**Open Science Data Framework (OSDF) API**

Contents

[1. Authentication 2](#_Toc318820534)

[2. Show OSDF information 2](#_Toc318820535)

[3. Namespaces 3](#_Toc318820536)

[a) List Available Namespaces 3](#_Toc318820537)

[b) Retrieve Namespace Details 5](#_Toc318820538)

[4. Access Control (ACLs) 5](#_Toc318820539)

[a) List ACL Membership 5](#_Toc318820540)

[b) Edit ACLs 6](#_Toc318820541)

[5. Controlled Vocabularies 6](#_Toc318820542)

[a) List Namespace Vocabularies 7](#_Toc318820543)

[b) Retrieve Namespace Vocabulary 8](#_Toc318820544)

[i) By ID 8](#_Toc318820545)

[ii) By name 9](#_Toc318820546)

[c) Create Namespace Vocabulary 11](#_Toc318820547)

[d) Edit Namespace Vocabulary 11](#_Toc318820548)

[e) Delete Namespace Vocabulary 12](#_Toc318820549)

[6. Node templates 12](#_Toc318820550)

[a) List templates 13](#_Toc318820551)

[b) Retrieve template 15](#_Toc318820552)

[i) By ID 15](#_Toc318820553)

[ii) By name 16](#_Toc318820554)

[c) Create template 17](#_Toc318820555)

[d) Edit Template 18](#_Toc318820556)

[e) Delete template 18](#_Toc318820557)

[7. Node management 18](#_Toc318820558)

[a) Create a Node 18](#_Toc318820559)

[b) Retrieve a Node 20](#_Toc318820560)

[c) Retrieve a Previous Node Version 22](#_Toc318820561)

[d) Retrieve Node Linkage 24](#_Toc318820562)

[e) Edit a Node 25](#_Toc318820563)

[f) Delete a Node 27](#_Toc318820564)

[8. Querying 27](#_Toc318820565)

[a) Making Queries 27](#_Toc318820566)

[b) Elasticsearch DSL examples 28](#_Toc318820567)

[c) Sorting Results 31](#_Toc318820568)

[d) Range / Paginated Results 32](#_Toc318820569)

# 1. Authentication

Authentication to the OSDF is managed with “Basic” HTTP authentication and each request must include the “Authorization” HTTP header. Use your assigned OSDF username and the API token as the password. The initial implementation of OSDF will not actually check for the correctness of the API token or whether it is the correct value for the specified username, however, the service will check that the Authorization header is present and well formed. In a later version, each user will need to apply for a username and a bona fide API token via a web form/application.

Requests that do NOT provide an “Authentication” header will result in an HTTP 401 (Unauthorized) response code.

|  |
| --- |
| Example request with the “curl” utility |
| Username: jdoe  API key : b1946ac92492d2347c6235b4d2611184  $ curl **-u jdoe:b1946ac92492d2347c6235b4d2611184** -G <OSDF\_URL>/nodes/4 |

For the remainder of this API documentation, the combination of the username and the API token, will be referred to as "<AUTH>" for brevity.

# 2. Show OSDF information

Basic information about the OSDF instance, such as description, contact information, and the version of the API (which will change over time) may be retrieved with a simple HTTP GET request. The "title", "comment1" and "comment2" fields are limited to 128 characters, the "description" is limited to 512 characters. The email contact fields values must adhere to RFC 5322 and RFC 5321 but are further limited to a maximum length of 128 characters.

|  |  |
| --- | --- |
| Retrieve OSDF information | GET /info |
| Request | |
| Example request:  $ curl -u <AUTH> -G <OSDF\_URL>/info | |
| Response (application/json) | |
| A GET to the URL will yield an array of namespaces.  Example Response:  {  "api\_version" = "<api\_version>",  "title": "<title1>",  "description": "<description1>",  "url": "<url1>",  "admin\_contact\_email1": "<admin\_email1>",  "admin\_contact\_email2": "<admin\_email2>",  "technical\_contact1": "<tech\_email1>",  "technical\_contact2": "<tech\_email2>",  "comment1": "<comment1>",  "comment2": "<comment2>"  }  Concrete example:  {  "api\_version" = "1.1.6",  "title": "IGS-OSDF",  "description": "This is an instance of the Open Science Data Framework (OSDF) at the Institute for Genome Sciences, University of Maryland School of Medicine.",  "url": "http://osdf.igs.umaryland.edu",  "admin\_contact\_email1": "osdf-admin@som.umaryland.edu",  "admin\_contact\_email2": "help@som.umaryland.edu",  "technical\_contact1": "osdf-helpdesk@som.umaryland.edu",  "technical\_contact2": "osdf@som.umaryland.edu",  "comment1": "801 W. Baltimore Street, 6th Floor",  "comment2": "Baltimore, MD 21201"  } | |

# 3. Namespaces

Each namespace associated with the OSDF will have its own specific controlled vocabularies that determine how a developer will interact with it. Namespaces will be globally unique. In other words, there cannot be a namespace name collision. Namespaces names beginning with the string "osdf" are reserved for internal system and implementation use and are prohibited from use by end users. Namespace names are limited to 32 characters and descriptions to 256 characters. For the namespace name, each character must correspond to the alphanumeric ASCII character set (A-Za-z0-9) as well as the underscore (\_) and hyphen (-).

