Abstract

The Planetary Data System is a federated system of nodes that archive planetary science data. The PDS Application Programming Interface (API) is an attempt at provide a consistent interface for sharing archival data and services across PDS, among planetary archives, and within the planetary science community. The API is one of the cornerstone applications for providing an integrated worldwide data services platform that enables the efficient discovery, dissemination, use and analysis of internationally sponsored planetary science archives.

The latest released version of this document is currently maintained on the PDS API Github Repo, and the development version is maintained on Google Drive. Feedback and comments welcome.

- Released: https://github.com/NASA-PDS/pds-api/blob/main/docs/spec/pds-api-specification.md
- In Development: https://docs.google.com/document/d/16d0MVh48bFLvWsa5-B_Hy-cby1rGWdnNojWOJpUcOvA/edit?usp=sharing

Reference Documents

Several websites and documents were used as references for designing this API and the accompanying guidelines, including:

1. Open API Initiative
2. Open APIs Specification
3. Microsoft API Guidelines
4. Microsoft API Design Best Practices
5. NASA Earth Data APIs
6. Google Custom Search REST API
7. EPN-TAP
8. Earth Data Common Metadata Repository (CMR)
9. Swagger for Developing API Spec
10. Open Search
11. Library of Congress Search/Retrieval by URL
12. PDS OPUS API
13. PDS Imaging Atlas API
14. OGC Environmental Data Retrieval

General Applicable Open API Conventions

Specification standard

The API complies with open api 3.0.

Restful principles

Resources

Resources are coded as URI (e.g. http://domain/api/pets). Resources should be nouns (verbs are bad)

Verbs

Users interact with resources through HTTP request verbs. The PDS API uses GET and POST:

- GET is relevant to get resource representation from the API when the extraction criteria is simple.
- POST, in a read-only context, is relevant to provide the API with complex request criteria.

Future iterations of the API will transform it to be an idempotent REST API, utilizing GET, PUT, DELETE, HEAD, OPTIONS and TRACE HTTP methods.

Resource representation

When a HTTP request verb (e.g. GET, POST, etc.) is applied to a resource (e.g. http://domain/api/pets) he/she gets a resource representation.

Here are some characteristics of

Many different flavors of resource representations are possible to be returned from a single resource. For example, subsets of a whole, formats, versions, etc..

The resource representation should be self-described as much as possible.

They can be wrapped in envelopes which prevent from vulnerabilities linked to the direct access to json arrays in javascript code (see https://haacked.com/archive/2008/11/20/anatomy-of-a-subtle-json-vulnerability.aspx/). A response with this format is fine:

```
{ metadata: {...}, data: [...] }
```

Other conventions

Beyond the OpenAPI standard, there are multiple options regarding general design of an API. We primarily use the following source which is very complete and not too dogmatic: https://www.moesif.com/blog/api-guide/api-design-guidelines/

Some peer web API specifications are also considered as references for the design for the PDS API specification:

- OGC environment data retrieval: http://docs.opengeospatial.org/DRAFTS/19-086.html
- ESDIS Common Metadata Repository API: https://earthdata.nasa.gov/collaborate/open-data-services-and-software/api/cmr-api

The chosen options are listed here after:

(TBD once spec is built out in Swagger)

URL Resource Naming: Case

Kebab-case (lower case and hyphens â€˜-â€™ used to fill the spaces in) is used for url resource naming.

For example:

- `discipline-nodes` in http://pds.nasa.gov/0.1/api/discipline-nodes

(see https://stackoverflow.com/questions/10302179/hyphen-underscore-or-camelcase-as-word-delimiter-in-uris).

URL Resource Naming: Plural vs singular

Resources are named plural or singular depending on the use case.

Plural are used when the resources is a collection the user will subset from, for example `/pets/scooby-doo` or `/planets/mars` or â€˜/profiles/user-idâ€™ or â€˜collections?q=...â€™â€™

Singular are used when the resource is accessed as one. For example â€˜/profileâ€™ to access the profile of the current user.

See https://medium.com/@atomaka/single-and-plural-rails-routes-for-the-same-resource-330d985b6595 for more details.

URL Resource Naming: API Versioning

The API will have versions and the deployed versions are likely to be heterogeneous in the PDS system.

Two options have been considered to manage versions (see https://restfulapi.net/versioning/):

- Version in the URL, e.g. pds.nasa.gov/1.0/products/
- Content negotiation headers (e.g. Accept: application/vnd.example+json;version=1.0)

To keep things as simple as possible, content negotiation will not be used for version management. A server API implementation will implement a single version of the API definition.

