plone.restapi Documentation

Release 1.0a1

Plone Foundation

Jun 28, 2017

Contents

Warning: plone.restapi ist still in a alpha stage. Any functionality described by this document might not be currently available on Plone, and/or might change significantly in the future.

CHAPTER 1

Contents

Introduction

API Browser Quick Guide

It can make your life easier if you use some kind of API browser application to explore the API when diving into this documentation.

- We recommend to use the free Postman browser plugin.
- For easy onboarding take a look at **our** exploring-api-postman-onboarding **Quick-Guide**.

A hypermedia API provides an entry point to the API, which contains hyperlinks the clients can follow. Just like a human user of a regular website, who knows the initial URL of a website and then follows hyperlinks to navigate through the site. This has the advantage that the client only needs to understand how to detect and follow links. The URLs (apart from the inital entry point) and other details of the API can change without breaking the client.

The entry point to the Plone RESTful API is the portal root. The client can ask for a *REST* API response by setting the `'Accept'` HTTP header to `'application/json'`: http

```
GET /plone HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone -H "Accept: application/json" --user admin:secret
```

httpie

```
http -j http://nohost/plone -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone', headers={'Accept': 'application/json'}, auth=(
↪'admin', 'secret'))
```

This uses so-called 'content negotiation'

1.1.1 Content Negotiation

Content negotiation is a mechanism defined in the HTTP specification that makes it possible to serve different versions of a document (or more generally, a resource representation) at the same URI, so that user agents can specify which version fit their capabilities the best.

The user agent (or the REST consumer) can ask for a specific representation by providing an Accept HTTP header that lists acceptable media types (e.g. JSON):

```
GET /
Accept: application/json
```

The server is then able to supply the version of the resource that best fits the user agent's needs. This is reflected in the Content-Type header:

```
HTTP 200 OK
Content-Type: application/json

{
  'data': ...
}
```

The server will then respond with the portal root in the JSON format:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone",
  "@type": "Plone Site",
  "id": "plone",
  "items": [
    {
      "@id": "http://localhost:55001/plone/front-page",
      "@type": "Document",
      "description": "Congratulations! You have successfully installed Plone.",
      "review_state": "private",
      "title": "Welcome to Plone"
    }
  ],
  "items_total": 1,
  "parent": {},
  "sharing": {
    "@id": "http://localhost:55001/plone/@sharing",
    "title": "Sharing"
  }
}
```

`@id` is a unique identifier for resources (IRIs). The `@id` property can be used to navigate through the web API by following the links.

`@type` sets the data type of a node or typed value

`items` is a list that contains all objects within that resource.

A client application can "follow" the links (by calling the @id property) to other resources. This allows to build a losely coupled client that does not break if some of the URLs change, only the entry point of the entire API (in our case the portal root) needs to be known in advance.

Another example, this time showing a request and response for a document. Click on the buttons below to show the different syntaxes for the request. http

```
GET /plone/front-page HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/front-page -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/front-page -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/front-page', headers={'Accept': 'application/json'},
↪ auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/front-page",
  "@type": "Document",
  "UID": "SomeUUID00000000000000000000001",
  "allow_discussion": null,
  "changeNote": "",
  "contributors": [],
  "created": "2016-01-21T01:14:48+00:00",
  "creators": [
    "test_user_1_",
    "admin"
  ],
  "description": "Congratulations! You have successfully installed Plone.",
  "effective": null,
  "exclude_from_nav": false,
  "expires": null,
  "id": "front-page",
  "language": "",
  "layout": "document_view",
  "modified": "2016-01-21T01:24:11+00:00",
  "parent": {
    "@id": "http://localhost:55001/plone",
    "@type": "Plone Site",
    "description": "",
    "title": "Plone site"
  },
  "relatedItems": [],
```

```
  "review_state": "private",
  "rights": "",
  "sharing": {
    "@id": "http://localhost:55001/plone/front-page/@sharing",
    "title": "Sharing"
  },
  "subjects": [],
  "table_of_contents": null,
  "text": {
    "content-type": "text/plain",
    "data": "<p>If you're seeing this instead of the web site you were expecting, the␣
↪owner of this web site has just installed Plone. Do not contact the Plone Team or␣
↪the Plone mailing lists about this.</p>",
    "encoding": "utf-8"
  },
  "title": "Welcome to Plone",
  "version": "current"
}
```


And so on, see

1.1.2 Plone Content

How to get all standard Plone content representations. The syntax is given in various tools, click on 'curl', 'http-request' or 'python-requests' to see examples.

Note: For folderish types, collections or search results, the results will be **batched** if the size of the resultset exceeds the batch size. See *Batching* for more details on how to work with batched results.

Plone Portal Root:

http

```
GET /plone HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone -H "Accept: application/json" --user admin:secret
```

httpie

```
http -j http://nohost/plone -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone', headers={'Accept': 'application/json'},␣
↪auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "@id": "http://localhost:55001/plone",
  "@type": "Plone Site",
  "id": "plone",
  "items": [
    {
      "@id": "http://localhost:55001/plone/front-page",
      "@type": "Document",
      "description": "Congratulations! You have successfully installed Plone.
↪",
      "review_state": "private",
      "title": "Welcome to Plone"
    }
  ],
  "items_total": 1,
  "parent": {},
  "sharing": {
    "@id": "http://localhost:55001/plone/@sharing",
    "title": "Sharing"
  }
}
```


Plone Folder:

http

```
GET /plone/folder HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/folder -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/folder -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/folder', headers={'Accept': 'application/
↪json'}, auth=('admin', 'secret'))
```


```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/folder",
  "@type": "Folder",
  "UID": "SomeUUID00000000000000000000002",
  "allow_discussion": null,
  "contributors": [],
  "created": "2016-01-21T07:14:48+00:00",
  "creators": [
    "test_user_1_",
```

```
    "admin"
  ],
  "description": "This is a folder with two documents",
  "effective": null,
  "exclude_from_nav": false,
  "expires": null,
  "id": "folder",
  "items": [
    {
      "@id": "http://localhost:55001/plone/folder/doc1",
      "@type": "Document",
      "description": "",
      "review_state": "private",
      "title": "A document within a folder"
    },
    {
      "@id": "http://localhost:55001/plone/folder/doc2",
      "@type": "Document",
      "description": "",
      "review_state": "private",
      "title": "A document within a folder"
    }
  ],
  "items_total": 2,
  "language": "",
  "layout": "listing_view",
  "modified": "2016-01-21T07:24:11+00:00",
  "nextPreviousEnabled": false,
  "parent": {
    "@id": "http://localhost:55001/plone",
    "@type": "Plone Site",
    "description": "",
    "title": "Plone site"
  },
  "relatedItems": [],
  "review_state": "private",
  "rights": "",
  "sharing": {
    "@id": "http://localhost:55001/plone/folder/@sharing",
    "title": "Sharing"
  },
  "subjects": [],
  "title": "My Folder",
  "version": "current"
}
```


Plone Document:

http

```
GET /plone/front-page HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/front-page -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/front-page -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/front-page', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/front-page",
  "@type": "Document",
  "UID": "SomeUUID00000000000000000000000001",
  "allow_discussion": null,
  "changeNote": "",
  "contributors": [],
  "created": "2016-01-21T01:14:48+00:00",
  "creators": [
    "test_user_1_",
    "admin"
  ],
  "description": "Congratulations! You have successfully installed Plone.",
  "effective": null,
  "exclude_from_nav": false,
  "expires": null,
  "id": "front-page",
  "language": "",
  "layout": "document_view",
  "modified": "2016-01-21T01:24:11+00:00",
  "parent": {
    "@id": "http://localhost:55001/plone",
    "@type": "Plone Site",
    "description": "",
    "title": "Plone site"
  },
  "relatedItems": [],
  "review_state": "private",
  "rights": "",
  "sharing": {
    "@id": "http://localhost:55001/plone/front-page/@sharing",
    "title": "Sharing"
  },
  "subjects": [],
  "table_of_contents": null,
  "text": {
    "content-type": "text/plain",
    "data": "<p>If you're seeing this instead of the web site you were␣
↪expecting, the owner of this web site has just installed Plone. Do not␣
↪contact the Plone Team or the Plone mailing lists about this.</p>",
    "encoding": "utf-8"
  },
  "title": "Welcome to Plone",
```

```
  "version": "current"
}
```

News Item:

http

```
GET /plone/newsitem HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/newsitem -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/newsitem -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/newsitem', headers={'Accept': 'application/
↪json'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/newsitem",
  "@type": "News Item",
  "UID": "SomeUUID00000000000000000000002",
  "allow_discussion": null,
  "changeNote": "",
  "contributors": [],
  "created": "2016-01-21T02:14:48+00:00",
  "creators": [
    "test_user_1_",
    "admin"
  ],
  "description": "This is a news item",
  "effective": null,
  "exclude_from_nav": false,
  "expires": null,
  "id": "newsitem",
  "image": {
    "content-type": "image/png",
    "download": "http://localhost:55001/plone/newsitem/@@images/image",
    "filename": "image.png",
    "height": 56,
    "scales": {
      "icon": {
        "download": "http://localhost:55001/plone/newsitem/@@images/image/
↪icon",
        "height": 8,
        "width": 32
```

```
      },
      "large": {
        "download": "http://localhost:55001/plone/newsitem/@@images/image/
↪large",
        "height": 56,
        "width": 215
      },
      "listing": {
        "download": "http://localhost:55001/plone/newsitem/@@images/image/
↪listing",
        "height": 4,
        "width": 16
      },
      "mini": {
        "download": "http://localhost:55001/plone/newsitem/@@images/image/
↪mini",
        "height": 52,
        "width": 200
      },
      "preview": {
        "download": "http://localhost:55001/plone/newsitem/@@images/image/
↪preview",
        "height": 56,
        "width": 215
      },
      "thumb": {
        "download": "http://localhost:55001/plone/newsitem/@@images/image/
↪thumb",
        "height": 33,
        "width": 128
      },
      "tile": {
        "download": "http://localhost:55001/plone/newsitem/@@images/image/
↪tile",
        "height": 16,
        "width": 64
      }
    },
    "size": 1185,
    "width": 215
  },
  "image_caption": "This is an image caption.",
  "language": "",
  "layout": "newsitem_view",
  "modified": "2016-01-21T02:24:11+00:00",
  "parent": {
    "@id": "http://localhost:55001/plone",
    "@type": "Plone Site",
    "description": "",
    "title": "Plone site"
  },
  "relatedItems": [],
  "review_state": "private",
  "rights": "",
  "sharing": {
    "@id": "http://localhost:55001/plone/newsitem/@sharing",
    "title": "Sharing"
  },
```

```
  "subjects": [],
  "text": {
    "content-type": "text/plain",
    "data": "<p>Lorem ipsum</p>",
    "encoding": "utf-8"
  },
  "title": "My News Item",
  "version": "current"
}
```


Event:

http

```
GET /plone/event HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/event -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/event -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/event', headers={'Accept': 'application/
↪json'}, auth=('admin', 'secret'))
```


```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/event",
  "@type": "Event",
  "UID": "SomeUUID00000000000000000000002",
  "allow_discussion": null,
  "attendees": [],
  "changeNote": "",
  "contact_email": null,
  "contact_name": null,
  "contact_phone": null,
  "contributors": [],
  "created": "2016-01-21T03:14:48+00:00",
  "creators": [
    "test_user_1_",
    "admin"
  ],
  "description": "This is an event",
  "effective": null,
  "end": null,
  "event_url": null,
  "exclude_from_nav": false,
```

```
  "expires": null,
  "id": "event",
  "language": "",
  "layout": "event_view",
  "location": null,
  "modified": "2016-01-21T03:24:11+00:00",
  "open_end": null,
  "parent": {
    "@id": "http://localhost:55001/plone",
    "@type": "Plone Site",
    "description": "",
    "title": "Plone site"
  },
  "recurrence": null,
  "relatedItems": [],
  "review_state": "private",
  "rights": "",
  "sharing": {
    "@id": "http://localhost:55001/plone/event/@sharing",
    "title": "Sharing"
  },
  "start": null,
  "subjects": [],
  "sync_uid": null,
  "text": null,
  "timezone": null,
  "title": "Event",
  "version": "current",
  "whole_day": null
}
```


Image:

http

```
GET /plone/image HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/image -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/image -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/image', headers={'Accept': 'application/
↪json'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "@id": "http://localhost:55001/plone/image",
  "@type": "Image",
  "UID": "SomeUUID00000000000000000000002",
  "allow_discussion": null,
  "contributors": [],
  "created": "2016-01-21T06:14:48+00:00",
  "creators": [
    "test_user_1_",
    "admin"
  ],
  "description": "This is an image",
  "effective": null,
  "exclude_from_nav": false,
  "expires": null,
  "id": "image",
  "image": {
    "content-type": "image/png",
    "download": "http://localhost:55001/plone/image/@@images/image",
    "filename": "image.png",
    "height": 56,
    "scales": {
      "icon": {
        "download": "http://localhost:55001/plone/image/@@images/image/icon",
        "height": 8,
        "width": 32
      },
      "large": {
        "download": "http://localhost:55001/plone/image/@@images/image/large
↪",
        "height": 56,
        "width": 215
      },
      "listing": {
        "download": "http://localhost:55001/plone/image/@@images/image/
↪listing",
        "height": 4,
        "width": 16
      },
      "mini": {
        "download": "http://localhost:55001/plone/image/@@images/image/mini",
        "height": 52,
        "width": 200
      },
      "preview": {
        "download": "http://localhost:55001/plone/image/@@images/image/
↪preview",
        "height": 56,
        "width": 215
      },
      "thumb": {
        "download": "http://localhost:55001/plone/image/@@images/image/thumb
↪",
        "height": 33,
        "width": 128
      },
      "tile": {
        "download": "http://localhost:55001/plone/image/@@images/image/tile",
```

```
        "height": 16,
        "width": 64
      }
    },
    "size": 1185,
    "width": 215
  },
  "language": "",
  "layout": "image_view",
  "modified": "2016-01-21T06:24:11+00:00",
  "parent": {
    "@id": "http://localhost:55001/plone",
    "@type": "Plone Site",
    "description": "",
    "title": "Plone site"
  },
  "relatedItems": [],
  "review_state": null,
  "rights": "",
  "sharing": {
    "@id": "http://localhost:55001/plone/image/@sharing",
    "title": "Sharing"
  },
  "subjects": [],
  "title": "My Image",
  "version": "current"
}
```

File:

http

```
GET /plone/file HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/file -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/file -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/file', headers={'Accept': 'application/json
↪'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/file",
  "@type": "File",
```

```
  "UID": "SomeUUID00000000000000000000002",
  "allow_discussion": null,
  "contributors": [],
  "created": "2016-01-21T05:14:48+00:00",
  "creators": [
    "test_user_1_",
    "admin"
  ],
  "description": "This is a file",
  "effective": null,
  "exclude_from_nav": false,
  "expires": null,
  "file": {
    "content-type": "application/pdf",
    "download": "http://localhost:55001/plone/file/@@download/file",
    "filename": "file.pdf",
    "size": 74429
  },
  "id": "file",
  "language": "",
  "layout": "file_view",
  "modified": "2016-01-21T05:24:11+00:00",
  "parent": {
    "@id": "http://localhost:55001/plone",
    "@type": "Plone Site",
    "description": "",
    "title": "Plone site"
  },
  "relatedItems": [],
  "review_state": null,
  "rights": "",
  "sharing": {
    "@id": "http://localhost:55001/plone/file/@sharing",
    "title": "Sharing"
  },
  "subjects": [],
  "title": "My File",
  "version": "current"
}
```


Link:

http