## List Available Namespaces

To retrieve a list of namespaces:

|  |  |
| --- | --- |
| Retrieve Namespace List | GET /namespaces |
| Request | |
| An HTTP GET to the URL will yield the list of the namespaces.  Example request:  $ curl –u <AUTH> –G <OSDF\_URL>/namespaces | |
| Response (application/json) | |
| A GET to the URL will yield a JSON object with namespace names as key.  Example Response:  {  "<ns1>": { "title": "<title1>",  "description": "<description1>",  "url": "<url1>" },  "<ns2>": { "title": "<title2>",  "description": "<description2>",  "url": "<url2>" },  "<nsN>": { "title": "<titleN>",  "description": "<descriptionN>",  "url": "<urlN>" }  }  Concrete example:  {  "hmp": { "title": "The Human Microbiome Project",  "description": "The aim of the Human Microbiome Project is to characterize microbial communities found at multiple human body sites and to look for correlations between changes in the microbiome and human health.",  "url": "http://www.hmpdacc.org" },  "mg-rast": { "title": "Metagenomics Analysis Server",  "description": "The Metagenomics Analysis Server is an automated analysis platform for metagenomes providing quantitative insights into microbial populations based on sequence data.",  "url": "http://metagenomics.anl.gov" }  } | |

## Retrieve Namespace Details

|  |  |
| --- | --- |
| Retrieve Namespace Details | GET /namespaces/${ns} |
| Request | |
| A GET to the URL will yield specific namespace details.  Example request:  $ curl –u <AUTH> –G <OSDF\_URL>/namespaces/hmp | |
| Response (application/json) | |
| A JSON object with the details of the requested namespace.  Example Response:  {  "<ns>": {  "title": "<title>",  "description": "<description>",  "url": "<url>"  }  }  Concrete example:  {  "hmp": {  "title": " The Human Microbiome Project",  "description": "The aim of the Human Microbiome Project is to characterize microbial communities found at multiple human body sites and to look for correlations between changes in the microbiome and human health.",  "url": "http://www.hmpdacc.org"  }  } | |

# 4. Access Control (ACLs)

ACL names have the same restrictions that namespace names and controlled vocabulary names have: a maximum of 32 characters and using the ASCII alphanumeric character set (A-Za-z0-9) plus underscores (\_) and hyphens (-). Additionally, the "all" value for ACL names is reserved and taken to mean global access, even across namespaces.

## List ACL Membership

Users may retrieve a list of the ACL groups they are members of. Users may belong to various ACLs in different namespaces, so the result set contains a list of ACLs for each namespace the user is a member of. Users do not have the ability to see what ACLs and namespaces other users are members of.

|  |  |
| --- | --- |
| Retrieve ACL List by Username | GET /acls |
| Response (applicaton/json) | |
| A GET to the URL will yield information on ACL membership.  Abstract form:  {  "<ns1>": [ "<acl1>", "<acl2>", "<aclM>" ],  "<ns2>": [ "<acl3>", "<acl4>", "<aclN>" ]  }  Concrete example:  {  "hmp": [ "dawg", "dacc", "research\_network" ],  "mg-rast": [ "uploaders", "web\_devel" ]  } | |
| An invalid API token will result in an HTTP 403 "Forbidden" status code. Other errors will result in an HTTP 500 "Server Error" response. | |

## Edit ACLs

# 5. Controlled Vocabularies

Each namespace may have multiple controlled vocabularies (CVs) to describe nodes and their connections. A controlled vocabulary is simply a flat list of terms that the namespace participants have agreed to use help describe their data and relationships. In OSDF vocabularies are used to define the node types, as well as the linkages between nodes. Therefore, every namespace will automatically be assigned 2 empty vocabularies: "node\_types" and “linkage” for those two functions.

For instance, a namespace can choose to have node connections, or edges, characterized as "derived\_from". Perhaps other node connections should be called "collected\_from", "computed\_from", or "is\_a". Before nodes can be created with connections to other nodes, the "linkage" CV must be populated with this list of possible connection names. For node\_types, the namespace can choose to have nodes called "sample", "subject" and "analysis". This list of terms must be entered into the "node\_types" CV. Since these two CVs are fundamental to the operation of OSDF, they are held in the “osdf” key. These vocabularies are intended to be populated, or "filled in", by the namespace administrators/users. Additional "custom" vocabularies are also possible, but when defined, will appear in the “ns\_cv” key of the returned data structure. Each entry has the vocabulary name as a key, and a brief (<= 256 character) description. Custom vocabulary names are limited to the same restrictions as namespace names which is a maximum of 32 characters in the ASCII alphanumeric set: (A-Za-z0-9) plus underscore (\_) and hyphen (-). Controlled vocabularies beginning with "osdf" are restricted for internal use.

## List Namespace Vocabularies

To retrieve a list of the vocabularies for a namespace:

|  |  |
| --- | --- |
| Retrieve a namespace’s vocabularies | GET /namespaces/${ns}/vocabs |
| Request | |
| A GET to the URL will yield the list of controlled vocabularies.  Example request:  $ curl –u <AUTH> –G <OSDF\_URL>/namespaces/hmp/vocabs | |
| Response (application/json) | |
| A JSON object with the namespace vocabulary names as keys.  Example Response:  {  "result\_count": <result\_count>,  "page": <page\_count>,  "results": [  {  "ns": "<namespace\_id>",  "id": "<id1>",  "vocab\_name": "<vocab\_name1>",  "vocab\_url": "http://cv.example.com/example.cv",  "vocab\_description": "<brief description of the vocabulary>",  "acl": { "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>", "<aclN>" ]  },  "terms": [  "<term1>",  "<term2>,  "<term3>",  "<termN>"  ]  },  {  "ns": "<namespace\_id>",  "id": "<id2>",  "vocab\_name": "<vocab\_name2>",  "vocab\_url": "http://cv.example.com/example.cv",  "vocab\_description": "<brief description of the vocabulary>",  "acl": { "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>", "<aclN>" ]  },  "terms": [  "<term4>",  "<term5>,  "<term6>",  "<termM>"  ]  }  ]  } | |
| Failed requests (unrecognized namespaces) will yield HTTP 422 ("Unprocessable Entity") or 500 ("Server error") responses.  Security related errors for users attempting to access data without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" response. | |

If the result is too large to return in a single response, the server will return a partial result set of vocabularies, and an HTTP 206 response (“Partial content”) code. The “X-OSDF-Vocab-ResultSet” header will then contain a URL to allow the retrieval of the next page of vocabularies. If, in a vocabulary set response, the XOSDF-Vocab-ResultSet header is absent, then there are no further vocabularies available.