However:

- We advise to use the version in the URL of the API when it is deployed, although it is not part of the API definition.
- The version is mandatory in the resource representations (result of a request)
- The version will be managed (optional) in the â€˜routesâ€™ resource to be able to manage deployments of implementations of different version of the API definition. For example, GEO Discipline Node can provide 2 end-points for the API, v1.0 and v1.1. The routing should allow a client to choose between these end-points. The latest will be provided by default.

Pagination/Sort

The query parameters for pagination are:

| start | Index of first item returned in the response | |-------|---| | limit | Maximum number of item expected in the response |

See https://www.moesif.com/blog/technical/api-design/REST-API-Design-Filtering-Sorting-and-Pagination/

PDS Search API

General Request Details

Maximum URL Length

The Maximum URL Lengths supported by the PDS Search API are:

- For programmatic GET requests - 6,000 characters
- For web browser access - 2048 characters

Clients using the Search API with query parameters should be careful not to exceed this limit or they may get an HTTP response of 413 FULL HEAD due to server or browser performance restrictions. If a client expects that the query url could be extra long so that it exceeds 6k characters, they should use the a POST request through the JSON Request API for searching.

Query Syntax

The query string query parameters, *q*, allows a client to filter the results from an API request. The expression specified with *q* is evaluated for each resource in the collection, and only items where the expression evaluates to true are included in the response. Resources for which the expression evaluates to false or to null, or which reference properties that are unavailable due to permissions, are omitted from the response.

The PDS Search API also supports wild cards ? and *. A search with no *q* parameter specified will default to *q=** (search for all possible records).

Example: return all collections whose start time is prior to June 1, 2020:

```
GET https://pds.nasa.gov/api/v0.1/collections?q=Time_Coordinates.start_date_time LT 2020-06-01T00:00:00Z
```

Query String Operations

The PDS Search API supports the following minimal set of operations.

Operator	**Description**	**Example**
Comparison Operators		
eq	Equal	target_name ***eq*** "Mars"
ne	Not equal	target_name ***ne*** "Saturn"
gt	Greater than	Time_Coordinates.start_date_time ***gt*** 2001-05-10T00:00:00Z
ge	Greater than or equal	Time_Coordinates.start_date_time ***ge*** 2001-05-10T00:00:00Z
lt	Less than	Time_Coordinates.start_date_time ***lt*** 2020-06-01T00:00:00Z
le	Less than or equal	Time_Coordinates.start_date_time ***le*** 2020-06-01T00:00:00Z
Logical Operators		
and	Logical and	target_name ***eq*** "Mars" ***and*** instrument_name ***eq*** "hirise"
or	Logical or	target_name ***eq*** "Mars" ***or*** target_name ***eq*** "Phobos"
not	Logical negation	***not*** target_name ***eq*** "Mars"
Grouping Operators		
( )	Precedence grouping	***(***target_name ***eq*** "Mars" ***or*** target_name ***eq*** "Phobos"***)*** ***and*** instrument_name ***eq*** "hirise"

Reserved Query Parameters

The following are a table of reserved query parameters that have special meaning to support search.

Query Parameter	**Description**	**Example**
q	(Required, string) Query string you wish to parse and use for search. See Query string syntax.	`q=target_name eq "Mars"`
fields	(Optional, array of strings) Array of fields you wish to return.	`fields=lid,Time_Coordinates.start_date_time`
start	(Optional, integer, default=0) The search result to start with in the returned records. For instance, start=10 will return records 10-19.	`start=100`
limit	(Optional, integer) The number of records/results to return. Defaults to 25.	`limit=100`
sort	(Optional, string, default=LIDVID) Field to sort on and whether it should be sorted ascending (ASC) or descending (DESC). `fieldName asc` or `fieldName desc`. There can be several sort parameters (order is important).	`sort=lidvid asc, Time_Coordinates.start_date_time desc`

Endpoints

Based upon the PDS API Specification guidelines for Request URL Specification, the following are the base URIs for performing GET requests for searching PDS data.

For collection-level search:

Search All PDS Data Collections:

```
GET https://pds.nasa.gov/api/search/0.1/collections?{searchTerms}
```

Search Geosciences Node Collections:

```
GET https://pds.nasa.gov/api/search-geo/0.1/collections?{searchTerms}
```

See Query Parameters for more specific examples for various searches with different parameters.

