plone.jsonapi.routes Documentation

Release 0.9.3

Ramon Bartl

May 14, 2017

Contents

Welcome to *plone.jsonapi.routes* documentation. This documentation is divided in different parts. I recommend that you get started with *Installation* and then head over to the *Quickstart*. Please check out the *API* documentation for internals about *plone.jsonapi.routes*.

Table of Contents:

CHAPTER 1

Installation

This package depends on two external libraries, plone.jsonapi.core and plone.api. The first one is the core framework which takes care of the dispatching of requests and route registration. The latter one is a simplified API to the Plone CMS.

Buildout

The simplest way to install plone.jsonapi.routes is to add it to the buildout configuration:

```
[buildout]

...

[instance]
...
eggs =
    ...
    plone.jsonapi.routes
```

Run the buildout and your Plone site will become RESTful.

The routes for the standard Plone content types get registered on startup. The following URL should be available after startup:

http://localhost:8080/Plone/@@API/plone/api/1.0

JSONView

Use the JSONView Plugin for your browser to view the generated JSON in a more comfortable way. You can find the extensions here:

- Firefox: https://addons.mozilla.org/de/firefox/addon/jsonview

- Chrome: https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc?hl=en

Advanced Rest Client

Use this Chrome Plugin to send POST request to the Plone JSON API. You can find it here:

https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo

CHAPTER 2

Quickstart

This section gives a good introduction about plone.jsonapi.routes. It assumes you already have Plone and plone.jsonapi.routes installed. Since all the coming examples are executed directly in Google Chrome, it assumes that you have also installed JSONView and the Advanced Rest Client Application (see *Installation* for details)

Environment

The Plone site is hosted on *http://localhost:8080/Plone*. The JSON API is therefore located at *http://localhost:8080/Plone/@@API/plone/api/1.0*. Make sure your Plone site is located on the same URL, so you can directly click on the links within the exapmles.

Version

The *version* route prints out the current version of *plone.jsonapi.routes*.

http://localhost:8080/Plone/@@API/plone/api/1.0/version

```
{
    url: "http://localhost:8080/Plone/@@API/plone/api/1.0/version",
    date: "2017-05-12",
    version: "0.9.2",
    _runtime: 0.0001528865814208984
}
```

Note: The runtime indicates the time spent in milliseconds until the response is prepared.

Content Routes

These *Resources* are automatically generated for **all** available content types in Plone.

Each content route is located at the *Base URL*, e.g.

- http://localhost:8080/Plone/@@API/plone/api/1.0/folder
- http://localhost:8080/Plone/@@API/plone/api/1.0/image
- http://localhost:8080/Plone/@@API/plone/api/1.0/file
- http://localhost:8080/Plone/@@API/plone/api/1.0/document
- http://localhost:8080/Plone/@@API/plone/api/1.0/collection
- http://localhost:8080/Plone/@@API/plone/api/1.0/<myportaltype>

The name of each of these content routes is transformed to lower case, so it is also perfectly ok to call these *Resources* like so:

- http://localhost:8080/Plone/@@API/plone/api/1.0/Folder
- http://localhost:8080/Plone/@@API/plone/api/1.0/Image
- http://localhost:8080/Plone/@@API/plone/api/1.0/File
- http://localhost:8080/Plone/@@API/plone/api/1.0/Document
- http://localhost:8080/Plone/@@API/plone/api/1.0/Collection
- http://localhost:8080/Plone/@@API/plone/api/1.0/<MyPortalType>

Calling a content route like this will return a JSON format similar to this:

```
{
    count: 14,
    pagesize: 25,
    items: [
        {},
        {},
        {},
        {},
        ...
    ],
    page: 1,
    _runtime: 0.0038590431213378906,
    next: null,
    pages: 1,
    previous: null
}
```

The *Response Format* in *plone.jsonapi.routes* content URLs is always the same. The top level keys (data after the first `{}`) are meta informations about the gathered data.

The *items* list will contain the list of catalog results for the portal type requested. This means that each result contains just the metadata available in the catalog. Therfore, no object is “waked up” to retrieve the data at this stage. This is because of the APIs **two step** concept, which postpones expensive opreations, until the user really wants it.

All *items* are batched to increase performance of the API. The *count* number returns the total number objects found, while the *page* number returns the number of pages in the batch, which can be navigated with the *next* and *previous* links.

New in version 0.3: The result is now always batched. This means you get the items split up into batches onto multiple sites.

Getting the Full Data

To get all data from an object, you can either add the `complete=True` parameter, or you can request the data with the object `UID`.

- http://localhost:8080/Plone/@@API/plone/api/1.0/folder?complete=True
- http://localhost:8080/Plone/@@API/plone/api/1.0/image/<uid>
- http://localhost:8080/Plone/@@API/plone/api/1.0/<uid>

The requested content(s) is now loaded by the API and all fields were gathered.