## Retrieve Namespace Vocabulary

An HTTP GET request must be made to retrieve the definitions for a particular vocabulary list. A JSON document is returned listing the allowable values for each part of the vocabulary list.

### By ID

|  |  |
| --- | --- |
| Retrieve Vocabulary Definition by ID | GET /namespaces/${ns}/vocabs/ids/${vocab\_id} |
| Request | |
| Request a description and listing of a controlled vocabulary.  Example request:  $ curl –u <AUTH> –G \  <OSDF\_URL>/namespaces/hmp/vocabs/ids/71hbRmdih6hf5Yqwdjtl | |
| Response (application/json) | |
| A GET to the URL will yield a JSON object defining the vocabulary for the named list in that namespace.  Example response:  {  "ns": "<namespace\_id>",  "id": "<id>",  "vocab\_name": "<vocab\_name>",  "vocab\_url": "http://cv.example.com/example.cv",  "vocab\_description": "<brief description of the vocabulary>",  // acl name values are namespace defined lists, similar to controlled vocabularies  "acl": { "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>", "<aclN>" ]  },  "terms": [  "<term1>",  "<term2>,  "<term3>",  "<termN>"  ]  }  Concrete example:  {  "ns": "hmp",  "id": "71hbRmdih6hf5Yqwdjtl",  "vocab\_name": "node\_types",  "vocab\_url": "http://cv.hmpdacc.org/obos/node\_types.cv",  "vocab\_description": "A controlled vocabulary for HMP node types",  "acl": { "read": [ "all" ], "write": [ "hmpdacc" ] },  "terms": [  "sample",  "subject",  "project",  "study",  "reference\_genome",  "assembly",  "sop"  ]  } | |
| Failed requests (invalid or unrecognized vocabary ID) will yield HTTP 404 ("Not Found"). Other errors will result in HTTP 500.  Security related errors for users attempting to access data without appropriate entries in the namespace ACLs will result in an HTTP 403 "Forbidden" status code. | |

### By name

|  |  |
| --- | --- |
| Retrieve Vocabulary Definition by name | GET /namespaces/${ns}/vocabs/${vocab\_name} |
| Request | |
| Request a description and listing of a controlled vocabulary.  Example request:  $ curl –u <AUTH> –G <OSDF\_URL>/namespaces/hmp/vocabs/node\_types | |
| Response (application/json) | |
| A GET to the URL will yield a JSON object defining the vocabulary for the named list in that namespace.  Example response:  {  "ns": "<namespace\_id>",  "id": "<id>",  "vocab\_name": "<vocab\_name>",  "vocab\_url": "http://cv.example.com/example.cv",  "vocab\_description": "<brief description of the vocabulary>",  // acl name values are namespace defined lists, similar to controlled vocabularies  "acl": { "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>", "<aclN>" ]  },  "terms": [  "<term1>",  "<term2>,  "<term3>",  "<termN>"  ]  }  Concrete example:  {  "ns": "hmp",  "id": "71hbRmdih6hf5Yqwdjtl",  "vocab\_name": "node\_types",  "vocab\_url": "http://cv.hmpdacc.org/obos/node\_types.cv",  "vocab\_description": "A controlled vocabulary for HMP node types",  "acl": { "read": [ "all" ], "write": [ "hmpdacc" ] },  "terms": [  "sample",  "subject",  "project",  "study",  "reference\_genome",  "assembly",  "sop"  ]  } | |
| Failed requests (unrecognized vocabulary name) will yield HTTP 404 ("Not Found"). Other errors will result in HTTP 500.  Security related errors for users attempting to access data without appropriate entries in the namespace ACLs will result in an HTTP 403 "Forbidden" status code. | |

## Create Namespace Vocabulary

|  |  |
| --- | --- |
| Create Vocabulary Definition | POST /namespaces/${ns}/vocabs/${vocab} |
| Request (application/json) | |
| Abstract form:  {  "ns": "<namespace\_id>",  "vocab\_name": "<vocab\_name>",  "vocab\_url": "http://cv.example.com/example.cv",  "vocab\_description": "<brief description of the vocabulary>",  "acl": { "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>", "<aclN>" ]  },  "terms": [  "<term1>",  "<term2>,  "<term3>",  "<termN>"  ]  }  Concrete example:  {  "ns": "hmp",  "vocab\_name": "node\_types",  "vocab\_url": "http://cv.hmpdacc.org/obos/node\_types.cv",  "vocab\_description": "A controlled vocabulary for HMP node types",  "acl": { "read": [ "all" ], "write": [ "hmpdacc" ] },  "terms": [  "sample",  "subject",  "project",  "study",  "reference\_genome",  "assembly",  "sop"  ]  } | |
| Response | |
| Returns HTTP 201 ("Created") on success. The location header is set to the URL for the new vocabulary and its ID can be extracted from the URL. Failed requests, such as the use of an invalid namespace, or the use of a vocabulary name that already exists will yield an HTTP 422 ("Unprocessable Entity"). Other errors will result in HTTP 500 ("Server error") responses.  Security related errors for users attempting to create vocabularies without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

## Edit Namespace Vocabulary

## Delete Namespace Vocabulary

|  |  |
| --- | --- |
| Delete/Destroy Vocabulary | DELETE /namespaces/${ns}/vocabs/${vocab} |
| Request | |
| A DELETE request to the URL containing the vocabulary to delete.  $ curl –u <AUTH> -X DELETE <OSDF\_URL>/namespaces/hmp/vocabs/diseases | |
| Response | |
| Returns HTTP 204 ("No content") on success.  Failed requests (unrecognized vocabulary) will yield HTTP 422 ("Unprocessable Entity") or 500 ("Server error") responses.  Security related errors for users attempting to delete vocabularies without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

6. Node templates  
To assist with the validation of nodes as they are being inserted, and to make use of the concept of the custom controlled vocabularies, OSDF provides for the concept of a node "template". Node templates are meant as a "blueprint" for what nodes should look like in their "meta" sections. The template defines a set of keys and corresponding controlled vocabularies. Any new nodes with a "node\_type" that matches the template must then adhere to the minimum requirements of the template.