```
GET /plone/link HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/link -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/link -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/link', headers={'Accept': 'application/json
↪'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/link",
  "@type": "Link",
  "UID": "SomeUUID00000000000000000000002",
  "allow_discussion": null,
  "changeNote": "",
  "contributors": [],
  "created": "2016-01-21T04:14:48+00:00",
  "creators": [
    "test_user_1_",
    "admin"
  ],
  "description": "This is a link",
  "effective": null,
  "exclude_from_nav": false,
  "expires": null,
  "id": "link",
  "language": "",
  "layout": "link_redirect_view",
  "modified": "2016-01-21T04:24:11+00:00",
  "parent": {
    "@id": "http://localhost:55001/plone",
    "@type": "Plone Site",
    "description": "",
    "title": "Plone site"
  },
  "remoteUrl": "http://",
  "review_state": "private",
  "rights": "",
  "sharing": {
    "@id": "http://localhost:55001/plone/link/@sharing",
    "title": "Sharing"
  },
  "subjects": [],
  "title": "My Link",
  "version": "current"
}
```

Collection:

http

```
GET /plone/collection HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/collection -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/collection -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/collection', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/collection",
  "@type": "Collection",
  "UID": "SomeUUID00000000000000000000000002",
  "allow_discussion": null,
  "contributors": [],
  "created": "2016-01-21T08:14:48+00:00",
  "creators": [
    "test_user_1_",
    "admin"
  ],
  "customViewFields": [],
  "description": "This is a collection with two documents",
  "effective": null,
  "exclude_from_nav": false,
  "expires": null,
  "id": "collection",
  "item_count": 30,
  "items": [
    {
      "@id": "http://localhost:55001/plone/front-page",
      "@type": "Document",
      "description": "Congratulations! You have successfully installed Plone.
↪",
      "review_state": "private",
      "title": "Welcome to Plone"
    },
    {
      "@id": "http://localhost:55001/plone/doc1",
      "@type": "Document",
      "description": "",
      "review_state": "private",
      "title": "Document 1"
    },
    {
      "@id": "http://localhost:55001/plone/doc2",
      "@type": "Document",
      "description": "",
      "review_state": "private",
      "title": "Document 2"
    }
```

```
  ],
  "items_total": 3,
  "language": "",
  "layout": "listing_view",
  "limit": 1000,
  "modified": "2016-01-21T08:24:11+00:00",
  "parent": {
    "@id": "http://localhost:55001/plone",
    "@type": "Plone Site",
    "description": "",
    "title": "Plone site"
  },
  "query": [
    {
      "i": "portal_type",
      "o": "plone.app.querystring.operation.string.is",
      "v": "Document"
    }
  ],
  "relatedItems": [],
  "review_state": "private",
  "rights": "",
  "sharing": {
    "@id": "http://localhost:55001/plone/collection/@sharing",
    "title": "Sharing"
  },
  "sort_on": null,
  "sort_reversed": null,
  "subjects": [],
  "text": null,
  "title": "My Collection",
  "version": "current"
}
```


Authentication

`plone.restapi` uses Plone PAS for Authentication.

That means that any authentication method supported by an installed PAS Plugin should work, assuming it's an authentication method that makes sense to use with an API.

For example, to authenticate using HTTP basic auth, you'd set an `Authorization` header:

```
GET /Plone HTTP/1.1
Authorization: Basic Zm9vYmFyOmZvb2Jhcgo=
Accept: application/json
```

HTTP client libraries usually contain helper functions to produce a proper `Authorization` header for you based on given credentials.

Using the `requests` library, you'd set up a session with basic auth like this:

```
import requests

session = requests.Session()
session.auth = ('username', 'password')
```

```
session.headers.update({'Accept': 'application/json'})

response = session.get(url)
```

Or the same example using `curl`:

```
curl -u username:password -H 'Accept:application/json' $URL
```

JSON Web Tokens (JWT)

`plone.restapi` includes a Plone PAS plugin for authentication with JWT. The plugin is installed automatically when installing the product.

Acquiring a token (@login)

A JWT token can be acquired by posting a user's credentials to the `@login` endpoint. http

```
POST /plone/@login HTTP/1.1
Accept: application/json
Content-Type: application/json

{
    "login": "admin",
    "password": "secret"
}
```

curl

```
curl -i -X POST http://nohost/plone/@login -H "Accept: application/json" -H "Content-
↪Type: application/json" --data-raw '{"login": "admin", "password": "secret"}'
```

httpie

```
http -j POST http://nohost/plone/@login login=admin password=secret
```

python-requests

```
requests.post('http://nohost/plone/@login', headers={'Accept': 'application/json'},␣
↪json={'login': 'admin', 'password': 'secret'})
```

The server responds with a JSON object containing the token.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4"
}
```

Authenticating with a token

The token can now be used in subsequent requests by including it in the `Authorization` header with the `Bearer` scheme: http

```
GET /plone/ HTTP/1.1
Accept: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4
```

curl

```
curl -i http://nohost/plone/ -H "Accept: application/json" -H "Authorization: Bearer␣
↪eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_
↪baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4"
```

httpie

```
http -j http://nohost/plone/ Authorization:"Bearer␣
↪eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_
↪baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4"
```

python-requests

```
requests.get('http://nohost/plone/', headers={'Accept': 'application/json',
↪'Authorization': 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4
↪'})
```

Renewing a token (@login-renew)

By default the token will expire after 12 hours and thus must be renewed before expiration. To renew the token simply post to the `@login-renew` endpoint. http

```
POST /plone/@login-renew HTTP/1.1
Accept: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4
```

curl

```
curl -i -X POST http://nohost/plone/@login-renew -H "Accept: application/json" -H
↪"Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4"
```

httpie

```
http -j POST http://nohost/plone/@login-renew Authorization:"Bearer␣
↪eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_
↪baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4"
```

python-requests

```
requests.post('http://nohost/plone/@login-renew', headers={'Accept': 'application/json
↪', 'Authorization': 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4
↪'})
```

The server returns a JSON object with a new token:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4"
}
```

Invalidating a token (@logout)

The `@logout` endpoint can be used to invalidate tokens. However by default tokens are not persisted on the server and thus can not be invalidated. To enable token invaldiation, activate the `store_tokes` option in the PAS plugin. If you need tokens that are valid indefinitely you should also disable the use of Plone's keyring in the PAS plugin (option `use_keyring`).

The logout request must contain the existing token in the `Authorization` header. http

```
POST /plone/@logout HTTP/1.1
Accept: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4
```

curl

```
curl -i -X POST http://nohost/plone/@logout -H "Accept: application/json" -H
↪"Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4"
```

httpie

```
http -j POST http://nohost/plone/@logout Authorization:"Bearer␣
↪eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_
↪baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4"
```

python-requests

```
requests.post('http://nohost/plone/@logout', headers={'Accept': 'application/json',
↪'Authorization': 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJmdWxsbmFtZSI6IiIsInN1YiI6ImFkbWluIn0.SZDnl_baH5M_StJJrzfbj7o-5My30NmSFbMrhpSX5I4
↪'})
```

If invalidation succeeds, the server responds with an empty 204 reponse:

```
HTTP/1.1 204 No Content
```

Permissions

In order for a user to use the REST API, the `plone.restapi: Use REST API` permission is required.

By default, installing the `plone.restapi:default` profile will assign this permission to the `Anonymous` role, so everybody is allowed to use the REST API by default.

If you wish to control in more detail which roles are allowed to use the REST API, please assign this permission accordingly.

As well as the `plone.restapi: Use REST API` permission some of the common Plone permissions are also required, depending on the particular service. For example, retrieving a resource using GET will require `View`, adding an object using POST will require `Add portal content`, and so on.

In order to modify/override this behavior, if your custom service class inherits from `plone.restapi.services.Service`, just override the method `check_permission` and add your custom checks accordingly.

Content Manipulation

plone.restapi does not only expose content objects via a RESTful API. The API consumer can create, read, update, and delete a content object. Those operations can be mapped to the HTTP verbs POST (Create), GET (Read), PUT (Update) and DELETE (Delete).

Manipulating resources across the network by using HTTP as an application protocol is one of core principles of the REST architectural pattern. This allows us to interact with a specific resource in a standardized way:

Verb	URL	Action
POST	/folder	Creates a new document within the folder
GET	/folder/{document-id}	Request the current state of the document
PATCH	/folder/{document-id}	Update the document details
DELETE	/folder/{document-id}	Remove the document

Creating a Resource with POST

To create a new resource, we send a POST request to the resource container. If we want to create a new document within an existing folder, we send a POST request to that folder: http

```
POST /plone/folder HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "@type": "Document",
    "title": "My Document"
}
```

curl

```
curl -i -X POST http://nohost/plone/folder -H "Accept: application/json" -H "Content-
↪Type: application/json" --data-raw '{"@type": "Document", "title": "My Document"}' -
↪-user admin:secret
```

httpie

```
http -j POST http://nohost/plone/folder \\@type=Document title="My Document" -a␣
↪admin:secret
```

python-requests

```
requests.post('http://nohost/plone/folder', headers={'Accept': 'application/json'},␣
↪json={'@type': 'Document', 'title': 'My Document'}, auth=('admin', 'secret'))
```

By setting the 'Accept' header, we tell the server that we would like to receive the response in the 'application/json' representation format.

The 'Content-Type' header indicates that the body uses the 'application/json' format.

The request body contains the minimal necessary information needed to create a document (the type and the title). You could set other properties, like "description" here as well.

Successful Response (201 Created)

If a resource has been created, the server responds with the *201 Created* status code. The 'Location' header contains the URL of the newly created resource and the resource representation in the payload:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: http://localhost:55001/plone/folder/my-document

{
  "@id": "http://localhost:55001/plone/folder/my-document",
  "@type": "Document",
  "UID": "SomeUUID00000000000000000000000005",
  "allow_discussion": null,
  "changeNote": "",
  "contributors": [],
  "created": "2016-10-21T19:00:00+00:00",
  "creators": [
    "admin"
  ],
  "description": "",
  "effective": null,
  "exclude_from_nav": false,
  "expires": null,
  "id": "my-document",
  "language": "",
  "layout": "document_view",
  "modified": "2016-10-21T19:00:00+00:00",
  "parent": {
    "@id": "http://localhost:55001/plone/folder",
    "@type": "Folder",
    "description": "This is a folder with two documents",
    "review_state": "private",
    "title": "My Folder"
  },
  "relatedItems": [],
  "review_state": "private",
  "rights": "",
  "subjects": [],
  "table_of_contents": null,
  "text": null,
  "title": "My Document",
  "version": "current"
}
```

Unsuccessful Response (400 Bad Request)

If the resource could not be created, for instance because the title was missing in the request, the server responds with *400 Bad Request*:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
  'message': 'Required title field is missing'
}
```

The response body can contain information about why the request failed.

Unsuccessful Response (500 Internal Server Error)

If the server can not properly process a request, it responds with *500 Internal Server Error*:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
  'message': 'Internal Server Error'
}
```

The response body can contain further information such as an error trace or a link to the documentation.

Possible POST Responses

Possible server reponses for a POST request are:

- *201 Created* (Resource has been created successfully)
- *400 Bad Request* (malformed request to the service)
- *500 Internal Server Error* (server fault, can not recover internally)

POST Implementation

A pseudo-code example of the POST implementation on the server:

```
try:
    order = createOrder()
    if order == None:
        # Bad Request
        response.setStatus(400)
    else:
        # Created
        response.setStatus(201)
except:
    # Internal Server Error
    response.setStatus(500)
```

TODO: Link to the real implementation...

Reading a Resource with GET

After a successful POST, we can access the resource by sending a GET request to the resource URL: http

```
GET /plone/folder/my-document HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/folder/my-document -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/folder/my-document -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/folder/my-document', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

Successful Response (200 OK)

If a resource has been retrieved successfully, the server responds with *200 OK*:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/folder/my-document",
  "@type": "Document",
  "UID": "SomeUUID00000000000000000000005",
  "allow_discussion": null,
  "changeNote": "",
  "contributors": [],
  "created": "2016-10-21T19:00:00+00:00",
  "creators": [
    "admin"
  ],
  "description": "",
  "effective": null,
  "exclude_from_nav": false,
  "expires": null,
  "id": "my-document",
  "language": "",
  "layout": "document_view",
  "modified": "2016-10-21T19:00:00+00:00",
  "parent": {
    "@id": "http://localhost:55001/plone/folder",
    "@type": "Folder",
    "description": "This is a folder with two documents",
    "review_state": "private",
    "title": "My Folder"
  },
  "relatedItems": [],
  "review_state": "private",
  "rights": "",
  "sharing": {
    "@id": "http://localhost:55001/plone/folder/my-document/@sharing",
```

```
    "title": "Sharing"
  },
  "subjects": [],
  "table_of_contents": null,
  "text": null,
  "title": "My Document",
  "version": "current"
}
```

Note: For folderish types, collections or search results, the results will be **batched** if the size of the resultset exceeds the batch size. See *Batching* for more details on how to work with batched results.

Unsuccessful response (404 Not Found)

If a resource could not be found, the server will respond with *404 Not Found*:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
  'error': 'NotFound'
}
```

GET Implementation

A pseudo-code example of the GET implementation on the server:

```
try:
    order = getOrder()
    if order == None:
        # Not Found
        response.setStatus(404)
    else:
        # OK
        response.setStatus(200)
except:
    # Internal Server Error
    response.setStatus(500)
```

TODO: Link to the real implementation...

GET Responses

Possible server reponses for a GET request are:

- *200 OK*
- *404 Not Found*
- *500 Internal Server Error*

Updating a Resource with PATCH

To update an existing resource we send a PATCH request to the server. PATCH allows to provide just a subset of the resource (the values you actually want to change): http

```
PATCH /plone/folder/my-document HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "title": "My New Document Title"
}
```

curl

```
curl -i -X PATCH http://nohost/plone/folder/my-document -H "Accept: application/json"␣
↪-H "Content-Type: application/json" --data-raw '{"title": "My New Document Title"}'␣
↪--user admin:secret
```

httpie

```
http -j PATCH http://nohost/plone/folder/my-document title="My New Document Title" -a␣
↪admin:secret
```

python-requests

```
requests.patch('http://nohost/plone/folder/my-document', headers={'Accept':
↪'application/json'}, json={'title': 'My New Document Title'}, auth=('admin', 'secret
↪'))
```

Successful Response (204 No Content)

A successful response to a PATCH request will be indicated by a *204 No Content* response:

```
HTTP/1.1 204 No Content
```

See for full specs the RFC 5789: PATCH Method for HTTP

Replacing a Resource with PUT

Note: PUT is not implemented yet.

To replace an existing resource we send a PUT request to the server:

TODO: Add example.

In accordance with the HTTP specification, a successful PUT will not create a new resource or produce a new URL.

PUT expects the entire resource representation to be supplied to the server, rather than just changes to the resource state. This is usually not a problem since the consumer application requested the resource representation before a PUT anyways.

When the PUT request is accepted and processed by the service, the consumer will receive a *204 No Content* response (*200 OK* would be a valid alternative).

Successful Update (204 No Content)

When a resource has been updated successfully, the server sends a *204 No Content* response:

TODO: Add example.

Unsuccessful Update (409 Conflict)

Sometimes requests fail due to incompatible changes. The response body includes additional information about the problem.

TODO: Add example.

PUT Implementation

A pseudo-code example of the PUT implementation on the server:

```
try:
    order = getOrder()
    if order:
        try:
            saveOrder()
        except conflict:
            response.setStatus(409)
        # OK
        response.setStatus(200)
    else:
        # Not Found
        response.setStatus(404)
except:
    # Internal Server Error
    response.setStatus(500)
```

TODO: Link to the real implementation...

PUT Responses

Possible server reponses for a PUT request are:

- *200 OK*
- *404 Not Found*
- *409 Conflict*
- *500 Internal Server Error*

POST vs. PUT

Difference between POST and PUT:

- Use POST to create a resource identified by a service-generated URI
- Use POST to append a resource to a collection identified by a service-generated URI
- Use PUT to overwrite a resource

This follows RFC 7231: HTTP 1.1: PUT Method.

Removing a Resource with DELETE

We can delete an existing resource by sending a DELETE request: http

```
DELETE /plone/folder/my-document HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i -X DELETE http://nohost/plone/folder/my-document -H "Accept: application/json
↪" --user admin:secret
```

httpie

```
http -j DELETE http://nohost/plone/folder/my-document -a admin:secret
```

python-requests

```
requests.delete('http://nohost/plone/folder/my-document', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

A successful response will be indicated by a *204 No Content* response:

```
HTTP/1.1 204 No Content
```

DELETE Implementation

A pseudo-code example of the DELETE implementation on the server:

```
try:
    order = getOrder()
    if order:
        if can_delete(order):
            # No Content
            response.setStatus(204)
        else:
            # Not Allowed
            response.setStatus(405)
    else:
        # Not Found
        response.setStatus(404)
except:
    # Internal Server Error
    response.setStatus(500)
```

TODO: Link to the real implementation...

