). "IEEE Standard for Floating-Point Arithmetic". IEEE. doi:10.1109/IEEESTD.2008.4610935. ISBN 978-0-7381-5753-5. IEEE Std 754-2008
[6] IEEE Standard 754 Floating Point Numbers, http://steve.hollasch.net/cgindex/coding/ieeefloat.html

Contact

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