{  
  "id": "id",

"node\_type”: "sample",  
  "ver": 3,  
  "acl": {  
       "read": [  "all" ], "write”: [ "hmpdacc" ]  
  },  
  "template": {  
       "key\_1": "cv1",  
       "key\_2": "cv2"  
  }  
}

In the above example we have a template for nodes of type "sample" (listed in the namespace's "node\_types" vocabulary) that requires that a key named "key\_1" be present and that the key is mapped to a "cv1" vocabulary. When the template is created, any new nodes that are inserted with the "sample" node\_type then have that key present in the node's "meta" section. Furthermore, the value of the key must be found in the "cv1" custom controlled vocabulary. New nodes that are inserted may have additional keys, but MUST have the minimum keys and values that are defined by the template.

If a template is deleted or removed, no provision is made for altering nodes that have already been inserted. However, NEW nodes must adhere to the new version of the template.

## List templates

To retrieve a list of the templates for a namespace:

|  |  |
| --- | --- |
| Retrieve a namespace’s templates | GET /namespaces/${ns}/templates |
| Request | |
| A GET to the URL will yield the list of templates.  Example request:  $ curl –u <AUTH> –G <OSDF\_URL>/namespaces/hmp/templates | |
| Response (application/json) | |
| Abstract form:  {  "result\_count": <result\_count>,  "page": <page\_count>,  "results": [  {  "id": "<template\_id1>",  "node\_type": "<node\_type\_cv1>",  "ver": <version1>,  "acl": {  "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>", "<aclN>" ]  },  "template": {  "key\_1": "<controlled\_vocab\_name1>",  "key\_2": "<controlled\_vocab\_name2>"  }  },  {  "id": "<template\_id2>",  "node\_type": "<node\_type\_cv2>",  "ver": <version2>,  "acl": {  "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>", "<aclN>" ]  },  "template": {  "key\_3": "<controlled\_vocab\_name3>",  }  }  ]  }  Concrete example:  {  "result\_count": 1,  "page": 1,  "results": [  {     "id": "e7IsmbdiaVnpx85j4mgj",  "node\_type": "sample",   "ver": 3,   "acl": { "read": [ "all" ], "write": [ "hmpdacc" ] },    "template": {         "body\_site": "body\_site\_cv"     }  }  } | |
| Invalid requests such as those using an unrecognized namespace will yield an HTTP 422 ("Unprocessable Entity"). Other errors will result in an HTTP 500 ("Server error") response.  Security related errors for users attempting to access data without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

If the result is too large to return in a single response, the server will return a partial result set of entries, and an HTTP 206 response (“Partial content”) code. The “X-OSDF-Template-ResultSet” header will then contain a URL to allow the retrieval of the next page of templates. If, in a template set response, the XOSDF-Template-ResultSet header is absent, then there are no further templates available.

## Retrieve template

### By ID

|  |  |
| --- | --- |
| Retrieve template by ID | GET /namespaces/${ns}/templates/ids/${template\_id} |
| Request | |
| A GET to the URL will yield a document describing the template.  Example request:  $ curl –u <AUTH> –G \  <OSDF\_URL>/namespace/hmp/templates/ids/e7IsmbdiaVnpx85j4mgj | |
| Response (application/json) | |
| Abstract form:  {  "id": "<template\_id1>",  "node\_type": "<node\_type\_cv1>",  "ver": <version1>,  "acl": {  "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>", "<aclN>" ]  },  "template": {  "key\_1": "<controlled\_vocab\_name1>",  "key\_2": "<controlled\_vocab\_name2>"  }  }  Concrete example:  {    "id": "e7IsmbdiaVnpx85j4mgj",  "node\_type": "sample",   "ver": 3,   "acl": { "read": [ "all" ], "write": [ "hmpdacc" ] },   "template": {     "body\_site": "body\_site\_cv"    }  } | |
| Failed requests (unrecognized template ID) will yield an HTTP 404 (“Not found"). Other errors will result in an HTTP 500 (“Server error”) response.  Security related errors for users attempting to create nodes without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

### By name

|  |  |
| --- | --- |
| Retrieve template by name | GET /namespaces/${ns}/templates/${template\_name} |
| Request | |
| A GET to the URL will yield a document describing the template.  Example request:  $ curl –u <AUTH> –G <OSDF\_URL>/namespace/hmp/templates/sample | |
| Response (application/json) | |
| Abstract form:  {  "id": "<template\_id1>",  "node\_type": "<node\_type\_cv1>",  "ver": <version1>,  "acl": {  "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>", "<aclN>" ]  },  "template": {  "key\_1": "<controlled\_vocab\_name1>",  "key\_2": "<controlled\_vocab\_name2>"  }  }  Concrete example:  {    "id": "e7IsmbdiaVnpx85j4mgj",  "node\_type": "sample",   "ver": 3,   "acl": { "read": [ "all" ], "write": [ "hmpdacc" ] },   "template": {     "body\_site": "body\_site\_cv"    }  } | |
| Failed requests (unrecognized name) will yield an HTTP 404 (“Not found"). Other errors will result in an HTTP 500 (“Server error”) response.  Security related errors for users attempting to create nodes without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