For product-level search:

Template:

```
GET https://{node-hostname}/api/{service}/{pds.api.version}/products?{searchTerms}
```

Examples:

Search All PDS Data Products:

```
GET https://pds.nasa.gov/api/search/0.1/products?{searchTerms}
```

Search Geosciences Node:

```
GET https://pds.nasa.gov/api/search-geo/0.1/products?{searchTerms}
```

See Query Parameters for more specific examples for various searches with different parameters.

Additional Query String Examples

``` /v0.1/collections?q="2018-01-01" LE Time_Coordinates.start_date_time LE "2020-01-01"&start=100&limit=1000

/v0.1/products?q=Observing_System_Component.description EQ "ISSNA" AND (Optical_Filter.filter_name EQ "BL1" OR Optical_Filter.filter_name EQ "GRN")&fields=cassini.spacecraft_clock_start_count ```

## POST Request API

*For now, we may just want the POST request API to be pretty much identical to the GET, except you can embed the Response Format* `return_type` *in the HTTP header*

*If we wanted to create a Domain Specific Language (DSL), here are some references:*

- *ElasticSearch Query DSL: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html*
- *Solr Query DSL: https://lucene.apache.org/solr/guide/7_1/json-query-dsl.html*
- *ESDIS CMR Example JSON Request API: https://cmr.earthdata.nasa.gov/search/site/docs/search/api.html#search-with-json-query*

## Response Formats

Working Document: PDS API - Response Formats

A simple style of content negotiation is used to match the format requested by the client and the capability of the server. The client can specify the desired response format by including the HTTP header Accept. If no Accept header is present in the request, or if the requested content type is not available, the server will provide the response in JSON format by default.

The following table provides a list of the supported HTTP Accept header types:

| **Accept Header** | **Format** | **Note** | |-------------------|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------| | application/json | JSON | JS
response customized for PDS metadata. Contains â€œflattenedâ€ PDS4 metadata extracted from the metadata labels. | | application/xml | XML | Same as application/json, but in an XML format. | | application/pds4+json | JSON | JSON response containing the full PDS4 metadata translated to JSON, along with some additional supplemental metadata. | | application/pds4+xml | XML | Same as application/pds4+json, but in an XML format. | | application/kvp+json | JSON | JSON response containing key-value-pairs for the applicable metadata. | | text/csv | CSV | Returns a CSV table containing values for the parameters in the request. If no parameters were specified in the request, a default set should be returned. The first row of the CSV is a header row that describes the values in each column. |

Given the following sample HTTP request, a client can expect supported response formats to be consistent with the following examples:

https://pds.nasa.gov/api/v0.1/collections?q=identifier eq urn:nasa:pds:orex.ocams:data_raw&start eq 1&limit eq 1

### Missing values

Properties with empty or null values should be dropped from the JSON response unless the user asked specifically for the field (through `field` API parameter). In this case the value must be **null**, without quotes.

#### Rationale

If a property is optional or has an empty or null value, consider dropping the property from the JSON, unless there's a strong semantic reason for its existence (taken from https://softwareengineering.stackexchange.com/questions/285010/null-vs-missing-key-in-rest-api-response)

Following interactions with OGC/EDR specification group: https://github.com/opengeospatial/ogcapi-environmental-data-retrieval/issues/171#issuecomment-767805902

We choose **null** without quotes for missing values of fields explicitly requested by the user.

We conform to EDR specification for this aspect, see http://docs.opengeospatial.org/DRAFTS/19-086.html#req_edr_parameters-response

This should not be mistaken for an actual PDS4 value since missing values in PDS4 labels. are detailed with a nil:reason attribute.

### JSON
```

```
{ identifier: urn:nasa:pds:orex.ocams:data_raw, start: 1, limit: 1, total_results: 1, â€œresultsâ€: [ { â€œlidâ€:
urn:nasa:pds:orex.ocams:data_raw, title: â€˜OSIRIS-REx OCAMS raw science image data productsâ€™, instrument_host_name:
â€˜Rosetta Orbiterâ€™, instrument_name: â€˜OSIRIS-REx Camera Suite (OCAMS)â€™, target_name: â€˜101955 Bennuâ€™ } ] }
```

XML

```
<instrument_host>urn:nasa:pds:orex.ocams:data_raw</instrument_host>

<start>1</start>

<limit>1</limit>

<total_results>1</total_results>

<results>

<lid>urn:nasa:pds:orex.ocams:data_raw</lid>

<title>OSIRIS-REx OCAMS raw science image data products</title>

<instrument_host_name>Rosetta Orbiter</instrument_host_name>

<instrument_name>OSIRIS-REx Camera Suite (OCAMS)</instrument_name>

<target_name>101955 Bennu</target_name>

</results>
```

CSV

```
lid,title,instrument_host_name,instrument_name,target_name

urn:nasa:pds:orex.ocams.data_raw,OSIRIS-REx OCAMS raw science image data products,Rosetta Orbiter,OSIRIS-REx Camera Suite
(OCAMS),101955 Bennu
```

Open Data

See https://project-open-data.cio.gov/ and example of application at https://cmr.earthdata.nasa.gov/search/site/docs/search/api.html#open-data

Example Response

TBD

Query Parameters

Query parameters are the variables used to specify values in a query string or as part of an API for searching for specific values in a database.