Note: Please keep in mind that large data sets with the *?complete=True* Parameter might increase the loading time significantly.

Special Case: Files and Images

- http://localhost:8080/Plone/@@API/plone/api/1.0/file
- http://localhost:8080/Plone/@@API/plone/api/1.0/image

New in version 0.2: The object data contains now the base64 encoded file with the size and mimetype information.

New in version 0.7: You can pass in a *filename* in the JSON body to set the name of the file created. If omitted, the id or title will be used.

New in version 0.8: You can pass in a *mimetype* key to manually set the content type of the file. If omitted, the content type will be guessed by the filename. Default: *application/octet-stream*

New in version 0.8: The response data contains now the *filename* and the *download* url.

To create a new file in the portal, you have to HTTP POST to the *create* route:

http://localhost:8080/Plone/@@API/plone/api/1.0/file/create

The HTTP POST payload can look like this:

```
{
    "title": "Test.docx",
    "description": "A Word File",
    "filename": "test.docx",
    "parent_path": "/Plone/folder",
    "file":"UEsDBBQABgAIAAA..."
}
```

The *file* key in the HTTP POST payload contains the *base64* encoded content of the file/image.

UID Route

To fetch the full data of an object immediately, it is also possible to append the UID of the object directly on the root URL of the API, e.g.:

- http://localhost:8080/Plone/@@API/plone/api/1.0/553ce5b2c55847a08dea2a7016a0e11a
- http://localhost:8080/Plone/@@API/plone/api/1.0/document/c79c878703194ee78944c36dedd7b26d

Note: The given UID might seem different on your machine.

The response will give the data in the root of the JSON data, e.g.:

```
{
    "uid": "553ce5b2c55847a08dea2a7016a0e11a",
    "contributors": [],
    "file": "http://localhost:8080/Plone/w7614.pdf/@@download/file/w7614.pdf",
    "_runtime": 0.010680913925170898,
    "exclude_from_nav": false,
    "id": "w7614.pdf",
    "api_url": "http://localhost:8080/Plone/@@API/plone/api/1.0/file/
↪553ce5b2c55847a08dea2a7016a0e11a",
    "title": "w7614.pdf",
    "parent_id": "Plone",
    "subjects": null,
    "author": "admin",
    "parent_url": "http://localhost:8080/Plone/@@API/plone/api/1.0/plonesite/0",
    "description": "",
    "tags": [],
    "portal_type": "File",
    "expires": null,
    "relatedItems": [],
    "parent_uid": "0",
    "effective": null,
    "language": "",
    "rights": "",
    "url": "http://localhost:8080/Plone/w7614.pdf",
    "created": "2017-05-12T12:47:38+02:00",
    "modified": "2017-05-12T12:47:38+02:00",
    "allow_discussion": null,
    "creators": [
        "admin"
    ]
}
```

CHAPTER 3

API

This part of the documentation covers all resources (routes) provided by plone.jsonapi.routes. It also covers all the request parameters that can be applied to these resources to refine the results.

Concept

The Plone JSON API aims to be **as fast as possible**. So the concept of the API is to postpone **expensive operations** until the user really requests it. To do so, the API was built with a **two step architecture**.

An **expensive operation** is basically given, when the API needs to “wake up” an object to retrieve all its field values. This means the full object has to be loaded from the Database (ZODB) into the memory (RAM).

The **two step architecture** retrieves only the fields of the catalog results in the *first step*. Only if the user requests the API URL of a specific object, the object will be loaded and all the fields of the object will be returned.

Note: Since version 0.3, you can add a *complete=yes* paramter to bypass the two step behavior and retrieve the full object data immediately.

Base URL

After installation, the Plone API routes are available below the plone.jsonapi.core root URL (`@@API`) with the base `/plone/api/1.0`.

Example: `http://localhost:8080/Plone/@@API/plone/api/1.0/api.json`

Note: Please see the documentation of plone.jsonapi.core for the root URL.

There is also an overview of the registered routes, e.g.

```
http://localhost:8080/Plone/@@API/plone/api/1.0/api.json
```

Resources

URL Schema `<BASE URL>/<RESOURCE>/<OPERATION>/<uid:optional>`

A resource is equivalent with the portal type name in Plone.

This means that all portal types are fully supported by the API simply by adding the portal type to the end of the base url, e.g.:

- http://localhost:8080/Plone/@@API/plone/api/1.0/Folder
- http://localhost:8080/Plone/@@API/plone/api/1.0/Image
- http://localhost:8080/Plone/@@API/plone/api/1.0/File

Note: Lower case portal type names are also supported.