DELETE Responses

Possible responses to a delete request are:

- *204 No Content*
- *404 Not Found* (if the resource does not exist)
- *405 Method Not Allowed* (if deleting the resource is not allowed)

- *500 Internal Server Error*

Reordering sub resources

The resources contained within a resource can be reordered using the *ordering* key using a PATCH request on the container.

Use the *obj_id* subkey to specify which resource to reorder. The subkey *delta* can be 'top', 'bottom', or a negative or positive integer for moving up or down. Reordering resources within a subset of resources can be done using the *subset_ids* subkey. A response 400 BadRequest with a message 'Client/server ordering mismatch' will be returned if the value is not in the same order as serverside. http

```
PATCH /plone/folder/my-document HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "ordering": {"obj_id": "item_3", "delta": "top", "subset_ids": ["item_1", "item_3
↪", "item5"]}
}
```

curl

```
curl -i -X PATCH http://nohost/plone/folder/my-document -H "Accept: application/json"␣
↪-H "Content-Type: application/json" --data-raw '{"ordering": {"delta": "top", "obj_
↪id": "item_3", "subset_ids": ["item_1", "item_3", "item5"]}}' --user admin:secret
```

httpie

```
http -j PATCH http://nohost/plone/folder/my-document ordering:='{"delta": "top", "obj_
↪id": "item_3", "subset_ids": ["item_1", "item_3", "item5"]}' -a admin:secret
```

python-requests

```
requests.patch('http://nohost/plone/folder/my-document', headers={'Accept':
↪'application/json'}, json={'ordering': OrderedDict([(u'delta', u'top'), (u'obj_id',␣
↪u'item_3'), (u'subset_ids', [u'item_1', u'item_3', u'item5'])])}, auth=('admin',
↪'secret'))
```

History

The @history endpoint exposes history and versioning information on previous versions of the content. Each change or workflow change on a content object or file is listed. It also allows to revert to a previous version of the file.

Listing the History of a Content Object

Listing versions and history of a resource: http

```
GET /plone/front-page/@history HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/front-page/@history -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/front-page/@history -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/front-page/@history', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "action": "Create",
    "actor": {
      "@id": "http://localhost:55001/plone/@users/test_user_1_",
      "fullname": "",
      "id": "test_user_1_",
      "username": "test-user"
    },
    "comments": "",
    "review_state": "private",
    "state_title": "Private",
    "time": "2016-10-21T19:00:00",
    "transition_title": "Create",
    "type": "workflow"
  },
  {
    "@id": "http://localhost:55001/plone/front-page/@history/0",
    "action": "Edited",
    "actor": {
      "@id": "http://localhost:55001/plone/@users/test-user",
      "fullname": "test-user",
      "id": "test-user",
      "username": null
    },
    "comments": null,
    "may_revert": true,
    "time": "2016-10-21T19:00:00",
    "transition_title": "Edited",
    "type": "versioning",
    "version": 0
  }
]
```

This following fields are returned:

- action: the workflow transition id, 'Edited' for versioning, or 'Create' for initial state.
- actor: the user who performed the action. This contains a subobject with the details.
- comments: a changenote
- @id: link to the content endpoint of this specific version.

- may_revert: true if the user has permission to revert.
- time: when this action occured in ISO format.
- transition_title: the workflow transition's title, 'Edited' for versioning, or 'Create' for initial state.
- type: 'workflow' for workflow changes, 'versioning' for editing, or null for content creation.
- version: identifier for this specific version of the resource.

Get a Historical Version

Older versions of a resource can be retrieved by appending *version* to the @history endpoint url. http

```
GET /plone/folder/my-document/@history/0 HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/folder/my-document/@history/0 -H "Accept: application/json
↪" --user admin:secret
```

httpie

```
http -j http://nohost/plone/folder/my-document/@history/0 -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/folder/my-document/@history/0', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

Revert to a Historical Version

Reverting to an older versions of a resource can be done by sending a PATCH request to the @history endpoint and appending the version you want to revert to. http

```
PATCH /plone/front-page/@history HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "version": 0
}
```

curl

```
curl -i -X PATCH http://nohost/plone/front-page/@history -H "Accept: application/json
↪" -H "Content-Type: application/json" --data-raw '{"version": 0}' --user␣
↪admin:secret
```

httpie

```
http -j PATCH http://nohost/plone/front-page/@history version:=0 -a admin:secret
```

python-requests

```
requests.patch('http://nohost/plone/front-page/@history', headers={'Accept':
↪'application/json'}, json={'version': 0}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "message": "Welcome to Plone has been reverted to revision 0."
}
```

Batching

Representations of collection-like resources are batched / paginated if the size of the resulset exceeds the batching size:

```
{
  "@id": "http://.../folder/search",
  "batching": {
    "@id": "http://.../folder/search?b_size=10&b_start=20",
    "first": "http://.../plone/folder/search?b_size=10&b_start=0",
    "last": "http://.../plone/folder/search?b_size=10&b_start=170",
    "prev": "http://.../plone/folder/search?b_size=10&b_start=10",
    "next": "http://.../plone/folder/search?b_size=10&b_start=30"
  },
  "items": [
    "..."
  ],
  "items_total": 175,
}
```

If the entire resulset fits into a single batch page (as determined by `b_size`), the top-level `batching` links will be omitted.

Top-level attributes

Attribute	Description
`@id`	Canonical base URL for the resource, without any batching parameters
`items`	Current batch of items / members of the collection-like resource
`items_total`	Total number of items
`batching`	Batching related navigation links (see below)

Batching links

If, and only if, the resultset has been batched over several pages, the response body will contain a top-level attribute `batching` that contains batching links. These links that can be used to navigate batches in a Hypermedia fashion:

Attribute	Description
`@id`	Link to the current batch page
`first`	Link to the first batch page
`prev`	Link to the previous batch page (*if applicable*)
`next`	Link to the next batch page (*if applicable*)
`last`	Link to the last batch page

Parameters

Batching can be controlled with two query string parameters. In order to address a specific batch page, the `b_start` parameter can be used to request a specific batch page, containing `b_size` items starting from `b_start`.

Parameter	Description
`b_size`	Batch size (default is 25)
`b_start`	First item of the batch

Full example of a batched request and response: http

```
GET /plone/folder/@search?b_size=5&sort_on=path HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/folder/@search?b_size=5&sort_on=path -H "Accept:␣
↪application/json" --user admin:secret
```

httpie

```
http -j http://nohost/plone/folder/@search?b_size=5&sort_on=path -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/folder/@search?b_size=5&sort_on=path', headers={
↪'Accept': 'application/json'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/folder/@search",
  "batching": {
    "@id": "http://localhost:55001/plone/folder/@search?b_size=5&sort_on=path",
    "first": "http://localhost:55001/plone/folder/@search?b_size=5&sort_on=path&b_
↪start=0",
    "last": "http://localhost:55001/plone/folder/@search?b_size=5&sort_on=path&b_
↪start=5",
    "next": "http://localhost:55001/plone/folder/@search?b_size=5&sort_on=path&b_
↪start=5"
  },
  "items": [
    {
      "@id": "http://localhost:55001/plone/folder",
      "@type": "Folder",
      "description": "",
      "review_state": "private",
      "title": "Folder"
    },
    {
      "@id": "http://localhost:55001/plone/folder/doc-1",
      "@type": "Document",
      "description": "",
      "review_state": "private",
      "title": "Document 1"
    },
```

```
    {
      "@id": "http://localhost:55001/plone/folder/doc-2",
      "@type": "Document",
      "description": "",
      "review_state": "private",
      "title": "Document 2"
    },
    {
      "@id": "http://localhost:55001/plone/folder/doc-3",
      "@type": "Document",
      "description": "",
      "review_state": "private",
      "title": "Document 3"
    },
    {
      "@id": "http://localhost:55001/plone/folder/doc-4",
      "@type": "Document",
      "description": "",
      "review_state": "private",
      "title": "Document 4"
    }
  ],
  "items_total": 8
}
```


Comments

Plone offers users to post comments on any content object with plone.app.discussion.

Commenting can be enabled globally, for specific content types and for single content objects.

When commenting is enabled on your content object, you can retrieve a list of all existing comments, add new comments, reply to existing comments or delete a comment.

Listing Comments

You can list the existing comment on a content object by sending a GET request to the URL of the content object and appending '/@comments': http

```
GET /plone/front-page/@comments HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/front-page/@comments -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/front-page/@comments -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/front-page/@comments', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

The server will respond with a *Status 200* and a batched list of all comments:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/front-page/@comments",
  "items": [
    {
      "@id": "http://localhost:55001/plone/front-page/@comments/1477076400000000",
      "@parent": null,
      "@type": "Discussion Item",
      "author_name": null,
      "author_username": null,
      "comment_id": "1477076400000000",
      "creation_date": "2016-10-21T19:00:00",
      "in_reply_to": null,
      "modification_date": "2016-10-21T19:00:00",
      "text": {
        "data": "Comment 1",
        "mime-type": "text/plain"
      },
      "user_notification": null
    },
    {
      "@id": "http://localhost:55001/plone/front-page/@comments/1477076400000001",
      "@parent": "http://localhost:55001/plone/front-page/@comments/1477076400000000",
      "@type": "Discussion Item",
      "author_name": null,
      "author_username": null,
      "comment_id": "1477076400000001",
      "creation_date": "2016-10-21T19:00:00",
      "in_reply_to": "1477076400000000",
      "modification_date": "2016-10-21T19:00:00",
      "text": {
        "data": "Comment 1.1",
        "mime-type": "text/plain"
      },
      "user_notification": null
    }
  ],
  "items_total": 2
}
```

These following fields are returned:

- @id: Link to the current endpoint
- items: a list of comments for the current resource
- items_total: the total number of comments for the resource
- batching: batching information

The items attribute returns a list of comments, each comment provides the following fields:

- @id: hyperlink to the comment

- @parent: (optional) the parent comment
- author_name: the full name of the author of this comment
- author_username: the username of the author of this comment
- comment_id: the comment ID uniquely identifies the comment
- in_reply_to: the comment ID of the parent comment
- creation_date: when the comment was placed
- modification_date: when the comment was last updated
- text: contains a 'mime-type' and 'text' attribute with the text of the comment. Default mime-type is 'text/plain'.
- user_notification: boolean value to indicate if the author of the comment requested notifications on replies

Adding a Comment

To add a new comment to a content object, send a POST request to the URL of the content object and append '/@comments' to the URL. The body of the request needs to contain a JSON structure with a 'text' attribute that contains the comment text: http

```
POST /plone/front-page/@comments/ HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "text": "My comment"
}
```

curl

```
curl -i -X POST http://nohost/plone/front-page/@comments/ -H "Accept: application/json
↪" -H "Content-Type: application/json" --data-raw '{"text": "My comment"}' --user␣
↪admin:secret
```

httpie

```
http -j POST http://nohost/plone/front-page/@comments/ text="My comment" -a␣
↪admin:secret
```

python-requests

```
requests.post('http://nohost/plone/front-page/@comments/', headers={'Accept':
↪'application/json'}, json={'text': 'My comment'}, auth=('admin', 'secret'))
```

If the creation of the comment has been successful, the server will respond with a *204 No Content* status and the URL of the newly created comment in the location header:

```
HTTP/1.1 204 No Content
Location: http://localhost:55001/plone/front-page/@comments/123456
```

Replying to a Comment

To add a direct reply to an existing comment, send a POST request to the URL of the comment you want to reply to. The body of the request needs to contain a JSON structure with a 'text' attribute that contains the comment text: http

```
POST /plone/front-page/@comments/123456 HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "text": "My reply"
}
```

curl

```
curl -i -X POST http://nohost/plone/front-page/@comments/123456 -H "Accept:␣
↪application/json" -H "Content-Type: application/json" --data-raw '{"text": "My reply
↪"}' --user admin:secret
```

httpie

```
http -j POST http://nohost/plone/front-page/@comments/123456 text="My reply" -a␣
↪admin:secret
```

python-requests

```
requests.post('http://nohost/plone/front-page/@comments/123456', headers={'Accept':
↪'application/json'}, json={'text': 'My reply'}, auth=('admin', 'secret'))
```

If the creation of the comment has been successful, the server will respond with a *204 No Content* status and the URL of the newly created comment in the location header:

```
HTTP/1.1 204 No Content
Location: http://localhost:55001/plone/front-page/@comments/123456
```

Updating a Comment

..note: The permission to update a comment is, by default, only granted to the creater (owner role) of the comment.

An existing comment can be updated by sending a PATCH request to the URL of the comment. The request body needs to contain a JSON structure with at least a 'text' attribute: http

```
PATCH /plone/front-page/@comments/123456 HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "text": "My NEW comment"
}
```

curl

```
curl -i -X PATCH http://nohost/plone/front-page/@comments/123456 -H "Accept:␣
↪application/json" -H "Content-Type: application/json" --data-raw '{"text": "My NEW␣
↪comment"}' --user admin:secret
```

httpie

```
http -j PATCH http://nohost/plone/front-page/@comments/123456 text="My NEW comment" -
↪a admin:secret
```

python-requests

```
requests.patch('http://nohost/plone/front-page/@comments/123456', headers={'Accept':
↪'application/json'}, json={'text': 'My NEW comment'}, auth=('admin', 'secret'))
```

The server will respond with a *204 No Content* response and a location header with the comment URL when the comment has been updated successfully:

```
HTTP/1.1 204 No Content
Location: http://localhost:55001/plone/front-page/@comments/123456
```

Deleting a Comment

An existing comment can be deleted by sending a DELETE request to the URL of the comment.

..note: Deleting a comment will, by default, also delete all existing replies to that comment. http

```
DELETE /plone/front-page/@comments/123456 HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i -X DELETE http://nohost/plone/front-page/@comments/123456 -H "Accept:␣
↪application/json" --user admin:secret
```

httpie

```
http -j DELETE http://nohost/plone/front-page/@comments/123456 -a admin:secret
```

python-requests

```
requests.delete('http://nohost/plone/front-page/@comments/123456', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

When the comment has been deleted successfully, the server will respond with a *204 No Content* response:

```
HTTP/1.1 204 No Content
```

Copy / Move

Copying an object

To copy a content object send a POST request to the `/@copy` endpoint at the destinations url with the source object specified in the request body. The source object can be specified either by url, path, UID or intid. http

```
POST /plone/@copy HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json
```

```
{
    "source": "http://localhost:55001/plone/front-page"
}
```

curl

```
curl -i -X POST http://nohost/plone/@copy -H "Accept: application/json" -H "Content-
↪Type: application/json" --data-raw '{"source": "http://localhost:55001/plone/front-
↪page"}' --user admin:secret
```

httpie

```
http -j POST http://nohost/plone/@copy source=http://localhost:55001/plone/front-page␣
↪-a admin:secret
```

python-requests

```
requests.post('http://nohost/plone/@copy', headers={'Accept': 'application/json'},␣
↪json={'source': 'http://localhost:55001/plone/front-page'}, auth=('admin', 'secret
↪'))
```

If the copy operation succeeds, the server will respond with status 200 (OK) and return the new and old url of the copied object.