## Create template

|  |  |
| --- | --- |
| Create Template | POST /namespaces/${ns}/templates |
| Request (application/json) | |
| Abstract form:  {  "node\_type": "<node\_type\_cv1>",  "acl": {  "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>", "<aclN>" ]  },  "template": {  "key\_1": "<controlled\_vocab\_name1>",  "key\_2": "<controlled\_vocab\_name2>"  }  }  Concrete example:  {  "node\_type": "sample",  "acl": { "read" : [ "all" ], "write": [ "hmpdacc" ] },  "template": {  "body\_site": "body\_site\_cv" **// this is the name of a controlled vocabulary**  }  } | |
| Response | |
| Returns HTTP 201 ("Created") on success. The location header is set to the URL for the new template and the template's ID can be extracted from the URL. Failed requests, such as the use of an invalid vocabulary or ACL value, will yield HTTP 422 ("Unprocessable Entity"). Other errors will result in HTTP 500 ("Server error") responses.  Security related errors for users attempting to create templates without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

## Edit Template

## Delete template

|  |  |
| --- | --- |
| Delete/Destroy Template | DELETE /namespaces/${ns}/templates/${template\_id} |
| Request | |
| A DELETE request to the URL containing the template to delete.  $ curl –u <AUTH> -X DELETE \  <OSDF\_URL>/namespaces/hmp/templates/e7IsmbdiaVnpx85j4mgj | |
| Response | |
| Returns HTTP 204 ("No content") on success.  Invalid deletions (unrecognized template ID) will yield HTTP 422 ("Unprocessable Entity") or 500 ("Server error") responses.  Security related errors for users attempting to delete vocabularies without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

# 7. Node management

An OSDF node is a generic container, the only specific mandatory attributes of which are a namespace defining the general project, a unique ID, the linkage or relations to other nodes, and the node type, ACLs restricting for access control, and a generic “meta” key for arbitrary data. The intent of the “meta” field is to hold the namespace specific node content (controlled by the namespace and in conjunction with the use of controlled vocabularies). In the “linkage” field, each node describes how it is connected to other nodes. Since there may be multiple connection types (through the “linkage” controlled vocabulary), there can be multiple links per linkage type. Each linkage type has node members listed by node id. Attempts to create a node with a linkage term that is not listed in the namespace's "linkage" controlled vocabulary, or to create a node with an invalid "node\_type", will result in error.

## Create a Node

|  |  |
| --- | --- |
| Create Node | POST /nodes |
| Request (application/json) | |
| Abstract form:  {  "ns": "<namespace\_id>",  "linkage": { "<ns\_linkage\_cv1>": ["<osdf\_id\_1>", "<osdf\_id\_2>", "<osdf\_id\_N>" ],  "<ns\_linkage\_cv2>": ["<osdf\_id\_1>", "<osdf\_id\_2>", "<osdf\_id\_N>" ],  "<ns\_linkage\_cvN>": ["<osdf\_id\_1>", "<osdf\_id\_2>", "<osdf\_id\_N>" ]  },  // acl name values are namespace defined lists, similar to controlled vocabularies  "acl": { "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>", "<aclN>" ]  },  "node\_type": "<node\_type\_cv1>",  "meta": {  **<arbitrary\_json\_defined\_by\_namespace>**  }  }  Concrete example:  {  "ns": "hmp",  "linkage": { "derives\_from": [ "23425234", "2342562346356" ],  "collected\_from" : [ "23435e57", "309808234507" ]  },  "acl": { "read" : [ "hmpdacc" ], "write": [ "hmpdacc" ] },  "node\_type": "reference\_genome",  "meta": {  "name": "New sample name",  "alt\_name": “New alternate name",  "description": “New description",  "tags": [ "female", "oral" ]  }  } | |
| Response | |
| Returns HTTP 201 ("Created") on success. The location header is set to the URL for the new node and the node’s ID can be extracted from the URL. Failed requests, such as the use of an invalid linkage term, an invalid "node\_type", or ACL value, will yield HTTP 422 ("Unprocessable Entity"). Other errors will result in HTTP 500 ("Server error") responses.  Security related errors for users attempting to create nodes without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