Operations

The API understands the basic CRUD operations on the *content resources*. Only the VIEW operation is accessible via a HTTP GET request. All other operations have to be sent via a HTTP POST request.

OPERATION	URL	METHOD
VIEW	<BASE URL>/<RESOURCE>/<uid:optional>	GET
CREATE	<BASE URL>/<RESOURCE>/create/<uid:optional>	POST
UPDATE	<BASE URL>/<RESOURCE>/update/<uid:optional>	POST
DELETE	<BASE URL>/<RESOURCE>/delete/<uid:optional>	POST

New in version 0.9.1.

The API is now fully aware of all registered portal types in Plone. Each resource is now equivalent to the portal type name.

It is also possible now to get all contents by UID directly from the base url, e.g.: http://localhost:8080/Plone/@@API/plone/api/1.0/<uid>

Search Resource

The search route omits the portal type and is therefore capable to search for **any** content type within the portal.

The search route accepts all available indexes which are defined in the portal catalog tool, e.g.:

- http://localhost:8080/Plone/@@API/plone/api/1.0/search

Returns **all** contents, which were indexed by the catalog.

- http://localhost:8080/Plone/@@API/plone/api/1.0/search?id=test

Returns contents which match the given value of the *id* parameter.

User Resource

The API is capable to find Plone users, e.g.:

- http://localhost:8080/Plone/@@API/plone/api/1.0/users
- http://localhost:8080/Plone/@@API/plone/api/1.0/users/current
- http://localhost:8080/Plone/@@API/plone/api/1.0/users/<username>

```
{
    "count": 1,
    "pagesize": 25,
    "items": [
        {
            "username": "ramon",
            "visible_ids": false,
            "authenticated": false,
            "api_url": "http://localhost:8080/Plone/@@API/plone/api/1.0/users/ramon",
            "roles": [
                "Member",
                "Authenticated"
            ],
            "home_page": "",
            "description": "",
            "wysiwyg_editor": "",
            "location": "",
            "error_log_update": 0,
            "language": "",
            "listed": true,
            "groups": [
                "AuthenticatedUsers"
            ],
            "portal_skin": "",
            "fullname": "Ramon Bartl",
            "login_time": "2000-01-01T00:00:00",
            "email": "rb@ridingbytes.com",
            "ext_editor": false,
            "last_login_time": "2000-01-01T00:00:00"
        }
    ],
    "page": 1,
    "_runtime": 0.008383989334106445,
    "next": null,
    "pages": 1,
    "previous": null
}
```

The results come as well as batches of 25 items per default. It is also possible to get a higher or lower number of users per batch with the *?limit=n* request parameter, e.g.:

http://localhost:8080/Plone/@@API/plone/api/1.0/users?limit=1

Note: This route lists all users for **authenticated** users only.

The username *current* is reserved to fech the current logged in user:

http://localhost:8080/Plone/@@API/plone/api/1.0/users/current

Overview

Resource	Action	Description
users	<username>,current	Resource for Plone Users
auth		Basic Authentication
login		Login with __ac_name and __ac_password
logout		Deauthenticate

Parameters

URL Schema `<BASE URL>/<RESOURCE>?<KEY>=<VALUE>&<KEY>=<VALUE>`

All content resources accept to be filtered by request parameters.

Key	Value	Description
q	searchterm	Search the SearchableText index for the given query string
path	/physical/path	Specifiy a physical path to only return results below it. See how to Query by path in the Plone docs for details.
depth	0..n	Specify the depth of a path query. Only relevant when using the path parameter.
limit	1..n	Limit the results to the given *limit* number. This will return batched results with *x* pages and *n* items per page
sort_on	catalog index	Sort the results by the given index
sort_order	asc / desc	Sort ascending or descending (default: ascending)
sort_limit	1..n	Limit the result set to n items. The portal catalog will only return n items.
complete	yes/y/1/True	Flag to return the full object results immediately. Bypasses the *two step* behavior of the API
children	yes/y/1/True	Flag to return the folder contents of a folder below the *children* key Only visible if complete flag is true or if an UID is provided
workflow	yes/y/1/True	Flag to include the workflow data below the *workflow* key
filedata	yes/y/1/True	Flag to include the base64 encoded file
recent_created	today, yesterday this-week, this-month this-year	Specify a recent created date range, to find all items created within this date range until today. This uses internally *'range': 'min'* query.
recent_modified	today, yesterday this-week, this-month this-year	Specify a recent modified date range, to find all items modified within this date range until today. This uses internally *'range': 'min'* query.
sharing	yes/y/1/True	Flag to include the sharing rights. Only visible if complete flag is true.

Using Plone Indexes

It is also possible to use the Plone catalog indexes directly as request parameters.

New in version 0.4: Support for DateIndex, KeywordIndex and BooleanIndex. Support for 'recent_modified' and 'recent_created' literals.