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "source": "http://localhost:55001/plone/front-page",
    "target": "http://localhost:55001/plone/copy_of_front-page"
  }
]
```

Moving an object

To move a content object send a POST request to the `/@move` endpoint at the destinations url with the source object specified in the request body. The source object can be specified either by url, path, UID or intid. http

```
POST /plone/folder/@move HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "source": "http://localhost:55001/plone/front-page"
}
```

curl

```
curl -i -X POST http://nohost/plone/folder/@move -H "Accept: application/json" -H
↪"Content-Type: application/json" --data-raw '{"source": "http://localhost:55001/
↪plone/front-page"}' --user admin:secret
```

httpie

```
http -j POST http://nohost/plone/folder/@move source=http://localhost:55001/plone/
↪front-page -a admin:secret
```

python-requests

```
requests.post('http://nohost/plone/folder/@move', headers={'Accept': 'application/json
↪'}, json={'source': 'http://localhost:55001/plone/front-page'}, auth=('admin',
↪'secret'))
```

If the move operation succeeds, the server will respond with status 200 (OK) and return the new and old url of the moved object.

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "source": "http://localhost:55001/plone/front-page",
    "target": "http://localhost:55001/plone/folder/front-page"
  }
]
```

Copying/moving multiple objects

Multiple objects can be moved/copied by giving a list of sources. http

```
POST /plone/@copy HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "source": [
        "http://localhost:55001/plone/front-page",
        "http://localhost:55001/plone/newsitem"
    ]
}
```

curl

```
curl -i -X POST http://nohost/plone/@copy -H "Accept: application/json" -H "Content-
↪Type: application/json" --data-raw '{"source": ["http://localhost:55001/plone/front-
↪page", "http://localhost:55001/plone/newsitem"]}' --user admin:secret
```

httpie

```
http -j POST http://nohost/plone/@copy source:='["http://localhost:55001/plone/front-
↪page", "http://localhost:55001/plone/newsitem"]' -a admin:secret
```

python-requests

```
requests.post('http://nohost/plone/@copy', headers={'Accept': 'application/json'},␣
↪json={'source': [u'http://localhost:55001/plone/front-page', u'http://
↪localhost:55001/plone/newsitem']}, auth=('admin', 'secret'))
```

If the operation succeeds, the server will respond with status 200 (OK) and return the new and old urls for each copied/moved object.

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "source": "http://localhost:55001/plone/front-page",
    "target": "http://localhost:55001/plone/copy_of_front-page"
  },
  {
    "source": "http://localhost:55001/plone/newsitem",
    "target": "http://localhost:55001/plone/copy_of_newsitem"
  }
]
```

Workflow

Note: Currently the workflow support is limited to executing transitions on content.

In Plone, content almost always has a *workflow* attached. We can get the current state and history of an object by issuing a `GET` request using on any context: http

```
GET /plone/front-page/@workflow HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/front-page/@workflow -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/front-page/@workflow -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/front-page/@workflow', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "history": [
    {
      "action": null,
      "actor": "test_user_1_",
      "comments": "",
      "review_state": "private",
      "time": "2016-10-21T19:00:00+00:00",
      "title": "Private"
```

```
    }
  ],
  "transitions": [
    {
      "@id": "http://localhost:55001/plone/front-page/@workflow/publish",
      "title": "Publish"
    },
    {
      "@id": "http://localhost:55001/plone/front-page/@workflow/submit",
      "title": "Submit for publication"
    }
  ]
}
```

Now, if we want to change the state of the front page to publish, we would proceed by issuing a `POST` request to the given URL: http

```
POST /plone/front-page/@workflow/publish HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i -X POST http://nohost/plone/front-page/@workflow/publish -H "Accept:␣
↪application/json" --user admin:secret
```

httpie

```
http -j POST http://nohost/plone/front-page/@workflow/publish -a admin:secret
```

python-requests

```
requests.post('http://nohost/plone/front-page/@workflow/publish', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "action": "publish",
  "actor": "admin",
  "comments": "",
  "review_state": "published",
  "time": "2016-10-21T19:05:00+00:00"
}
```

Sharing

Plone comes with a sophisticated user management system that allows to assign users and groups with global roles and permissions. Sometimes this in not enough though and you might want to give users the permission to access or edit a specific part of your website or a specific content object. This is where local roles (located in the Plone sharing tab) come in handy.

Retrieve Local Roles

In plone.restapi, the representation of any content object will include a hypermedia link to the local role / sharing information in the 'sharing' attribute:

```
GET /plone/folder
Accept: application/json

HTTP 200 OK
content-type: application/json

{
  "@id": "http://localhost:55001/plone/folder",
  "@type": "Folder",
  ...
  "sharing": {
    "@id": "http://localhost:55001/plone/folder/@sharing",
    "title": "Sharing",
  }
}
```

The sharing information of a content object can also be directly accessed by appending '/@sharing' to the GET request to the URL of a content object. E.g. to access the sharing information for a top-level folder, do: http

```
GET /plone/folder/@sharing HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/folder/@sharing -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/folder/@sharing -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/folder/@sharing', headers={'Accept': 'application/
↪json'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "available_roles": [
    "Contributor",
    "Editor",
    "Reviewer",
    "Reader"
  ],
  "entries": [
    {
      "disabled": false,
      "id": "AuthenticatedUsers",
      "login": null,
      "roles": {
```

```
        "Contributor": false,
        "Editor": false,
        "Reader": false,
        "Reviewer": false
      },
      "title": "Logged-in users",
      "type": "group"
    }
  ],
  "inherit": true
}
```

Users and/or groups without a sharing entry can be found by appending the argument *search* to the query string. ie search=admin. Global roles are marked with the string "global". Inherited roles are marked with the string "acquired".
http

```
GET /plone/folder/doc/@sharing?search=admin HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/folder/doc/@sharing?search=admin -H "Accept: application/
↪json" --user admin:secret
```

httpie

```
http -j http://nohost/plone/folder/doc/@sharing?search=admin -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/folder/doc/@sharing?search=admin', headers={'Accept
↪': 'application/json'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "available_roles": [
    "Contributor",
    "Editor",
    "Reviewer",
    "Reader"
  ],
  "entries": [
    {
      "id": "Administrators",
      "login": null,
      "roles": {
        "Contributor": false,
        "Editor": false,
        "Reader": false,
        "Reviewer": false
      },
      "title": "Administrators",
      "type": "group"
    },
```

```
    {
      "disabled": false,
      "id": "AuthenticatedUsers",
      "login": null,
      "roles": {
        "Contributor": false,
        "Editor": false,
        "Reader": false,
        "Reviewer": false
      },
      "title": "Logged-in users",
      "type": "group"
    },
    {
      "id": "Site Administrators",
      "login": null,
      "roles": {
        "Contributor": false,
        "Editor": false,
        "Reader": false,
        "Reviewer": false
      },
      "title": "Site Administrators",
      "type": "group"
    },
    {
      "disabled": true,
      "id": "admin",
      "roles": {
        "Contributor": "global",
        "Editor": "acquired",
        "Reader": false,
        "Reviewer": false
      },
      "title": "admin",
      "type": "user"
    }
  ],
  "inherit": true
}
```

Update Local Roles

You can update the 'sharing' information by sending a POST request to the object URL and appending '/@sharing', e.g. '/plone/folder/@sharing'. E.g. say you want to give the AuthenticatedUsers group the 'reader' local role for a folder: http

```
POST /plone/folder/@sharing HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "entries": [
        {
            "id": "test_user_1_",
```

```
            "roles": {
                "Contributor": false,
                "Editor": false,
                "Reader": true,
                "Reviewer": true
            },
            "type": "user"
        }
    ],
    "inherit": true
}
```

curl

```
curl -i -X POST http://nohost/plone/folder/@sharing -H "Accept: application/json" -H
↪"Content-Type: application/json" --data-raw '{"entries": [{"type": "user", "id":
↪"test_user_1_", "roles": {"Contributor": false, "Reviewer": true, "Editor": false,
↪"Reader": true}}], "inherit": true}' --user admin:secret
```

httpie

```
http -j POST http://nohost/plone/folder/@sharing entries:='[{"type": "user", "id":
↪"test_user_1_", "roles": {"Contributor": false, "Reviewer": true, "Editor": false,
↪"Reader": true}}]' inherit:=true -a admin:secret
```

python-requests

```
requests.post('http://nohost/plone/folder/@sharing', headers={'Accept': 'application/
↪json'}, json={'entries': [{u'type': u'user', u'id': u'test_user_1_', u'roles': {u
↪'Contributor': False, u'Reviewer': True, u'Editor': False, u'Reader': True}}],
↪'inherit': True}, auth=('admin', 'secret'))
```

```
HTTP/1.1 204 No Content
```

Registry

Registry records can be addressed through the `@registry` endpoint on the Plone site. In order to address a specific record, the fully qualified dotted name of the registry record has to be passed as a path segment (e.g. */plone/@registy/my.record*).

Reading or writing registry records require the `cmf.ManagePortal` permission.

Reading registry records

Reading a single record: http

```
GET /plone/@registry/plone.app.querystring.field.path.title HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@registry/plone.app.querystring.field.path.title -H
↪"Accept: application/json" --user admin:secret
```

httpie

```
http -j http://nohost/plone/@registry/plone.app.querystring.field.path.title -a␣
↪admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@registry/plone.app.querystring.field.path.title',␣
↪headers={'Accept': 'application/json'}, auth=('admin', 'secret'))
```

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"Location"
```

Listing registry records

The registry records listing uses a batched method to access all registry records. See *Batching* for more details on how to work with batched results.

The output per record contains the following fields: name: The record's fully qualified dotted name. value: The record's value. This is the same as GETting *@registry/name*. http

```
GET /plone/@registry HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@registry -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/@registry -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@registry', headers={'Accept': 'application/json'},␣
↪auth=('admin', 'secret'))
```

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/@registry",
  "batching": {
    "@id": "http://localhost:55001/plone/@registry",
    "first": "http://localhost:55001/plone/@registry?b_start=0",
    "last": "http://localhost:55001/plone/@registry?b_start=275",
    "next": "http://localhost:55001/plone/@registry?b_start=25"
  },
  "items": [
```

```
    {
      "name": "Products.CMFPlone.i18nl10n.override_dateformat.Enabled",
      "schema": {
        "properties": {
          "description": "Override the translation machinery",
          "title": "Enabled",
          "type": "boolean"
        }
      },
      "value": false
    },
    {
      "name": "Products.CMFPlone.i18nl10n.override_dateformat.date_format_long",
      "schema": {
        "properties": {
          "description": "Default value: %Y-%m-%d %H:%M (2038-01-19 03:14)",
          "title": "old ZMI property: localLongTimeFormat",
          "type": "string"
        }
      },
      "value": "%Y-%m-%d %H:%M"
    },
    {
      "name": "Products.CMFPlone.i18nl10n.override_dateformat.date_format_short",
      "schema": {
        "properties": {
          "description": "Default value: %Y-%m-%d (2038-01-19)",
          "title": "old ZMI property: localTimeFormat",
          "type": "string"
        }
      },
      "value": "%Y-%m-%d"
    },
    {
      "name": "Products.CMFPlone.i18nl10n.override_dateformat.time_format",
      "schema": {
        "properties": {
          "description": "Default value: %H:%M (03:14)",
          "title": "old ZMI property: localTimeOnlyFormat",
          "type": "string"
        }
      },
      "value": "%H:%M"
    },
    {
      "name": "Products.CMFPlone.interfaces.syndication.ISiteSyndicationSettings.
↪allowed",
      "schema": {
        "properties": {
          "default": true,
          "description": "Allow syndication for collections and folders on site.",
          "title": "Allowed",
          "type": "boolean"
        }
      },
      "value": true
    },
    {
```

```
      "name": "Products.CMFPlone.interfaces.syndication.ISiteSyndicationSettings.
↪allowed_feed_types",
      "schema": {
        "properties": {
          "additionalItems": true,
          "default": [
            "RSS|RSS 1.0",
            "rss.xml|RSS 2.0",
            "atom.xml|Atom",
            "itunes.xml|iTunes"
          ],
          "description": "Separate view name and title by '|'",
          "items": {
            "description": "",
            "title": "",
            "type": "string"
          },
          "title": "Allowed Feed Types",
          "type": "array",
          "uniqueItems": true
        }
      },
      "value": [
        "RSS|RSS 1.0",
        "rss.xml|RSS 2.0",
        "atom.xml|Atom",
        "itunes.xml|iTunes"
      ]
    },
    {
      "name": "Products.CMFPlone.interfaces.syndication.ISiteSyndicationSettings.
↪default_enabled",
      "schema": {
        "properties": {
          "default": false,
          "description": "If syndication should be enabled by default for all folders␣
↪and collections.",
          "title": "Enabled by default",
          "type": "boolean"
        }
      },
      "value": false
    },
    {
      "name": "Products.CMFPlone.interfaces.syndication.ISiteSyndicationSettings.max_
↪items",
      "schema": {
        "properties": {
          "default": 15,
          "description": "Maximum number of items that will be syndicated.",
          "title": "Maximum Items",
          "type": "integer"
        }
      },
      "value": 15
    },
    {
      "name": "Products.CMFPlone.interfaces.syndication.ISiteSyndicationSettings.
↪render_body",
```

```
      "schema": {
        "properties": {
          "default": false,
          "description": "If body text available for item, render it, otherwise use␣
↪description.",
          "title": "Render Body",
          "type": "boolean"
        }
      },
      "value": false
    },
    {
      "name": "Products.CMFPlone.interfaces.syndication.ISiteSyndicationSettings.
↪search_rss_enabled",
      "schema": {
        "properties": {
          "default": true,
          "description": "Allows users to subscribe to feeds of search results",
          "title": "Search RSS enabled",
          "type": "boolean"
        }
      },
      "value": true
    },
    {
      "name": "Products.CMFPlone.interfaces.syndication.ISiteSyndicationSettings.show_
↪author_info",
      "schema": {
        "properties": {
          "default": true,
          "description": "Should feeds include author information",
          "title": "Show author info",
          "type": "boolean"
        }
      },
      "value": true
    },
    {
      "name": "Products.CMFPlone.interfaces.syndication.ISiteSyndicationSettings.show_
↪syndication_button",
      "schema": {
        "properties": {
          "description": "Makes it possible to customize syndication settings for␣
↪particular folders and collections ",
          "title": "Show Settings Button",
          "type": "boolean"
        }
      },
      "value": null
    },
    {
      "name": "Products.CMFPlone.interfaces.syndication.ISiteSyndicationSettings.show_
↪syndication_link",
      "schema": {
        "properties": {
          "description": "Enable RSS link document action on the syndication content␣
↪item.",
          "title": "Show Feed Link",
```

```
          "type": "boolean"
        }
      },
      "value": null
    },
    {
      "name": "Products.CMFPlone.interfaces.syndication.ISiteSyndicationSettings.site_
↪rss_items",
      "schema": {
        "properties": {
          "additionalItems": true,
          "default": [
            "/news/aggregator"
          ],
          "description": "Paths to folders and collections to link to at the portal␣
↪root.",
          "items": {
            "choices": [],
            "description": "",
            "enum": [],
            "enumNames": [],
            "title": "",
            "type": "string"
          },
          "title": "Site RSS",
          "type": "array",
          "uniqueItems": true
        }
      },
      "value": [
        "/news/aggregator"
      ]
    },
    {
      "name": "Products.ResourceRegistries.interfaces.settings.
↪IResourceRegistriesSettings.resourceBundlesForThemes",
      "schema": {
        "properties": {
          "description": "Maps skin names to lists of resource bundle names",
          "key_type": {
            "additional": {},
            "schema": {
              "description": "",
              "title": "",
              "type": "string"
            }
          },
          "title": "Resource bundles for themes",
          "type": "dict",
          "value_type": {
            "additional": {},
            "schema": {
              "description": "",
              "title": "",
              "type": "string"
            }
          }
        }
```