## Retrieve a Node

|  |  |
| --- | --- |
| Retrieve Node | GET /nodes/${node\_id} |
| Request | |
| A GET to the URL will yield a document describing the node.  Example request:  $ curl –u <AUTH> –G <OSDF\_URL>/nodes/bhpmc7hbiyr1jh1vYJ8w | |
| Response (application/json) | |
| Abstract form:  {  "ns": "<namespace\_id>",  "id": "<id>",  "ver": 4,  "linkage": { "<ns\_linkage\_cv1>": [ "<osdf\_id\_1>", "<osdf\_id\_2>", "<osdf\_id\_N>" ],  "<ns\_linkage\_cv2>": [ "<osdf\_id\_1>", "<osdf\_id\_2>", "<osdf\_id\_N>" ],  "<ns\_linkage\_cvN>": [ "<osdf\_id\_1>", "<osdf\_id\_2>", "<osdf\_id\_N>" ]  },  "acl": { "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>", "<aclN>" ]  },  "node\_type": "<node\_type\_cv1>",  "meta": {  **<arbitrary\_json\_defined\_by\_the\_namespace>**  }  }  Concrete example:  {  "ns": "hmp",  "id": "dc0e2473ac6cf2f2739104f10ffef1ef",  "ver": "1-19c7bd9e8f606c65216a832e27629877",  "linkage": {  "derived\_from": [ "dc0e2473ac6cf2f2739104f10fea5632" ],  "part\_of": [ "10b955c73ee92a34fa55631571ad70ca" ]  },  "acl": { "read": [ "all" ], "write": [ "hmpdacc" ] },  "node\_type": "16S\_DNA\_prep",  "meta": {  "lib\_screen": "",  "biotic\_relationship": "",  "source\_mat\_id": [],  "specific\_host": "",  "assembly": "",  "rand\_subject\_id": "159005010",  "hmp\_body\_site": "throat",  "storage\_duration": "",  "trophic\_level": "",  "repository\_id": "SRS011515",  "storage\_temperature": "",  "host\_spec\_range": "",  "num\_replicons": "",  "body\_product": "",  "health\_disease\_stat": "",  "annot\_source": "",  "estimated\_size": "",  "fma\_body\_site": "Throat [FMA:228738]",  "pathogenicity": "",  "hmp\_supersite": "oral",  "extrachrom\_elements": "",  "storage\_location": "",  "encoded\_traits": "",  "subspecf\_gen\_lin": "",  "ncbi\_taxon\_id": "646099",  "finishing\_strategy": "",  "mimarks": {  "lib\_vector": "",  "geo\_loc\_name": "United States of America",  "nucl\_acid\_amp": "",  "collection\_date": "Unknown",  "seq\_meth": "pyrosequencing",  "lib\_size": "",  "feature": "human-associated habitat [ENVO:00009003]",  "mid": "",  "pcr\_primers": "",  "sop": [],  "biome": "terrestrial biome [ENVO:00000446]",  "url": [],  "rel\_to\_oxygen": "",  "project\_name": "Human Microbiome Project 16S rRNA Clinical Production Pilot",  "samp\_collect\_device": "Catch-All sample collection swab",  "samp\_size": "",  "samp\_mat\_process": "",  "isol\_growth\_condt": "",  "target\_subfragment": "",  "experimental\_factor": "",  "lib\_reads\_seqd": "",  "target\_gene": "16S rRNA",  "env\_package": "human-associated",  "submitted\_to\_insdc": "Yes",  "lat\_lon": "Unknown",  "lib\_const\_meth": "",  "adapters": "",  "investigation\_type": "mimarks-survey",  "material": "biological product [ENVO:02000043]",  "nucl\_acid\_ext": "",  "pcr\_cond": ""  },  "sex": "female",  "assembly\_name": "",  "tags": [],  "visit\_number": "1",  "comment": "",  "nap\_id": "700016369"  }  } | |
| Failed requests (unrecognized id) will yield HTTP 422 ("Unprocessable Entity") or 500 ("Server error") responses.  Security related errors for users attempting to create nodes without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

## Retrieve a Previous Node Version

All Nodes are versioned so that a user can always retrieve a previous version of a node. This ensures that the combination of node ID is immutable and can be easily retrieved at any point in time unless the node itself is deleted.

|  |  |
| --- | --- |
| Retrieve Node Version | GET /nodes/${node\_id}/ver/${version\_number} |
| Request | |
| A GET to the URL will yield a document describing the node.  Example request:  $ curl –u <AUTH> –G <OSDF\_URL>/nodes/dc0e2473ac6cf2f2739104f10ff57561/ver/2 | |
| Response (application/json) | |
| Abstract form:  {  "ns": "<namespace\_id>",  "id": "<id>",  "ver": <version\_number>,  "linkage": {  "<ns\_linkage\_cv1>": [ "<osdf\_id\_1>", "<osdf\_id\_2>", "<osdf\_id\_N>" ],  "<ns\_linkage\_cv2>": [ "<osdf\_id\_1>", "<osdf\_id\_2>", "<osdf\_id\_N>" ],  "<ns\_linkage\_cvN>": [ "<osdf\_id\_1>", "<osdf\_id\_2>", "<osdf\_id\_N>" ]  },  // acls are namespace defined (controlled vocabulary)  "acl": { "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2>, "<aclN>" ]  },  "node\_type": "<node\_type\_cv1>",  "meta": {  **<arbitrary\_json\_defined\_by\_the\_namespace>**  }  }  Concrete example:  {  "ns": "hmp",  "id": "dc0e2473ac6cf2f2739104f10ff57561",  "ver": "2-69caaa097cc4877f8538d2cd541ed60f",  "linkage": {  "part\_of": [ "10b955c73ee92a34fa55631571ad70ca" ],  "collected\_from": [ "10b955c73ee92a34fa55631571c0b398" ]  },  "node\_type": "sample",  "acl": { "read": [ "all" ], "write": [ "hmpdacc" ] },  "meta": {  "rand\_subject\_id": "159713063",  "hmp\_body\_site": "throat",  "mimarks": {  "geo\_loc\_name": "United States of America",  "collection\_date": "Unknown",  "feature": "human-associated habitat [ENVO:00009003]",  "biome": "terrestrial biome [ENVO:00000446]",  "rel\_to\_oxygen": "",  "samp\_collect\_device": "Catch-All sample collection swab",  "samp\_size": "",  "samp\_mat\_process": "",  "env\_package": "human-associated",  "lat\_lon": "Unknown",  "investigation\_type": "mimarks-survey",  "material": "biological product [ENVO:02000043]"  },  "visit\_number": "2"  }  } | |
| Failed requests (unrecognized id) will yield HTTP 404 ("Not found"). Other errors will result in an HTTP 500 ("Server error") response.  Security related errors for users attempting to create nodes without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

## Retrieve Node Linkage

Using the "linkage" key, nodes may list other nodes that they connect to. The precise name and nature of the connection is left to the namespace to determine with the "linkage" controlled vocabulary ( See "" ). Retrieval of connected nodes is extremely important for users and software to be able to easily traverse the node graph, so OSDF provides a specialized API method to obtain a node's neighbors. In the result set, nodes are only listed once at most, regardless of how many times the source node links to it.