The following are the defined set of common query parameters for PDS defined as a query model in the PDS4 Information Model.

Collection-level Search

The initial revision of the query parameters will be based upon the PDS Common Dictionary Query Model (designed by the Data Design Working Group (DDWG)), which was intended to enable searching at the ***data collection level***. Some of the query model parameters may not apply or will be grouped together, but these are the values that were decided would be most helpful for searching for a particular data collection. Below are the attributes of most interest:

- Product_Collection.collection_type=**Data** (ignored in query parameters)
- Primary_Result_Summary.processing_level
- Investigation_Area.type=**Mission** (ignored in query parameters)
- Science_Facets.domain
- Identification_Area.product_class=**Product_Collection** (ignored in query parameters)
- Science_Facets.wavelength_range
- Primary_Result_Summary.purpose=**Science** (ignored in query parameters)
- Science_Facets.discipline_name
- Time_Coordinates.start_date_time
- Collection.description
- Time_Coordinates.stop_date_time
- Product_Context.Instrument.type
- Target_Identification.name
- Identification_Area.logical_identifier
- Target_Identification.type
- Identification_Area.version_id
- Investigation_Area.name
- Science_Facets.facet1
- Observing_System.name
- Science_Facets.facet2

- Observing_System_Component.name
- Primary_Result_Summary.description
- Observing_System_Component.type=**Instrument** (ignored in query parameters)
- Target_Identification.description
- Identification_Area.title
- system.archive_status
- Citation_Information.keyword
- Citation_Information.description

These are all specified in the PDS Common tab of the PDS Query Parameters spreadsheet.

Product-level Search

For enabling product-level search, the collection-level search query model should be extended for each discipline-specific implementation. For starters, we can simply add parameters we deem useful to the specification. However, in the end, all query parameters should be specified in the PDS4 Information Model and the applicable local data dictionary, either by query model or some other TBD mechanism.

Each discipline node should create a tab or update their applicable tab in the PDS Query Parameters spreadsheet with parameters useful for their specific discipline node or product search. Ideally attempting to follow the guidelines for specifying query parameters noted above.

Additional Example Search Scenarios

TBD

PDS REST Web Services supporting the API federation

URN resolver for LIDVID

The PDS has a powerful URN scheme for naming its resources.

The PDS API will provide a service to resolve URN wherever the resource identified is managed.

Routes

The PDS API end-points will be distributed in the PDS system and hosted by Discipline Nodes and Engineering Nodes.

To act as a federated API system, the API should enable redirection of clientsâ€™ requests to the most appropriate API end-point depending on user requests.

Other PDS REST Web Services

TBD a more RESTful approach to searching the registry

Implementation

4 Stages:

- Standard preparation: swaggerHub
- Standard definition: one yml or json file on a github repository
- Standard libraries: python and java standard implementation (client,severs stubs) shared on PYPI and MAVEN artifactory

- Standard implementations, by Engineering Node (demo, validator, core) and Discipline Nodes (actual access).

3.1 Preparation

The definition of the API is prepared in swaggerhub: https://app.swaggerhub.com/apis/PDS_APIs/pds_federated_api

3.2 Definition

The definition of the API is published in a yml and json file in a public github repository.

https://github.com/NASA-PDS/pds-api-base

3.3 Libraries

Libraries in JAVA and PYTHON are generated from the standard definition.

They include client libraries and server stubs.

They are published on PYPI and MAVEN Central Repository.

TO BE DONE, UNDER PROGRESS

3.4 Implementations

The foreseen implementations are:

- Discipline node servers
- Validator/Demo client
- Demo server
- Core server (router/proxy)

They can be in any language but may use the standard libraries, which should help to comply with the standard AND implement basic behaviours (route, urn resolution).