```
      },
      "value": {
        "(default)": [
          "jquery",
          "default"
        ]
      }
    },
    {
      "name": "plone.app.discussion.interfaces.IDiscussionSettings.anonymous_comments
↪",
      "schema": {
        "properties": {
          "default": false,
          "description": "If selected, anonymous users are able to post comments␣
↪without loggin in. It is highly recommended to use a captcha solution to prevent␣
↪spam if this setting is enabled.",
          "title": "Enable anonymous comments",
          "type": "boolean"
        }
      },
      "value": false
    },
    {
      "name": "plone.app.discussion.interfaces.IDiscussionSettings.anonymous_email_
↪enabled",
      "schema": {
        "properties": {
          "default": false,
          "description": "If selected, anonymous user will have to give their email.",
          "title": "Enable anonymous email field",
          "type": "boolean"
        }
      },
      "value": false
    },
    {
      "name": "plone.app.discussion.interfaces.IDiscussionSettings.captcha",
      "schema": {
        "properties": {
          "choices": [
            [
              "disabled",
              "Disabled"
            ]
          ],
          "default": "disabled",
          "description": "Use this setting to enable or disable Captcha validation␣
↪for comments. Install plone.formwidget.captcha, plone.formwidget.recaptcha,␣
↪collective.akismet, or collective.z3cform.norobots if there are no options␣
↪available.",
          "enum": [
            "disabled"
          ],
          "enumNames": [
            "Disabled"
          ],
          "title": "Captcha",
```

```
          "type": "string"
        }
      },
      "value": "disabled"
    },
    {
      "name": "plone.app.discussion.interfaces.IDiscussionSettings.edit_comment_
↪enabled",
      "schema": {
        "properties": {
          "default": false,
          "description": "If selected, supports editing and deletion of comments for␣
↪users with the 'Edit comments' permission.",
          "title": "Enable editing of comments",
          "type": "boolean"
        }
      },
      "value": false
    },
    {
      "name": "plone.app.discussion.interfaces.IDiscussionSettings.globally_enabled",
      "schema": {
        "properties": {
          "default": false,
          "description": "If selected, users are able to post comments on the site.␣
↪Though, you have to enable comments for specific content types, folders or content␣
↪objects before users will be able to post comments.",
          "title": "Globally enable comments",
          "type": "boolean"
        }
      },
      "value": false
    },
    {
      "name": "plone.app.discussion.interfaces.IDiscussionSettings.moderation_enabled
↪",
      "schema": {
        "properties": {
          "default": false,
          "description": "If selected, comments will enter a 'Pending' state in which␣
↪they are invisible to the public. A user with the 'Review comments' permission (
↪'Reviewer' or 'Manager') can approve comments to make them visible to the public.␣
↪If you want to enable a custom comment workflow, you have to go to the types␣
↪control panel.",
          "title": "Enable comment moderation",
          "type": "boolean"
        }
      },
      "value": false
    },
    {
      "name": "plone.app.discussion.interfaces.IDiscussionSettings.moderator_email",
      "schema": {
        "properties": {
          "description": "Address to which moderator notifications will be sent.",
          "title": "Moderator Email Address",
          "type": "string"
        }
```

```
      },
      "value": null
    },
    {
      "name": "plone.app.discussion.interfaces.IDiscussionSettings.moderator_
↪notification_enabled",
      "schema": {
        "properties": {
          "default": false,
          "description": "If selected, the moderator is notified if a comment needs␣
↪attention. The moderator email address can be found in the 'Mail settings' control␣
↪panel (Site 'From' address)",
          "title": "Enable moderator email notification",
          "type": "boolean"
        }
      },
      "value": false
    },
    {
      "name": "plone.app.discussion.interfaces.IDiscussionSettings.show_commenter_
↪image",
      "schema": {
        "properties": {
          "default": true,
          "description": "If selected, an image of the user is shown next to the␣
↪comment.",
          "title": "Show commenter image",
          "type": "boolean"
        }
      },
      "value": true
    },
    {
      "name": "plone.app.discussion.interfaces.IDiscussionSettings.text_transform",
      "schema": {
        "properties": {
          "choices": [
            [
              "text/plain",
              "Plain text"
            ],
            [
              "text/html",
              "HTML"
            ],
            [
              "text/x-web-markdown",
              "Markdown"
            ],
            [
              "text/x-web-intelligent",
              "Intelligent text"
            ]
          ],
          "default": "text/plain",
          "description": "Use this setting to choose if the comment text should be␣
↪transformed in any way. You can choose between 'Plain text' and 'Intelligent text'.
↪'Intelligent text' converts plain text into HTML where line breaks and indentation␣
↪is preserved, and web and email addresses are made into clickable links.",
```

```
          "enum": [
            "text/plain",
            "text/html",
            "text/x-web-markdown",
            "text/x-web-intelligent"
          ],
          "enumNames": [
            "Plain text",
            "HTML",
            "Markdown",
            "Intelligent text"
          ],
          "title": "Comment text transform",
          "type": "string"
        }
      },
      "value": "text/plain"
    }
  ],
  "items_total": 284
}
```

Updating registry records

Updating an existing record: http

```
PATCH /plone/@registry/ HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "plone.app.querystring.field.path.title": "Value"
}
```

curl

```
curl -i -X PATCH http://nohost/plone/@registry/ -H "Accept: application/json" -H
↪"Content-Type: application/json" --data-raw '{"plone.app.querystring.field.path.
↪title": "Value"}' --user admin:secret
```

httpie

```
http -j PATCH http://nohost/plone/@registry/ plone.app.querystring.field.path.
↪title=Value -a admin:secret
```

python-requests

```
requests.patch('http://nohost/plone/@registry/', headers={'Accept': 'application/json
↪'}, json={'plone.app.querystring.field.path.title': 'Value'}, auth=('admin', 'secret
↪'))
```

Example Response:

```
HTTP/1.1 204 No Content
```

Types

Note: These docs are generated by code tests, therefore you will see some 'test' contenttypes appear here.

Available content types in a Plone site can be listed and queried by accessing the `/@types` endpoint on any context (requires an authenticated user). The 'addable' key specifies if the content type can be added to the current context. The 'layouts' key specifies the defined views. http

```
GET /plone/@types HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@types -H "Accept: application/json" --user admin:secret
```

httpie

```
http -j http://nohost/plone/@types -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@types', headers={'Accept': 'application/json'},␣
↪auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "@id": "http://localhost:55001/plone/@types/Collection",
    "addable": true,
    "title": "Collection"
  },
  {
    "@id": "http://localhost:55001/plone/@types/DXTestDocument",
    "addable": true,
    "title": "DX Test Document"
  },
  {
    "@id": "http://localhost:55001/plone/@types/Event",
    "addable": true,
    "title": "Event"
  },
  {
    "@id": "http://localhost:55001/plone/@types/File",
    "addable": true,
    "title": "File"
  },
  {
    "@id": "http://localhost:55001/plone/@types/Folder",
    "addable": true,
    "title": "Folder"
  },
  {
```

```
    "@id": "http://localhost:55001/plone/@types/Image",
    "addable": true,
    "title": "Image"
  },
  {
    "@id": "http://localhost:55001/plone/@types/Link",
    "addable": true,
    "title": "Link"
  },
  {
    "@id": "http://localhost:55001/plone/@types/News Item",
    "addable": true,
    "title": "News Item"
  },
  {
    "@id": "http://localhost:55001/plone/@types/Document",
    "addable": true,
    "title": "Page"
  }
]
```

To get the schema of a content type, access the `/@types` endpoint with the name of the content type, e.g. '/plone/@types/Document': http

```
GET /plone/@types/Document HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@types/Document -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/@types/Document -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@types/Document', headers={'Accept': 'application/
↪json'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json+schema

{
  "fieldsets": [
    {
      "fields": [
        "title",
        "description",
        "text",
        "changeNote"
      ],
      "id": "default",
      "title": "Default"
    },
```

```
    {
      "fields": [
        "allow_discussion",
        "exclude_from_nav",
        "table_of_contents"
      ],
      "id": "settings",
      "title": "Settings"
    },
    {
      "fields": [
        "subjects",
        "language",
        "relatedItems"
      ],
      "id": "categorization",
      "title": "Categorization"
    },
    {
      "fields": [
        "effective",
        "expires"
      ],
      "id": "dates",
      "title": "Dates"
    },
    {
      "fields": [
        "creators",
        "contributors",
        "rights"
      ],
      "id": "ownership",
      "title": "Ownership"
    }
  ],
  "layouts": [
    "document_view"
  ],
  "properties": {
    "allow_discussion": {
      "choices": [
        [
          "True",
          "Yes"
        ],
        [
          "False",
          "No"
        ]
      ],
      "description": "Allow discussion for this content object.",
      "enum": [
        "True",
        "False"
      ],
      "enumNames": [
        "Yes",
```

```
        "No"
      ],
      "title": "Allow discussion",
      "type": "string"
    },
    "changeNote": {
      "description": "Enter a comment that describes the changes you made.",
      "title": "Change Note",
      "type": "string"
    },
    "contributors": {
      "additionalItems": true,
      "description": "The names of people that have contributed to this item. Each␣
↪contributor should be on a separate line.",
      "items": {
        "description": "",
        "title": "",
        "type": "string"
      },
      "title": "Contributors",
      "type": "array",
      "uniqueItems": true
    },
    "creators": {
      "additionalItems": true,
      "description": "Persons responsible for creating the content of this item.␣
↪Please enter a list of user names, one per line. The principal creator should come␣
↪first.",
      "items": {
        "description": "",
        "title": "",
        "type": "string"
      },
      "title": "Creators",
      "type": "array",
      "uniqueItems": true
    },
    "description": {
      "description": "Used in item listings and search results.",
      "minLength": 0,
      "title": "Summary",
      "type": "string",
      "widget": "textarea"
    },
    "effective": {
      "description": "If this date is in the future, the content will not show up in␣
↪listings and searches until this date.",
      "title": "Publishing Date",
      "type": "string",
      "widget": "datetime"
    },
    "exclude_from_nav": {
      "default": false,
      "description": "If selected, this item will not appear in the navigation tree",
      "title": "Exclude from navigation",
      "type": "boolean"
    },
    "expires": {
```

```
      "description": "When this date is reached, the content will nolonger be visible␣
↪in listings and searches.",
      "title": "Expiration Date",
      "type": "string",
      "widget": "datetime"
    },
    "language": {
      "choices": [
        [
          "de",
          "Deutsch"
        ],
        [
          "en",
          "English"
        ],
        [
          "es",
          "Espa\u00f1ol"
        ],
        [
          "fr",
          "Fran\u00e7ais"
        ]
      ],
      "default": "en",
      "description": "",
      "enum": [
        "de",
        "en",
        "es",
        "fr"
      ],
      "enumNames": [
        "Deutsch",
        "English",
        "Espa\u00f1ol",
        "Fran\u00e7ais"
      ],
      "title": "Language",
      "type": "string"
    },
    "relatedItems": {
      "additionalItems": true,
      "default": [],
      "description": "",
      "items": {
        "description": "",
        "title": "Related",
        "type": "string"
      },
      "title": "Related Items",
      "type": "array",
      "uniqueItems": true
    },
    "rights": {
      "description": "Copyright statement or other rights information on this item.",
      "minLength": 0,
```

```
      "title": "Rights",
      "type": "string",
      "widget": "textarea"
    },
    "subjects": {
      "additionalItems": true,
      "description": "Tags are commonly used for ad-hoc organization of content.",
      "items": {
        "description": "",
        "title": "",
        "type": "string"
      },
      "title": "Tags",
      "type": "array",
      "uniqueItems": true
    },
    "table_of_contents": {
      "description": "If selected, this will show a table of contents at the top of␣
↪the page.",
      "title": "Table of contents",
      "type": "boolean"
    },
    "text": {
      "description": "",
      "title": "Text",
      "type": "string",
      "widget": "richtext"
    },
    "title": {
      "description": "",
      "title": "Title",
      "type": "string"
    }
  },
  "required": [
    "title",
    "exclude_from_nav"
  ],
  "title": "Page",
  "type": "object"
}
```


The content type schema uses the JSON Schema format.

See types-schema for a detailed documentation about the available field types.

Users

Available users in a Plone site can be created, queried, updated and deleted by interacting with the `/@users` endpoint on portal root (requires an authenticated user):

List Users

To retrieve a list of all current users in the portal, call the `/@users` endpoint with a `GET` request: http

```
GET /plone/@users HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@users -H "Accept: application/json" --user admin:secret
```

httpie

```
http -j http://nohost/plone/@users -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@users', headers={'Accept': 'application/json'},␣
↪auth=('admin', 'secret'))
```

The server will respond with a list of all users in the portal:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "@id": "http://localhost:55001/plone/@users/admin",
    "description": "This is an admin user",
    "email": "admin@example.com",
    "fullname": "Administrator",
    "home_page": "http://www.example.com",
    "id": "admin",
    "location": "Berlin",
    "roles": [
      "Manager"
    ],
    "username": "admin"
  },
  {
    "@id": "http://localhost:55001/plone/@users/test_user_1_",
    "description": "This is a test user",
    "email": "test@example.com",
    "fullname": "Test User",
    "home_page": "http://www.example.com",
    "id": "test_user_1_",
    "location": "Bonn",
    "roles": [
      "Manager"
    ],
    "username": "test-user"
  }
]
```

The endpoint supports some basic filtering: http

```
GET /plone/@users?query=noa HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@users?query=noa -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/@users?query=noa -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@users?query=noa', headers={'Accept': 'application/
↪json'}, auth=('admin', 'secret'))
```

The server will respond with a list the filtered users in the portal with username starts with the query.

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "@id": "http://localhost:55001/plone/@users/noam",
    "description": "Professor of Linguistics",
    "email": "noam.chomsky@example.com",
    "fullname": "Noam Avram Chomsky",
    "home_page": "web.mit.edu/chomsky",
    "id": "noam",
    "location": "Cambridge, MA",
    "roles": [
      "Member"
    ],
    "username": "noam"
  }
]
```

The endpoint also takes a `limit` parameter that defaults to a maximum of 25 users at a time for performance reasons.

Create User

To create a new user, send a `POST` request to the global `/@users` endpoint with a JSON representation of the user you want to create in the body: http

```
POST /plone/@users HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "description": "Professor of Linguistics",
    "email": "noam.chomsky@example.com",
    "fullname": "Noam Avram Chomsky",
    "home_page": "web.mit.edu/chomsky",
    "location": "Cambridge, MA",
    "password": "colorlessgreenideas",
    "roles": [
        "Contributor"
    ],
    "username": "noamchomsky"
}
```

curl

```
curl -i -X POST http://nohost/plone/@users -H "Accept: application/json" -H "Content-
↪Type: application/json" --data-raw '{"description": "Professor of Linguistics",
↪"email": "noam.chomsky@example.com", "fullname": "Noam Avram Chomsky", "home_page":
↪"web.mit.edu/chomsky", "location": "Cambridge, MA", "password": "colorlessgreenideas
↪", "roles": ["Contributor"], "username": "noamchomsky"}' --user admin:secret
```

httpie

```
http -j POST http://nohost/plone/@users description="Professor of Linguistics"␣
↪email=noam.chomsky@example.com fullname="Noam Avram Chomsky" home_page=web.mit.edu/
↪chomsky location="Cambridge, MA" password=colorlessgreenideas roles:='["Contributor
↪"]' username=noamchomsky -a admin:secret
```

python-requests

```
requests.post('http://nohost/plone/@users', headers={'Accept': 'application/json'},␣
↪json={'description': 'Professor of Linguistics', 'email': 'noam.chomsky@example.com
↪', 'fullname': 'Noam Avram Chomsky', 'home_page': 'web.mit.edu/chomsky', 'location
↪': 'Cambridge, MA', 'password': 'colorlessgreenideas', 'roles': [u'Contributor'],
↪'username': 'noamchomsky'}, auth=('admin', 'secret'))
```

Note: By default, "username", and "password" are required fields. If email login is enabled, "email" and "password" are required fields. All other fields in the example are optional.

If the user has been created successfully, the server will respond with a status 201 (Created). The `Location` header contains the URL of the newly created user and the resource representation in the payload:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: http://localhost:55001/plone/@users/noamchomsky

{
  "@id": "http://localhost:55001/plone/@users/noamchomsky",
  "description": "Professor of Linguistics",
  "email": "noam.chomsky@example.com",
  "fullname": "Noam Avram Chomsky",
  "home_page": "web.mit.edu/chomsky",
  "id": "noamchomsky",
  "location": "Cambridge, MA",
  "roles": [
    "Contributor"
  ],
  "username": "noamchomsky"
}
```