|  |  |
| --- | --- |
| Retrieve node linkage | GET /nodes/${node\_id}/linkage |
| Request | |
| A GET to the URL will yield a document describing the nodes that are connected to the node of interest.  Example request:  $ curl –u <AUTH> –G <OSDF\_URL>/nodes/bhpmc7hbiyr1jh1vYJ8w/linkage | |
| Response (application/json) | |
| Concrete example:  {  "result\_count": 2,  "page": 1,  "results": [  {  "ns": "hmp",  "id": "ado8ffhasidfuypa8df",  "ver": 2,  "linkage": {  "derives\_from": [ "u3nfe53n2kl47naTHfmv", "5jspw7jq73jd2iw2daJH" ],  "collected\_from" : ["z1v98q390AmfiogyymQp", "0Zfntlgncpu9Fnh35r81" ]  },  "acl": { "read" : [ "all" ], "write": [ "hmpdacc" ] },  "node\_type": "reference\_genome",  "meta": {  "name": "reference genome name",  "alt\_name": "alternate name",  "description": "description",  "tags": [ "female", "oral" ]  }  },  {  "ns": "hmp",  "id": "c855frGy1o8rxhadl2mY",  "ver": 4,  "linkage": {  "derives\_from": [ "u3nfe53n2kl47naTHfmv", "5jspw7jq73jd2iw2daJH" ],  "collected\_from": [ "z1v98q390AmfiogyymQp", "0Zfntlgncpu9Fnh35r81" ]  },  "acl": { "read": [ "all" ], "write": [ "hmpdacc" ] },  "node\_type": "reference\_genome",  "meta": {  "name": "another reference genome",  "alt\_name": "another alternate name",  "description": "Another description",  "tags": [ "female", "oral" ]  }  }  ]  } | |
| Failed requests (unrecognized id) will yield HTTP 404 ("Not found") or 500 ("Server error") responses.  Security related errors for users attempting to create nodes without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

## Edit a Node

Users may edit/update nodes with new data by using the HTTP PUT method. However, because of the nature of REST and HTTP, multiple simultaneous requests to update a node are possible. Therefore, for consistency, and to ensure that the correct version of a node is being operated on, requests to update a node must include the node version. If a request for a node is received for an old version, that request will fail.

|  |  |
| --- | --- |
| Edit/Update node | PUT /nodes/${node\_id} |
| Request (application/json) | |
| A PUT to the JSON URL with a JSON structure describing the new node data.  Abstract form:    "ns": "<namespace\_id>",  "ver": <version>  "linkage": { "<ns\_linkage\_cv1>": [ "<osdf\_id\_1>", "<osdf\_id\_2>", "<osdf\_id\_N>" ],  "<ns\_linkage\_cv2>": [ "<osdf\_id\_1>", "<osdf\_id\_2>", "<osdf\_id\_N>" ],  "<ns\_linkage\_cvN>": [ "<osdf\_id\_1>", "<osdf\_id\_2>", "<osdf\_id\_N>" ],  // acl values are namespace defined  "acl": { "read": [ "<acl1>", "<acl2>", "<aclN>" ],  "write": [ "<acl1>", "<acl2", "<aclN>" ]  },  "node\_type": "<node\_type\_cv>",  "meta": {  **<arbitrary\_json\_defined\_by\_the\_namespace>**  }  }  Concrete example:  {  "ns": "hmp",  "ver": 5,  "linkage": { "derives\_from": [ "2cJ4qdjufs9ifAc1ei14", "q8w4hCEbfj7d435ymcny" ],  "collected\_from" : [ "j23zrcp82zJKb2dfwv5p", "up644ioiF8alz03wmvhF" ]  },  "acl": { "read" : [ "all" ], "write": [ "hmpdacc" ] },  "node\_type": "reference\_genome",  "meta": {  "name": "New sample name",  "alt\_name": “New alternate name",  "description": "New description",  "tags": [ "female", "oral" ]  }  } | |
| Response (application/json) | |
| Returns HTTP status code 200 on success.  Failed requests, such as those using invalid an invalid node type, linkage or acl value, will yield HTTP 422 ("Unprocessable Entity") response. Other errors will result in HTTP 500 ("Server error") responses.  Security related errors for users attempting to edit/alter nodes without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

## Delete a Node

|  |  |
| --- | --- |
| Delete/Destroy node | DELETE /nodes /${node\_id} |
| Request | |
| A DELETE request to the URL containing the node id to delete.  $ curl –u <AUTH> -X DELETE <OSDF\_URL>/nodes/ as6hV2w2myskiapm5Hjy | |
| Response | |
| Returns HTTP 204 ("No content") on success.  Failed requests (unrecognized id) will yield HTTP 422 ("Unprocessable Entity") or 500 ("Server error") responses.  **Special note:** To prevent linkage errors when deleting nodes, users must first delete the node’s linkages through node edits. This is to avoid the problem of orphaned nodes and broken links.  Security related errors for users attempting to delete nodes without appropriate entries in the namespaces ACLs will result in an HTTP 403 "Forbidden" status code. | |

# 8. Querying

## Making Queries

Queries for nodes may be performed by posting an “elasticsearch DSL” JSON query to the nodes/query/${namespace} URI. The namespace is required and each search request is limited to a single namespace. “Elasticsearch” is an open source indexing and querying engine that comes with its own [DSL](http://www.elasticsearch.org/guide/reference/query-dsl/). This DSL provides a robust mechanism for formulating complicated queries in which terms can be logically combined, filtered; marked as must include, should include, or must not include; as well as many other search options.