Acknowledgements

PDS API Working Group

Mcclanahan, Timothy (PDS PO) Lynn Neakrase (ATM) Ed Guiness (GEO) Tom Stein (GEO) Dan Scholes (GEO) Mark Bentley (ESA) Myche McAuley (IMG) In Sook Moon (PPI) Rob French (RMS) Matt Tiscareno (RMS) Conor Kingston (SBN) David Chang (SBN) Daniel Darg (SBN) Emily Law (EN) Yevgen Karpenko (EN) Thomas Loubrieu (EN) Jordan Padams (EN) Boris Semenov (NAIF)

References

Portions of these guidelines were adapted from Microsoft API Guidelines licensed under the Creative Commons Attribution 4.0 International License

Appendix A - API Definition and Application Guidelines

This appendix is intended for the PDS Node to provide guidelines for defining and applying the PDS API Spec.

Goals

- To define a common, ***RESTful API specification*** to be adopted across the PDS.
- To define a ***common syntax and best practices for defining query parameters***.
- To define a ***common set of query parameters for top-level search criteria*** with the widest-ranging applicability across all PDS data.
- To define a set of best practices and processes for extending the common set of query parameters with ***discipline- and node-specific query parameters***.
- To provide a ***managed central location for describing and documenting all API specification details***, including query parameters, across the entire PDS.

Request URL Specification

Consistency of URI definitions is important for ubiquitous use of APIs across a federated system. Additionally, it enables the federated system to integrate as an architecture of microservice architecture. The following is a general guideline for PDS REST API endpoints:

https://{hostname}/api/v{pds.api.version}/{controller}?{searchTerms}

where {hostname} is the web hostname (e.g. https://pds.nasa.gov),

{controller} is some object controller e.g. search, translate, transform

{pds.api.version} is the version of the API being used.

Defining Query Parameters

Query Parameter Governance

Governance and stewardship over query parameters follows a similar pattern to the PDS4 Information Model in that it is a tiered governance model.

Defining Query Parameters using Query Models

In order to leverage the PDS4 Information Model for specifying these parameters, we can use the notion of *query models*. Query models are a means for discipline dictionary developers, or steward, to specify attributes they deem useful for search that they believe data providers will search by. As the discipline dictionary steward, they are close to the scientists and have a good understanding of their user community needs. It makes the most sense for that information to be specified by those specialists.

To get started, we will use the PDS Common Dictionary query model for top-level search integration, and then expand to the discipline dictionaries.

Query Parameter Formation Specification

The PDS4 Standard is the backbone of the future for the PDS, where careful consideration has been taken in the naming of all classes and attributes throughout the information model. Instead of creating our own names or using legacy PDS3 naming, we should leverage PDS4 as much as possible.

Dot Notation

The syntax will use a combination of the PDS4 Information Model and dot notation representations of an XML XPaths (with some exceptions).

Query parameters will use a combination of an attribute with its parent class in *all lowercase*:

{parent_class}.{attribute}

In the event that the {parent_class}.{attribute} combination does sufficiently guarantee uniqueness or sufficiency of search when a class is inherited by multiple classes, additional ancestor classes should be prepended to the query parameter until sufficient uniqueness is attained:

{ancestor_class}.{parent_class}.{attribute}

If the query parameter grows beyond 3 ancestor classes, a custom query parameter should be considered.

In the event that multiple attributes are to be grouped together for search, the parent class should be used as the query parameter:

{ancestor_class}.{parent_class}

Examples

Query Parameter	**XPath**
`Observing_System.description`	`//Observation_Area/Observing_System/description`
`geom.Camera_Model_Parameters.geom.model_type`	`//geom:Camera_Model_Parameters/geom:model_type`

Exceptions and Nuances

There are some exceptions to the query syntax guidelines:

1. Custom Query Parameters

There are several cases where custom query parameters are preferred over the Dot Notation, but should only be avoided wherever possible in order to minimize confusion amongst developers attempting to use the API. These are also subject to approval by Search Integration Working Group representative for each node. That member is responsible for providing those updates to Engineering Node.

Some reasons for custom query parameters:

- Combination of multiple attribute values into one
- Special cases where XQuery needs to be used for finding specific values (e.g. instrument/spacecraft described in Observing_System_Component class)
- Custom search fields on non-PDS4 metadata (e.g. image tags, operations note, etc.)
- Support common search or PDS4 terminology (e.g. target_name, lidvid)

2. Internal / External References

Because Internal and External References are inherited throughout discipline/mission dictionaries, the ancestor class should replace the parent. In addition, since lid_reference and lidvid_reference are both referring some identifier, we replace those both with id_reference, e.g.:

Query Parameter	**XPath**
Observing_System_Component.id_reference	//Observing_System_Component/Internal_Reference/lidvid_reference
Observing_System_Component.id_reference	//Observing_System_Component/Internal_Reference/lid_reference