Read User

To retrieve all details for a particular user, send a `GET` request to the `/@users` endpoint and append the user id to the URL: http

```
GET /plone/@users/noam HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@users/noam -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/@users/noam -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@users/noam', headers={'Accept': 'application/json'}
↪, auth=('admin', 'secret'))
```

The server will respond with a 200 OK status code and the JSON representation of the user in the body:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/@users/noam",
  "description": "Professor of Linguistics",
  "email": "noam.chomsky@example.com",
  "fullname": "Noam Avram Chomsky",
  "home_page": "web.mit.edu/chomsky",
  "id": "noam",
  "location": "Cambridge, MA",
  "roles": [
    "Member"
  ],
  "username": "noam"
}
```

The key 'roles' lists the globally defined roles for the user.

Update User

To update the settings of a user, send a `PATCH` request with the user details you want to amend to the URL of that particular user, e.g. if you want to update the email address of the admin user to: http

```
PATCH /plone/@users/noam HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "email": "avram.chomsky@example.com",
    "roles": {
        "Contributor": false
    }
}
```

curl

```
curl -i -X PATCH http://nohost/plone/@users/noam -H "Accept: application/json" -H
↪"Content-Type: application/json" --data-raw '{"email": "avram.chomsky@example.com",
↪"roles": {"Contributor": false}}' --user admin:secret
```

httpie

```
http -j PATCH http://nohost/plone/@users/noam email=avram.chomsky@example.com roles:='
↪{"Contributor": false}' -a admin:secret
```

python-requests

```
requests.patch('http://nohost/plone/@users/noam', headers={'Accept': 'application/json
↪'}, json={'email': 'avram.chomsky@example.com', 'roles': OrderedDict([(u'Contributor
↪', False)])}, auth=('admin', 'secret'))
```

A successful response to a PATCH request will be indicated by a *204 No Content* response:

```
HTTP/1.1 204 No Content
```

Note: The 'roles' object is a mapping of a role and a boolean indicating adding or removing.

Delete User

To delete a user send a `DELETE` request to the `/@users` endpoint and append the user id of the user you want to delete, e.g. to delete the user with the id johndoe: http

```
DELETE /plone/@users/noam HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i -X DELETE http://nohost/plone/@users/noam -H "Accept: application/json" --
↪user admin:secret
```

httpie

```
http -j DELETE http://nohost/plone/@users/noam -a admin:secret
```

python-requests

```
requests.delete('http://nohost/plone/@users/noam', headers={'Accept': 'application/
↪json'}, auth=('admin', 'secret'))
```

A successful response will be indicated by a *204 No Content* response:

```
HTTP/1.1 204 No Content
```

Reset User Password

Plone allows to reset a password for a user by sending a POST request to the user resource and appending */reset-password* to the URL:

```
POST /plone/@users/noam/reset-password HTTP/1.1
Host: localhost:8080
Accept: application/json
```

The server will respond with a *200 OK* response and send an email to the user to reset her password.

The token that is part of the reset url in the email can be used to authorize setting a new password:

```
..  http:example:: curl httpie python-requests
```

request _json/users_reset.req

Reset Own Password

Plone also allows a user to reset her own password directly without sending an email. The endpoint and the request is the same as above, but now the user can send the old password and the new password as payload:

```
POST /plone/@users/noam/reset-password HTTP/1.1
Host: localhost:8080
Accept: application/json
Content-Type: application/json

{
  'old_password': 'secret',
  'new_password': 'verysecret',
}
```

The server will respond with a *200 OK* response without sending an email.

To set the password with the old password you need either the `Set own password` or the `plone.app.controlpanel.UsersAndGroups` permission.

If an API consumer tries to send a password in the payload that is not the same as the currently logged in user, the server will respond with a *400 Bad Request* response.

Return Values

- *200 OK*
- *400 Bad Request*
- *403* (Unknown Token)
- *403* (Expired Token)
- *403* (Wrong user)
- *403* (Not allowed)
- *403* (Wrong password)
- *500 Internal Server Error* (server fault, can not recover internally)

Groups

Available groups in a Plone site can be created, queried, updated and deleted by interacting with the `/@groups` endpoint on portal root (requires an authenticated user):

List Groups

To retrieve a list of all current groups in the portal, call the `/@groups` endpoint with a `GET` request: http

```
GET /plone/@groups HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@groups -H "Accept: application/json" --user admin:secret
```

httpie

```
http -j http://nohost/plone/@groups -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@groups', headers={'Accept': 'application/json'},␣
↪auth=('admin', 'secret'))
```

The server will respond with a list of all groups in the portal:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "@id": "http://localhost:55001/plone/@groups/Administrators",
    "description": "",
    "email": "",
    "groupname": "Administrators",
    "id": "Administrators",
    "title": "Administrators"
  },
  {
    "@id": "http://localhost:55001/plone/@groups/Reviewers",
    "description": "",
    "email": "",
    "groupname": "Reviewers",
    "id": "Reviewers",
    "title": "Reviewers"
  },
  {
    "@id": "http://localhost:55001/plone/@groups/Site Administrators",
    "description": "",
    "email": "",
    "groupname": "Site Administrators",
    "id": "Site Administrators",
    "title": "Site Administrators"
  },
  {
    "@id": "http://localhost:55001/plone/@groups/ploneteam",
    "description": "We are Plone",
    "email": "ploneteam@plone.org",
    "groupname": "ploneteam",
    "id": "ploneteam",
    "title": "Plone Team"
```

```
  },
  {
    "@id": "http://localhost:55001/plone/@groups/AuthenticatedUsers",
    "description": "Automatic Group Provider",
    "email": "",
    "groupname": "AuthenticatedUsers",
    "id": "AuthenticatedUsers",
    "title": "Authenticated Users (Virtual Group)"
  }
]
```

The endpoint supports some basic filtering: http

```
GET /plone/@groups?query=plo HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@groups?query=plo -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/@groups?query=plo -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@groups?query=plo', headers={'Accept': 'application/
↪json'}, auth=('admin', 'secret'))
```

The server will respond with a list the filtered groups in the portal with groupname starts with the query.

The endpoint also takes a `limit` parameter that defaults to a maximum of 25 groups at a time for performance reasons.

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "@id": "http://localhost:55001/plone/@groups/ploneteam",
    "description": "We are Plone",
    "email": "ploneteam@plone.org",
    "groupname": "ploneteam",
    "id": "ploneteam",
    "title": "Plone Team"
  }
]
```

Create Group

To create a new group, send a `POST` request to the global `/@groups` endpoint with a JSON representation of the group you want to create in the body: http

```
POST /plone/@groups HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "description": "The Plone Framework Team",
    "email": "fwt@plone.org",
    "groupname": "fwt",
    "groups": [
        "Administrators"
    ],
    "roles": [
        "Manager"
    ],
    "title": "Framework Team",
    "users": [
        "admin",
        "test_user_1_"
    ]
}
```

curl

```
curl -i -X POST http://nohost/plone/@groups -H "Accept: application/json" -H "Content-
↪Type: application/json" --data-raw '{"description": "The Plone Framework Team",
↪"email": "fwt@plone.org", "groupname": "fwt", "groups": ["Administrators"], "roles
↪": ["Manager"], "title": "Framework Team", "users": ["admin", "test_user_1_"]}' --
↪user admin:secret
```

httpie

```
http -j POST http://nohost/plone/@groups description="The Plone Framework Team"␣
↪email=fwt@plone.org groupname=fwt groups:='["Administrators"]' roles:='["Manager"]'␣
↪title="Framework Team" users:='["admin", "test_user_1_"]' -a admin:secret
```

python-requests

```
requests.post('http://nohost/plone/@groups', headers={'Accept': 'application/json'},␣
↪json={'description': 'The Plone Framework Team', 'email': 'fwt@plone.org',
↪'groupname': 'fwt', 'groups': [u'Administrators'], 'roles': [u'Manager'], 'title':
↪'Framework Team', 'users': [u'admin', u'test_user_1_']}, auth=('admin', 'secret'))
```

Note: By default, "groupname" is a required field.

If the group has been created successfully, the server will respond with a status `201 (Created)`. The `Location` header contains the URL of the newly created group and the resource representation in the payload:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: http://localhost:55001/plone/@groups/fwt

{
  "@id": "http://localhost:55001/plone/@groups/fwt",
  "description": "The Plone Framework Team",
  "email": "fwt@plone.org",
```

```
  "groupname": "fwt",
  "id": "fwt",
  "title": "Framework Team",
  "users": {
    "@id": "http://localhost:55001/plone/@groups",
    "items": [
      "Administrators",
      "admin",
      "test_user_1_"
    ],
    "items_total": 3
  }
}
```

Read Group

To retrieve all details for a particular group, send a `GET` request to the `/@groups` endpoint and append the group id to the URL: http

```
GET /plone/@groups/ploneteam HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@groups/ploneteam -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/@groups/ploneteam -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@groups/ploneteam', headers={'Accept': 'application/
↪json'}, auth=('admin', 'secret'))
```

The server will respond with a `200 OK` status code and the JSON representation of the group in the body:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/@groups/ploneteam",
  "description": "We are Plone",
  "email": "ploneteam@plone.org",
  "groupname": "ploneteam",
  "id": "ploneteam",
  "title": "Plone Team",
  "users": {
    "@id": "http://localhost:55001/plone/@groups/ploneteam",
    "items": [],
    "items_total": 0
  }
}
```

Batching is supported for the 'users' object.

Update Group

To update the settings of a group, send a `PATCH` request with the group details you want to amend to the URL of that particular group: http

```
PATCH /plone/@groups/ploneteam HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Content-Type: application/json

{
    "email": "ploneteam2@plone.org",
    "users": {
        "test_user_1_": false
    }
}
```

curl

```
curl -i -X PATCH http://nohost/plone/@groups/ploneteam -H "Accept: application/json" -
↪H "Content-Type: application/json" --data-raw '{"email": "ploneteam2@plone.org",
↪"users": {"test_user_1_": false}}' --user admin:secret
```

httpie

```
http -j PATCH http://nohost/plone/@groups/ploneteam email=ploneteam2@plone.org users:=
↪'{"test_user_1_": false}' -a admin:secret
```

python-requests

```
requests.patch('http://nohost/plone/@groups/ploneteam', headers={'Accept':
↪'application/json'}, json={'email': 'ploneteam2@plone.org', 'users': OrderedDict([(u
↪'test_user_1_', False)])}, auth=('admin', 'secret'))
```

Note: The 'users' object is a mapping of a user_id and a boolean indicating adding or removing from the group.

A successful response to a PATCH request will be indicated by a *204 No Content* response:

```
HTTP/1.1 204 No Content
```

Delete Group

To delete a group send a `DELETE` request to the `/@groups` endpoint and append the group id of the group you want to delete: http

```
DELETE /plone/@groups/ploneteam HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i -X DELETE http://nohost/plone/@groups/ploneteam -H "Accept: application/json"␣
↪--user admin:secret
```

httpie

```
http -j DELETE http://nohost/plone/@groups/ploneteam -a admin:secret
```

python-requests

```
requests.delete('http://nohost/plone/@groups/ploneteam', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

A successful response will be indicated by a *204 No Content* response:

```
HTTP/1.1 204 No Content
```

Roles

Available roles in a Plone site can be queried by interacting with the `/@roles` endpoint on portal root (requires an authenticated user):

List Roles

To retrieve a list of all roles in the portal, call the `/@roles` endpoint with a `GET` request: http

```
GET /plone/@groups HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@groups -H "Accept: application/json" --user admin:secret
```

httpie

```
http -j http://nohost/plone/@groups -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@groups', headers={'Accept': 'application/json'},␣
↪auth=('admin', 'secret'))
```

The server will respond with a list of all groups in the portal:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "@id": "http://localhost:55001/plone/@roles/Contributor",
    "@type": "role",
    "id": "Contributor"
  },
  {
```

```
    "@id": "http://localhost:55001/plone/@roles/Editor",
    "@type": "role",
    "id": "Editor"
  },
  {
    "@id": "http://localhost:55001/plone/@roles/Member",
    "@type": "role",
    "id": "Member"
  },
  {
    "@id": "http://localhost:55001/plone/@roles/Reader",
    "@type": "role",
    "id": "Reader"
  },
  {
    "@id": "http://localhost:55001/plone/@roles/Reviewer",
    "@type": "role",
    "id": "Reviewer"
  },
  {
    "@id": "http://localhost:55001/plone/@roles/Site Administrator",
    "@type": "role",
    "id": "Site Administrator"
  },
  {
    "@id": "http://localhost:55001/plone/@roles/Manager",
    "@type": "role",
    "id": "Manager"
  }
]
```


Components

Warning: The @components endpoint is deprecated and will be removed in plone.restapi 1.0b1. *Breadcrumbs* and *Navigation* are now top-level endpoints.

How to get pages components (i.e. everything but the main content), like breadcrumbs, navigations, actions, etc.

Breadcrumbs

Getting the breadcrumbs for the current page: http

```
GET /plone/front-page/@components/breadcrumbs HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/front-page/@components/breadcrumbs -H "Accept:␣
↪application/json" --user admin:secret
```

httpie

```
http -j http://nohost/plone/front-page/@components/breadcrumbs -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/front-page/@components/breadcrumbs', headers={
↪'Accept': 'application/json'}, auth=('admin', 'secret'))
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "@id": "http://localhost:55001/plone/front-page/@components/breadcrumbs",
    "items": [
      {
        "title": "Welcome to Plone",
        "url": "http://localhost:55001/plone/front-page"
      }
    ]
  }
]
```

Navigation

Getting the top navigation items: http

```
GET /plone/front-page/@components/navigation HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/front-page/@components/navigation -H "Accept: application/
↪json" --user admin:secret
```

httpie

```
http -j http://nohost/plone/front-page/@components/navigation -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/front-page/@components/navigation', headers={'Accept
↪': 'application/json'}, auth=('admin', 'secret'))
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "@id": "http://localhost:55001/plone/front-page/@components/navigation",
    "items": [
      {
```

```
        "title": "Home",
        "url": "http://localhost:55001/plone"
      },
      {
        "title": "Welcome to Plone",
        "url": "http://localhost:55001/plone/front-page"
      }
    ]
  }
]
```

Breadcrumbs

Getting the breadcrumbs for the current page: http

```
GET /plone/front-page/@components/breadcrumbs HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/front-page/@components/breadcrumbs -H "Accept:␣
↪application/json" --user admin:secret
```

httpie

```
http -j http://nohost/plone/front-page/@components/breadcrumbs -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/front-page/@components/breadcrumbs', headers={
↪'Accept': 'application/json'}, auth=('admin', 'secret'))
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "@id": "http://localhost:55001/plone/front-page/@components/breadcrumbs",
    "items": [
      {
        "title": "Welcome to Plone",
        "url": "http://localhost:55001/plone/front-page"
      }
    ]
  }
]
```

Navigation

Getting the top navigation items: http

```
GET /plone/front-page/@components/navigation HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/front-page/@components/navigation -H "Accept: application/
↪json" --user admin:secret
```

httpie