Note: results are filtered by namespace read ACLs prior to returning results.

|  |  |
| --- | --- |
| Query nodes | POST /nodes/query/${namespace} |
| Request | |
| An HTTP POST to the URL with a properly formed query will return the search results within the given namespace.  Example request:  $ curl -u <AUTH> -d '{"query" : { "term" : { "node\_type" : "annotation" }}}' <OSDF\_URL>/nodes/query/<ns>  Or, if the JSON query is stored in a file:  $ curl -u <AUTH> -d @<filename> <OSDF\_URL>/nodes/query/<ns> | |
| Response (application/json) | |
| Returns HTTP status code 200 on success along with a JSON document  Concrete example:  {  "total": 1,  "max\_score": 2.9668937,  "hits":[  {  "\_index": "osdf",  "\_type": "osdf",  "\_id": "9a1696ea79327fc87db6942a43bb266a",  "\_score": 2.9668937,  "\_source": {  "\_rev": "3-70fff2af1604e4981f6e8c511e58e416",  "ns": "test",  "\_id": "9a1696ea79327fc87db6942a43bb266a",  "acl": {  "write": ["testgroup1"],  "read": ["testgroup1"]  },  "linkage": {  "computed\_from": ["9a1696ea79327fc87db6942a438d2531"]  },  "node\_type": "annotation",  "meta": {  "name": "New sample name",  "alt\_name": “New alternate name",  "description": "New description"  }  }  }  ]  }  Failed query requests will yield HTTP 422 ("Unprocessable Entity") if invalid JSON is provided or 500 ("Server error") if other errors have occurred. | |

## Elasticsearch DSL examples

Presented here, for convenience, are a few examples of common elasticsearch queries. The full DSL can be found [here](http://www.elasticsearch.org/guide/reference/query-dsl/).

One thing to note, is that though the DSL does not explicitly state it, **all** query types and filters **must** be contained within a JSON query element: { "query" :

|  |  |
| --- | --- |
| 1. Simple term query |  |
|  | |
| Will return all documents containing the top-level term “node\_type” with the value “annotation”.  Note: Be careful with the “term” query as it is “not analyzed” according to the [documentation](http://www.elasticsearch.org/guide/reference/query-dsl/term-query.html). The underlying Lucene analyzer both tokenizes the text and makes it lowercase. As a result, searching for the literal “Annotation” with this query type will not yield results.  {  "query": {  "term" : { "node\_type" : "annotation"}  }  } | |

|  |  |
| --- | --- |
| 1. Simple value query |  |
|  | |
| Will return all documents containing the literal “annotation” on any node in the document’s JSON structure.  {  "query": {  "query\_string" : { "query" : "annotation" }  }  } | |

|  |
| --- |
| 1. Query on specific hierarchy of nested JSON fields to a specific value |
|  |
| Nested JSON fields can be specified using the dot operator. This query will only return documents who’s hierarchical JSON structure matches the literal provided.  {  "query" : {  "query\_string" : {  "fields" : ["meta.process.software"],  "query" : "blast"  }  }  } |

|  |  |
| --- | --- |
| 1. Multiple literals AND’d and OR’d together |  |
|  | |
| Will return all documents containing "this" AND "that" OR "thus". The default operator, if none specified, is the OR operator. This will automatically be inserted between literals if no operator is specified or the default operator is not set. The default\_field is the JSON field on which to search, but is not required and defaults to "\_all".  {  "query": {  "query\_string" : {  "default\_field" : "node\_type",  "query" : "this AND that OR thus"  }  }  } | |

|  |  |
| --- | --- |
| 1. Boolean query with nested subqueries |  |
|  | |
| Each of the bool query fields (must, should, and must\_not) takes an array of other DSL queries. These sub-queries are not prefixed with the token “query” as is the outermost query.    {  "query" : {  "bool" : {  "must" : [],  "should" : [  {  "query\_string" : { "query" : "8a24bf1e62c5d26975ceed08ce1cc155" }  }  ],  "must\_not" : []  }  }  } | |

|  |  |
| --- | --- |
| 1. Filters |  |
|  | |
| In addition to queries, filters may be used to further limit the query results. For example, the following bool query has been filtered to only return documents that contain a “node\_type” of “annotation”. Filters have both a query field and a filter field and they return query results as do queries. As such, a nested filter query may be placed inside the query portion of an outer filter.    {  "query" : {  "filtered" : {  "query" : {  "bool" : {  "must" : [],  "should" : [  {  "query\_string" : {  "query" : "8a24bf1e62c5d26975ceed08ce1cc155"  }  }  ],  "must\_not" : []  },  "filter" : {  "and" : [ { "term" : { "node\_type" : "annotation" } } ]  }  }  }  }  } | |

## Sorting Results

Results may be sorted by specifying the “elasticsearch” sort field along with the supplied query. Here is a typical sorting example, though several other types of sorts are allowed. For the complete reference please visit the “elasticsearch” [sorting page](http://www.elasticsearch.org/guide/reference/api/search/sort.html).

Note: The OSDF JSON parser does not need to support ordering. As a result, the “elasticsearch” sort field must be contained within an array so that elements retain order.

|  |  |
| --- | --- |
| 1. Sort |  |
|  | |
| Here is a simple sorting example.  {  "sort" : [  { "last\_name" : {"order" : "asc"} },  { "first\_name" : {"order" : "desc"} },  "age"  ],  "query" : {  "term" : { "state" : "MD" }  }  } | |

## Range / Paginated Results

Query results may be paginated using the “elasticsearch” from and size keywords.

Note: When limiting the number of results in this way, the results

|  |  |
| --- | --- |
| 1. Pagination |  |
|  | |
| Here is a simple pagination example. This will start from index 40 of the search results and return the next 10. It’s should be considered to use pagination along with a sort field though it’s not required.  {  "from" : 40,  "size" : 10,  "sort" : ["\_id"],  "query" : {  "term" : { "node\_type" : "annotation" }  }  } | |