```
http -j http://nohost/plone/front-page/@components/navigation -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/front-page/@components/navigation', headers={'Accept
↪': 'application/json'}, auth=('admin', 'secret'))
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "@id": "http://localhost:55001/plone/front-page/@components/navigation",
    "items": [
      {
        "title": "Home",
        "url": "http://localhost:55001/plone"
      },
      {
        "title": "Welcome to Plone",
        "url": "http://localhost:55001/plone/front-page"
      }
    ]
  }
]
```

Serialization

Throughout the REST API, content needs to be serialized and deserialized to and from JSON representations.

In general, the format used for serializing content when reading from the API is the same as is used to submit content to the API for writing.

Basic Types

Basic Python data types that have a corresponding type in JSON, like integers or strings, will simply be translated between the Python type and the respective JSON type.

Dates and Times

Since JSON doesn't have native support for dates/times, the Python/Zope datetime types will be serialized to an ISO 8601 datestring.

Python	JSON
`time(19, 45, 55)`	`'19:45:55'`
`date(2015, 11, 23)`	`'2015-11-23'`
`datetime(2015, 11, 23, 19, 45, 55)`	`'2015-11-23T19:45:55'`
`DateTime('2015/11/23 19:45:55')`	`'2015-11-23T19:45:55'`

RichText fields

RichText fields will be serialized as follows:

A `RichTextValue` like

```
RichTextValue(u'<p>Hallöchen</p>',
              mimeType='text/html',
              outputMimeType='text/html')
```

will be serialized to

```
{
  "data": "<p>Hall\u00f6chen</p>",
  "content-type": "text/html",
  "encoding": "utf-8"
}
```

File / Image Fields

Download (serialization)

For download, a file field will be serialized to a mapping that contains the file's most basic metadata, and a hyperlink that the client can follow to download the file:

```
{
  "...": "",
  "@type": "File",
  "title": "My file",
  "file": {
    "content-type": "application/pdf",
    "download": "http://localhost:55001/plone/file/@@download/file",
    "filename": "file.pdf",
    "size": 74429
  }
}
```

That URL in the `download` property points to the regular Plone download view. The client can send a `GET` request to that URL with an `Accept` header containing the MIME type indicated in the `content-type` property, and will get a response containing the file.

Image fields are serialized in the same way, except that their serialization contains their `width` and `height`, and an additional property `scales` that contains a mapping with the available image scales:

```
{
  "icon": {
    "download": "http://localhost:55001/plone/image/@@images/image/icon",
    "height": 32,
    "width": 24
  },
  "large": {
    "download": "http://localhost:55001/plone/image/@@images/image/large",
    "height": 768,
    "width": 576
  },
  "..." : {}
}
```

Upload (deserialization)

For file or image fields, the client must provide the file's data as a mapping containg the file data and some additional metadata:

- `data` - the base64 encoded contents of the file
- `encoding` - the encoding you used to encode the data, so usually *base64*
- `content-type` - the MIME type of the file
- `filename` - the name of the file, including extension

```
{
  "...": "",
  "@type": "File",
  "title": "My file",
  "file": {
      "data": "TG9yZW0gSXBzdW0uCg==",
      "encoding": "base64",
      "filename": "lorem.txt",
      "content-type": "text/plain"}
}
```

Relations

Serialization

A `RelationValue` will be serialized to a short summary representation of the referenced object:

```
{
  "@id": "http://nohost/plone/doc1",
  "@type": "DXTestDocument",
  "title": "Document 1",
  "description": "Description"
}
```

The `RelationList` containing that reference will be represended as a list in JSON.

Deserialization

In order to set a relation when creating or updating content, you can use one of several ways to specify relations:

Type	Example
UID	`'9b6a4eadb9074dde97d86171bb332ae9'`
IntId	`123456`
Path	`'/plone/doc1'`
URL	`'http://localhost:8080/plone/doc1'`

Search

Content in a Plone site can be searched for by invoking the `/@search` endpoint on any context:

```
GET /plone/@search HTTP/1.1
Accept: application/json
```

A search is **contextual** by default, i.e. it is bound to a specific collection and searches within that collection and any sub-collections.

Since a Plone site is also a collection, we therefore have a global search (by invoking the `/@search` endpoint on the site root) and contextual searches (by invoking that endpoint on any other context) all using the same pattern.

In terms of the resulting catalog query this means that, by default, a search will be constrained by the path to the context it's invoked on, unless you explicitly supply your own `path` query.

Search results are represented similar to collections: http

```
GET /plone/@search?sort_on=path HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@search?sort_on=path -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/@search?sort_on=path -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@search?sort_on=path', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/@search",
  "items": [
    {
      "@id": "http://localhost:55001/plone/front-page",
      "@type": "Document",
      "description": "Congratulations! You have successfully installed Plone.",
```

```
      "review_state": "private",
      "title": "Welcome to Plone"
    }
  ],
  "items_total": 1
}
```

The default representation for search results is a summary that contains only the most basic information. In order to return specific metadata columns, see the documentation of the `metadata_fields` parameter below.

Note: Search results results will be **batched** if the size of the resultset exceeds the batch size. See *Batching* for more details on how to work with batched results.

Query format

Queries and query-wide options (like `sort_on`) are submitted as query string parameters to the `/@search` request:

```
GET /plone/@search?SearchableText=lorem HTTP/1.1
```

This is nearly identical to the way that queries are passed to the Plone `@@search` browser view, with only a few minor differences.

For general information on how to query the Plone catalog, please refer to the Plone Documentation on Querying.

Query options

In case you want to supply query options to a query against a particular index, you'll need to flatten the corresponding query dictionary and use a dotted notation to indicate nesting.

For example, to specify the `depth` query option for a path query, the original query as a Python dictionary would look like this:

```
query = {'path': {'query': '/folder',
                  'depth': 2}}
```

This dictionary will need to be flattened in dotted notation in order to pass it in a query string:

```
GET /plone/@search?path.query=%2Ffolder&path.depth=2 HTTP/1.1
```

Again, this is very similar to how Record Arguments are parsed by ZPublisher, except that you can omit the `:record` suffix.

Data types in queries

Because HTTP query strings contain no information about data types, any query string parameter value ends up as a string in the Zope's request. This means, that for values types that aren't string, these data types need to be reconstructed on the server side in plone.restapi.

For most index types and their query values and query options, plone.restapi can handle this for you. If you pass it `path.query=foo&path.depth=1`, it has the necessary knowledge about the `ExtendedPathIndex`'s options to turn the string `1` for the `depth` argument back into an integer before passing the query on to the catalog.

However, certain index types (a `FieldIndex` for example) may take arbitrary data types as query values. In that case, `plone.restapi` simply can't know what data type to cast your query value to, and you'll need to specify it using ZPublisher type hints:

```
GET /plone/@search?numeric_field=42:int HTTP/1.1
Accept: application/json
```

Please refer to the Documentation on Argument Conversion in ZPublisher for details.

Retrieving additional metadata

By default the results are represented as summaries that only contain the most basic information about the items, like their URL and title. If you need to retrieve additional metadata columns, you can do so by specifying the additional column names in the `metadata_fields` parameter:

```
GET /plone/@search?SearchableText=lorem&metadata_fields=modified HTTP/1.1
Accept: application/json
```

The metadata from those columns then will be included in the results. In order to specify multiple columns, simply repeat the query string parameter once for every column name (the `requests` module will do this automatically for you if you pass it a list of values for a query string parameter).

In order to retrieve all metadata columns that the catalog knows about, use `metadata_fields=_all`.

Retrieving full objects

If the data provided as metadata is not enough, you can retrieve search results as full serialized objects equivalent to what the resource GET request would produce.

You do so by specifying the `fullobjects` parameter:

```
GET /plone/@search?fullobjects&SearchableText=lorem HTTP/1.1
Accept: application/json
```

Warning: Be aware that this might induce performance issues when retrieving a lot of resources. Normally the search just serializes catalog brains, but with full objects we wake up all the returned objects.

TUS resumable upload

plone.restapi supports the TUS Open Protocol for resumable file uploads. There is a *@tus-upload* endpoint to upload a file and a *@tus-replace* endpoint to replace an existing file.

Creating an Upload URL

Note: POST requests to the *@tus-upload* endpoint are allowed on all IFolderish content types (e.g. Folder).

To create a new upload, send a POST request to the *@tus-upload* endpoint. http

```
POST /plone/folder/@tus-upload HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Tus-Resumable: 1.0.0
Upload-Length: 8
Upload-Metadata: filename dGVzdC50eHQ=,content-type dGV4dC9wbGFpbg==
```

curl

```
curl -i -X POST http://nohost/plone/folder/@tus-upload -H "Accept: application/json" -
↪H "Tus-Resumable: 1.0.0" -H "Upload-Length: 8" -H "Upload-Metadata: filename␣
↪dGVzdC50eHQ=,content-type dGV4dC9wbGFpbg==" --user admin:secret
```

httpie

```
http -j POST http://nohost/plone/folder/@tus-upload Tus-Resumable:1.0.0 Upload-
↪Length:8 Upload-Metadata:filename dGVzdC50eHQ=,content-type dGV4dC9wbGFpbg== -a␣
↪admin:secret
```

python-requests

```
requests.post('http://nohost/plone/folder/@tus-upload', headers={'Accept':
↪'application/json', 'Tus-Resumable': '1.0.0', 'Upload-Length': '8', 'Upload-Metadata
↪': 'filename dGVzdC50eHQ=,content-type dGV4dC9wbGFpbg=='}, auth=('admin', 'secret'))
```

The server will return a temporary upload URL in the location header of the response:

```
HTTP/1.1 201 Created
Tus-Resumable: 1.0.0
Location: http://localhost:55001/plone/folder/@tus-upload/
↪032803b64ad746b3ab46d9223ea3d90f
```

The file can then be uploaded in the next step to that temporary URL.

Uploading a File

Note: PATCH requests to the *@tus-upload* endpoint are allowed on all IContentish content types.

Once a temporary upload URL has been created, a client can send a PATCH request to upload a file. The file content should be send in the body of the request:

```
PATCH /plone/folder/@tus-upload/032803b64ad746b3ab46d9223ea3d90f HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Tus-Resumable: 1.0.0
Upload-Offset: 3
Content-Type: application/offset+octet-stream

defgh
```

When just a single file is uploaded at once, the server will respond with a *204: No Content* response after a successful upload. The HTTP location header contains he URL of the newly created content object:

```
HTTP/1.1 204 No Content
Upload-Offset: 8
Location: http://localhost:55001/plone/folder/document-2016-10-21
Tus-Resumable: 1.0.0
```

Partial Upload

TUS allows partial upload of files. A partial file is also uploaded by sending a PATCH request to the temporary URL:

```
PATCH /plone/folder/@tus-upload/032803b64ad746b3ab46d9223ea3d90f HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Tus-Resumable: 1.0.0
Upload-Offset: 0
Content-Type: application/offset+octet-stream

abc
```

The server will also respond with a *204: No content* response. Though, instead of providing the final file URL in the 'location' header, the server provides an updated 'Upload-Offset' value, to tell the client the new offset:

```
HTTP/1.1 204 No Content
Upload-Offset: 3
Tus-Resumable: 1.0.0
```

When the last partial file has been uploaded, the server will contain the final file URL in the 'location' header.

Replacing Existing Files

TUS can also be used to replace an existing file by sending a POST request to the *@tus-replace* endpoint instead.

```
POST /plone/myfile/@tus-replace HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Tus-Resumable: 1.0.0
Upload-Length: 8
Upload-Metadata: filename dGVzdC50eHQ=,content-type dGV4dC9wbGFpbg==
```

The server will respond with a *201: Created* status and return the URL of the temprorary created upload resource in the location header of the response:

```
HTTP/1.1 201 Created
Tus-Resumable: 1.0.0
Location: http://localhost:55001/plone/folder/@tus-upload/
↪032803b64ad746b3ab46d9223ea3d90f
```

The file can then be uploaded to that URL using the PATCH method in the same way as creating a new file:

```
PATCH /plone/myfile/@tus-upload/4e465958b24a46ec8657e6f3be720991 HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Tus-Resumable: 1.0.0
Upload-Offset: 0
Content-Type: application/offset+octet-stream
```

```
abcdefgh
```

The server will respond with a *204: No Content* response and the final file URL in the HTTP location header:

```
HTTP/1.1 204 No Content
Upload-Offset: 8
Location: http://localhost:55001/plone/myfile
Tus-Resumable: 1.0.0
```

Asking for the Current File Offset

To ask the server for the current file offset, the client can send a HEAD request to the upload URL: http

```
HEAD /plone/folder/@tus-upload/032803b64ad746b3ab46d9223ea3d90f HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Tus-Resumable: 1.0.0
```

curl

```
curl -i -X HEAD http://nohost/plone/folder/@tus-upload/
↪032803b64ad746b3ab46d9223ea3d90f -H "Accept: application/json" -H "Tus-Resumable: 1.
↪0.0" --user admin:secret
```

httpie

```
http -j HEAD http://nohost/plone/folder/@tus-upload/032803b64ad746b3ab46d9223ea3d90f␣
↪Tus-Resumable:1.0.0 -a admin:secret
```

python-requests

```
requests.head('http://nohost/plone/folder/@tus-upload/032803b64ad746b3ab46d9223ea3d90f
↪', headers={'Accept': 'application/json', 'Tus-Resumable': '1.0.0'}, auth=('admin',
↪'secret'))
```

The server will respond with a *200: Ok* status and the current file offset in the 'Upload-Offset' header:

```
HTTP/1.1 200 OK
Upload-Offset: 3
Upload-Length: 8
Tus-Resumable: 1.0.0
```

Configuration and Options

The current TUS configuration and a list of supported options can be retrieved sending an OPTIONS request to the *@tus-upload* endpoint: http

```
OPTIONS /plone/folder/@tus-upload HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i -X OPTIONS http://nohost/plone/folder/@tus-upload -H "Accept: application/json
↪" --user admin:secret
```

httpie

```
http -j OPTIONS http://nohost/plone/folder/@tus-upload -a admin:secret
```

python-requests

```
requests.options('http://nohost/plone/folder/@tus-upload', headers={'Accept':
↪'application/json'}, auth=('admin', 'secret'))
```

The server will respond with a *204: No content* status and HTTP headers containing information about the available extentions and the TUS version:

```
HTTP/1.1 204 No Content
Tus-Version: 1.0.0
Tus-Extension: creation,expiration
Tus-Resumable: 1.0.0
```

CORS Configuration

If you use CORS and want to make it work with TUS, you have to make sure the TUS specific HTTP headers are allowed by your CORS policy.

```
<plone:CORSPolicy
  allow_origin="http://localhost"
  allow_methods="DELETE,GET,OPTIONS,PATCH,POST,PUT"
  allow_credentials="true"
  allow_headers="Accept,Authorization,Origin,X-Requested-With,Content-Type,Upload-
↪Length,Upload-Offset,Tus-Resumable,Upload-Metadata"
  expose_headers="Upload-Offset,Location,Upload-Length,Tus-Version,Tus-Resumable,Tus-
↪Max-Size,Tus-Extension,Upload-Metadata"
  max_age="3600"
  />
```

See the plone.rest documentation for more information on how to configure CORS policies.

See http://tus.io/protocols/resumable-upload.html#headers for a list and description of the individual headers.

Temporary Upload Directory

During upload files are stored in a temporary directory that by default is located in the *CLIENT_HOME* directory. If you are using a multi ZEO client setup without session stickiness you *must* configure this to a directory shared by all ZEO clients by setting the *TUS_TMP_FILE_DIR* environment variable. E.g. `TUS_TMP_FILE_DIR=/tmp/tus-uploads`

Vocabularies

Vocabularies are utilities containing a list of values grouped by interest or different Plone features. For example, `plone.app.vocabularies.ReallyUserFriendlyTypes` will return all the content types registered in Plone. The vocabularies return a list of objects with the items `@id`, `title` and `token`.

Note: These docs are generated by code tests, therefore you will see some 'test' contenttypes appear here.

Get all vocabularies

To get a list of all the available content types, you can query using a GET to the @vocabulary endpoint: http

```
GET /plone/@vocabularies HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@vocabularies -H "Accept: application/json" --user␣
↪admin:secret
```

httpie

```
http -j http://nohost/plone/@vocabularies -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@vocabularies', headers={'Accept': 'application/json
↪'}, auth=('admin', 'secret'))
```

The response will include a list with all the dotted names of the available vocabularies in Plone.

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.Skins",
    "title": "plone.app.vocabularies.Skins"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪AvailableContentLanguages",
    "title": "plone.app.vocabularies.AvailableContentLanguages"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.schemaeditor.
↪VocabulariesVocabulary",
    "title": "plone.schemaeditor.VocabulariesVocabulary"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.event.Weekdays",
    "title": "plone.app.event.Weekdays"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.contentrules.events",
    "title": "plone.contentrules.events"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.discussion.
↪vocabularies.CaptchaVocabulary",
```

```
    "title": "plone.app.discussion.vocabularies.CaptchaVocabulary"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.Actions
↪",
    "title": "plone.app.vocabularies.Actions"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ImagesScales",
    "title": "plone.app.vocabularies.ImagesScales"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/wicked.vocabularies.
↪BaseConfigurationsOptions",
    "title": "wicked.vocabularies.BaseConfigurationsOptions"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.content.
↪ValidAddableTypes",
    "title": "plone.app.content.ValidAddableTypes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪CommonTimezones",
    "title": "plone.app.vocabularies.CommonTimezones"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.restapi.testing.context_
↪vocabulary",
    "title": "plone.restapi.testing.context_vocabulary"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪AvailableEditors",
    "title": "plone.app.vocabularies.AvailableEditors"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/cmf.calendar.
↪AvailableEventTypes",
    "title": "cmf.calendar.AvailableEventTypes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪SupportedContentLanguages",
    "title": "plone.app.vocabularies.SupportedContentLanguages"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.Keywords
↪",
    "title": "plone.app.vocabularies.Keywords"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.Groups",
    "title": "plone.app.vocabularies.Groups"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪AvailableTimezones",
```

```
    "title": "plone.app.vocabularies.AvailableTimezones"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.Catalog
↪",
    "title": "plone.app.vocabularies.Catalog"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.discussion.
↪vocabularies.TextTransformVocabulary",
    "title": "plone.app.discussion.vocabularies.TextTransformVocabulary"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/wicked.vocabularies.
↪CacheConfigurationsOptions",
    "title": "wicked.vocabularies.CacheConfigurationsOptions"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.Month",
    "title": "plone.app.vocabularies.Month"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.Weekdays
↪",
    "title": "plone.app.vocabularies.Weekdays"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪WorkflowTransitions",
    "title": "plone.app.vocabularies.WorkflowTransitions"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪WeekdaysShort",
    "title": "plone.app.vocabularies.WeekdaysShort"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.contenttypes.
↪metadatafields",
    "title": "plone.app.contenttypes.metadatafields"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/Behaviors",
    "title": "Behaviors"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/Interfaces",
    "title": "Interfaces"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.event.Timezones",
    "title": "plone.app.event.Timezones"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪WorkflowStates",
    "title": "plone.app.vocabularies.WorkflowStates"
  },
```

```
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.event.EventTypes",
    "title": "plone.app.event.EventTypes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/Fields",
    "title": "Fields"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.contenttypes.
↪migration.atctypes",
    "title": "plone.app.contenttypes.migration.atctypes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪SyndicatableFeedItems",
    "title": "plone.app.vocabularies.SyndicatableFeedItems"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.Users",
    "title": "plone.app.vocabularies.Users"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.event.
↪AvailableTimezones",
    "title": "plone.app.event.AvailableTimezones"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪WeekdaysAbbr",
    "title": "plone.app.vocabularies.WeekdaysAbbr"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪UserFriendlyTypes",
    "title": "plone.app.vocabularies.UserFriendlyTypes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪PortalTypes",
    "title": "plone.app.vocabularies.PortalTypes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.controlpanel.
↪WickedPortalTypes",
    "title": "plone.app.controlpanel.WickedPortalTypes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪SyndicationFeedTypes",
    "title": "plone.app.vocabularies.SyndicationFeedTypes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪Timezones",
    "title": "plone.app.vocabularies.Timezones"
  },
  {
```

```
    "@id": "http://localhost:55001/plone/@vocabularies/plone.formwidget.relations.
↪cmfcontentsearch",
    "title": "plone.formwidget.relations.cmfcontentsearch"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes",
    "title": "plone.app.vocabularies.ReallyUserFriendlyTypes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪AllowableContentTypes",
    "title": "plone.app.vocabularies.AllowableContentTypes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.contenttypes.
↪migration.extendedtypes",
    "title": "plone.app.contenttypes.migration.extendedtypes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪MonthAbbr",
    "title": "plone.app.vocabularies.MonthAbbr"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪Workflows",
    "title": "plone.app.vocabularies.Workflows"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.contenttypes.
↪migration.changed_base_classes",
    "title": "plone.app.contenttypes.migration.changed_base_classes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/Group Ids",
    "title": "Group Ids"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.event.
↪SynchronizationStrategies",
    "title": "plone.app.event.SynchronizationStrategies"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪AllowedContentTypes",
    "title": "plone.app.vocabularies.AllowedContentTypes"
  },
  {
    "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.Roles",
    "title": "plone.app.vocabularies.Roles"
  }
]
```

Get a vocabulary

To get a particular vocabulary, `/@vocabularies` endpoint with the name of the vocabulary, e.g. `/plone/@vocabularies/plone.app.vocabularies.ReallyUserFriendlyTypes`. The endpoint can be used with the site root and content objects. The right way is depending on the implementation of the vocabulary. http

```
GET /plone/@vocabularies/plone.app.vocabularies.ReallyUserFriendlyTypes HTTP/1.1
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
```

curl

```
curl -i http://nohost/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes -H "Accept: application/json" --user admin:secret
```

httpie

```
http -j http://nohost/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes -a admin:secret
```

python-requests

```
requests.get('http://nohost/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes', headers={'Accept': 'application/json'}, auth=('admin',
↪'secret'))
```

The server will respond with a list of terms. The title is pureley for display purposes. The token is what should be send to the server to retrive the value of the term.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes",
  "terms": [
    {
      "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes/Collection",
      "title": "Collection",
      "token": "Collection"
    },
    {
      "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes/Discussion Item",
      "title": "Comment",
      "token": "Discussion Item"
    },
    {
      "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes/DXTestDocument",
      "title": "DX Test Document",
      "token": "DXTestDocument"
    },
    {
      "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes/Event",
      "title": "Event",
```

```
      "token": "Event"
    },
    {
      "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes/File",
      "title": "File",
      "token": "File"
    },
    {
      "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes/Folder",
      "title": "Folder",
      "token": "Folder"
    },
    {
      "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes/Image",
      "title": "Image",
      "token": "Image"
    },
    {
      "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes/Link",
      "title": "Link",
      "token": "Link"
    },
    {
      "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes/News Item",
      "title": "News Item",
      "token": "News Item"
    },
    {
      "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes/Document",
      "title": "Page",
      "token": "Document"
    },
    {
      "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes/ATTestDocument",
      "title": "Test Document",
      "token": "ATTestDocument"
    },
    {
      "@id": "http://localhost:55001/plone/@vocabularies/plone.app.vocabularies.
↪ReallyUserFriendlyTypes/ATTestFolder",
      "title": "Test Folder",
      "token": "ATTestFolder"
    }
  ]
}
```

Customizing the API

Content serialization

Dexterity fields

The API automatically converts all field values to JSON compatible data, whenever possible. However, you might use fields which store data that cannot be automatically converted, or you might want to customize the representation of certain fields.

For extending or changing the serializing of certain dexterity fields you need to register an `IFieldSerializer`-adapter.

Example:

```
from plone.customfield.interfaces import ICustomField
from plone.dexterity.interfaces import IDexterityContent
from plone.restapi.interfaces import IFieldSerializer
from plone.restapi.serializer.converters import json_compatible
from plone.restapi.serializer.dxfields import DefaultFieldSerializer
from zope.component import adapter
from zope.interface import Interface
from zope.interface import implementer


@adapter(ICustomField, IDexterityContent, Interface)
@implementer(IFieldSerializer)
class CustomFieldSerializer(DefaultFieldSerializer):

    def __call__(self):
        value = self.get_value()
        if value is not None:
            # Do custom serializing here, e.g.:
            value = value.output()

        return json_compatible(value)
```

Register the adapter in ZCML:

```
<configure xmlns="http://namespaces.zope.org/zope">

    <adapter factory=".serializer.CustomFieldSerializer" />

</configure>
```

The `json_compatible` function recursively converts the value to JSON compatible data, when possible. When a value cannot be converted, a `TypeError` is raised. It is recommended to pass all values through `json_compatible` in order to validate and convert them.

For customizing a specific field instance, a named `IFieldSerializer` adapter can be registered. The name may either be the full dottedname of the field (`plone.app.dexterity.behaviors.exclfromnav.IExcludeFromNavigation.exclude_from_nav`) or the shortname of the field (`exclude_from_nav`).

Conventions

Nouns vs Verbs

Rule: Use nouns to represent resources.

Do:

```
/my-folder
/@registry
/@types
```

Don't:

```
/createFolder
/deleteDocument
/updateEvent
```

Reason:

RESTful URI should refer to a resource that is a thing (noun) instead of referring to an action (verb) because nouns have properties as verbs do not. The REST architectural principle uses HTTP verbs to interact with resources.

Though, there is an exception to that rule, verbs can be used for specific actions or calculations, .e.g.:

```
/login
/logout
/move-to
/reset-password
```

Singluar vs Plural

Rule: Use plural resources.

Do:

```
/users
/users/21
```

Don't:

```
/user
/user/21
```

Reason:

If you use singular for a collection like resource (e.g. "/user" to retrieve a list of all users) it feels wrong. Mixing singular and plural is confusing (e.g. user "/users" for retrieving users and "/user/21" to retrieve a single user).

Attribute names in URIs

Rule: Use hyphens to improve readability of URIs.

Do:

```
/users/noam/reset-password
```

Don't:

```
/users/noam/resetPassword
/users/noam/ResetPassword
/users/noam/reset_password
```

Reason:

Upper vs. Lowercase

Rule: Use lowercase letters in URIs.

Do:

```
http://example.com/my-folder/my-document
```

Don't:

```
http://example.com/My-Folder/My-Document
```

Reason: RFC 3986 defines URIs as case-sensitive except for the scheme and host components. e.g.

Those two URIs are equivalent:

```
http://example.org/my-folder/my-document
HTTP://EXAMPLE.ORG/my-folder/my-document
```

While this one is not equivalent to the two URIs above:

```
http://example.org/My-Folder/my-document
```

To avoid confusion we always use lowercase letters in URIs.

Versioning

Versioning APIs does make a lot of sense for public API services. Especially if an API provider needs to ship different versions of the API at the same time. Though, Plone already has a way to version packages and it currently does not make sense for us to expose that information via the API. We will always just ship one version of the API at a time and we are usually in full control over the backend and the frontend.

CHAPTER 2

Introduction

plone.restapi is a RESTful hypermedia API for Plone.

RESTful Hypermedia API

REST stands for Representational State Transfer. It is a software architectural principle to create loosely coupled web APIs.

Most web APIs have a tight coupling between client and server. This makes them brittle and hard to change over time. It requires them not only to fully document every small detail of the API, but also to write a client implementation that follows that specification 100% correctly and breaks as soon as you change any detail.

A hypermedia API just provides an entry point to the API that contains hyperlinks the clients can follow, just as a human user of a regular website knows the initial URL of the site and then follows hyperlinks to navigate through the site. This has the advantage that the client needs to understand only how to detect and follow links. The URL and other details of the API can change without breaking the client.

CHAPTER 3

Documentation

http://plonerestapi.readthedocs.org

CHAPTER 4

Roadmap

https://github.com/plone/plone.restapi/milestones

CHAPTER 5

Live Demo

A live demo of Plone 5 with the latest plone.restapi release is available at:

http://plonedemo.kitconcept.com

Example GET request on the portal root:

```
$ curl -i http://plonedemo.kitconcept.com -H "Accept: application/json"
```

Example POST request to create a new document:

```
$ curl -i -X POST http://plonedemo.kitconcept.com -H "Accept: application/json" -H
↪"Content-Type: application/json" --data-raw '{"@type": "Document", "title": "My␣
↪Document"}' --user admin:admin
```

Note: You will need some kind of API browser application to explore the API. We recommend using Postman.

CHAPTER 6

Design Decisions

- A truly RESTful API (Hypermedia / HATEOAS / Linked-data)
- JSON is the main target format; support for other formats (HTML, XML) will come later
- Use HTTP headers (to set format and versioning, also provide URL-based option to make it easier for people to try it out)
- No versioning; versioning in the HTTP header can be added later
- Field names just map over (we will not try to clean up attributes or enforce naming standards like pep8 (e.g. isPrincipiaFoldish -> is_folderish)

CHAPTER 7

Software Quality

- 100% test coverage
- 100% PEP8 compliant
- Documentation-first approach for enhancements

CHAPTER 8

Further Reading

- REST in Practice: Hypermedia and Systems Architecture (Webber, Parastatidis, Robinson)

CHAPTER 9

Standards

- JSON-LD
- JSON Schema
- Schema.org
- Hydra
- Collection+JSON
- Siren

CHAPTER 10

License

The project is licensed under the GPLv2.

Appendix, Indices and tables

HTTP Status Codes

This is the list of status codes that are used in plone.restapi. Here is a full list of all HTTP status codes.

200 OK Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request, the response will contain an entity describing or containing the result of the action.

201 Created The request has been fulfilled and resulted in a new resource being created.

204 No Content The server successfully processed the request, but is not returning any content. Usually used as a response to a successful delete request.

2xx Success This class of status codes indicates the action requested by the client was received, understood, accepted and processed successfully.

400 Bad Request The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing)

401 Unauthorized Similar to 403 Forbidden, but specifically for use when authentication is required and has failed or has not yet been provided. The response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource.

403 Forbidden The request was a valid request, but the server is refusing to respond to it. Unlike a 401 Unauthorized response, authenticating will make no difference.

404 Not Found The requested resource could not be found but may be available again in the future. Subsequent requests by the client are permissible.

405 Method Not Allowed A request method is not supported for the requested resource; for example, a GET request on a form which requires data to be presented via POST, or a PUT request on a read-only resource.

409 Conflict Indicates that the request could not be processed because of conflict in the request, such as an edit conflict in the case of multiple updates.

4xx Client Error The 4xx class of status code is intended for cases in which the client seems to have errored.

500 Internal Server Error The server failed to fulfill an apparently valid request.

5xx Server Error The server failed to fulfill an apparently valid request.

Glossary

Accept Header Part of the *Request* that is responsible to define the expected type of data to be accepted by the client in the *Response*.

Authentication Method Access restriction provided by the connection chain to the server exposed to the client.

Authorization Header Part of the *Request* that is responsible for the authentication related to the right user or service to ask for a *Response*.

Basic Auth A simple *Authentication Method* referenced in the *Authorization Header* that needs to be provided by the server.

Client

Server

Network

Socket

HTTP-Header

HTTP Header

Header The part of the communication of the client with the server that provides the initialisation of the communication of a *Request*.

HTTP-Request

HTTP Request

Request

Requests The initial action performed by a web client to communicate with a server. The *Request* is usually followed by a *Response* by the server, either synchronous or asynchronous (which is more complex to handle on the user side)

HTTP-Response

HTTP Response

Response Answer of or action by the server that is executed or send to the client after the *Request* is processed.

HTTP-Verb

HTTP Verb

Verb One of the basic actions that can be requested to be executed by the server (on an object) based on the *Request*.

Object URL The target object of the *Request*

REST REST stands for Representational State Transfer. It is a software architectural principle to create loosely coupled web APIs.

workflow A concept in Plone (and other CMS's) whereby a content object can be in a number of states (private, public, etcetera) and uses transitions to change between them (e.g. "publish", "approve", "reject", "retract"). See the Plone docs on Workflow

- genindex

Index