Note: Custom added indexes can also be used, as long as they accept a single string value as query.

Query Records

It is also possible to use the ZPublisher query record format.

Example

`http://localhost:8080/Plone/@@API/plone/api/1.0/folders?created.query:record:list:date=2015-01-02&created.range:record=min`

New in version 0.5: Support for ZPublisher query record format added.

Sharing

It is also possible to check the sharing settings for objects

Example:

`http://localhost:8080/Plone/@@API/plone/api/1.0/folders/<uid:required>?sharing=y` `http://localhost:8080/Plone/@@API/plone/api/1.0/folders?sharing=y&complete=y`

Response:

```
{
    "sharing": {
        "inherit": false,
        "role_settings": [
            {
                "disabled": false,
                "id": "AuthenticatedUsers",
                "login": null,
                "roles": {
                    "Contributor": false,
                    "Editor": false,
                    "Reader": false,
                    "Reviewer": false
                },
                "title": "Logged-in users",
                "type": "group"
            }
        ]
    }
}
```

Update inherit role settings:

`http://localhost:8080/Plone/@@API/plone/api/1.0/update/<uid:required>`

```
{
    "sharing": {
        "inherit": false,
        "role_settings": [
            {
                "disabled": false,
                "id": "AuthenticatedUsers",
                "login": null,
                "roles": {
                    "Contributor": false,
                    "Editor": false,
                    "Reader": false,
                    "Reviewer": false
                },
                "title": "Logged-in users",
                "type": "group"
            }
        ]
    }
}
```

Note: You can pass in the same format as you got from the API

New in version 0.8.4: Support sharing permissions of objects

Response Format

The response format is for all resources the same.

```
{
    count: 1, // number of found items
    pagesize: 25, // items per page
    items: [  // List of all item objexts
        {
            id: "front-page", // item data
            ...
        }
    ],
    page: 1, // current page
    _runtime: 0.00381,  // calculation time to generate the data
    next: null,  // URL to the next batch
    pages: 1,  //  number of total pages
    previous: null  // URL to the previous batch
}
```

count The number of found items – can be more than displayed on one site

pagesize Number of items per page

items List of found items – only catalog brain keys unless you add a *complete=yes* parameter to the request or request an URL with an UID at the end.

page The current page of the batched result set

_runtime The time in milliseconds needed to generate the data

next The URL to the next batch

pages The number of pages in the batch

previous The URL to the previous batch

The API Module

import *from plone.jsonapi.routes import api*

doc Provides core functionality to all other modules

CHAPTER 4

Authentication

Since version 0.8 (see: Changelog) the API provides a simple way to authenticate a user with Plone.

Login

URL Schema `<BASE URL>/login?__ac_name=<username>&__ac_password=<password>`

The response will set the *__ac* cookie for further cookie authenticated requests.

Note: Currently only cookie authentication works. Other PAS plugins might not work as expected.

Example

`http://localhost:8080/Plone/@@API/plone/api/1.0/login?__ac_name=admin&__ac_password=admin`

Response

```
{
    url: "http://localhost:8080/Plone/@@API/plone/api/1.0/users",
    count: 1,
    _runtime: 0.0019960403442382812,
    items: [
        {
            username: "admin",
            authenticated: true,
            last_login_time: "",
            roles: [
                "Manager",
                "Authenticated"
            ],
            url: "http://localhost:8080/Plone/@@API/plone/api/1.0/users/admin",
            email: null,
            groups: [ ],
            fullname: null,
            id: "admin",
            login_time: ""
        }
    ]
}
```

Logout

URL Schema `<BASE URL>/logout`

The response will expire the *__ac* cookie for further requests.

Example

`http://localhost:8080/Plone/@@API/plone/api/1.0/logout`

Response

```
{
    url: "http://localhost:8080/Plone/@@API/plone/api/1.0/users",
    _runtime: 0.0009028911590576172,
    success: true
}
```

Basic Authentication

URL Schema `<BASE URL>/auth`

If the reqeust is not authenticated, this route will raise an unauthorized response with status code 401. Browsers should display the Basic Authentication login. Example

`http://localhost:8080/Plone/@@API/plone/api/1.0/auth`

CHAPTER 5

Customizing

This package is built to be extended. You can either use the *Zope Component Architecture* and provide an specific Adapter to control what is being returned by the API or you simply write your own route provider.

This section will show how to build a custom route provider for an example content type. It will also show how to write and register a custom data adapter for this content type. It is even possible to customize how the fields of a specific content type can be accessed or modified.

Adding a custom route provider

Each route provider shipped with this package, provides the basic CRUD functionality to *get*, *create*, *delete* and *update* the resource handled.

The same functionality can be used to provide this behavior for custom content types. All neccessary functions are located in the *api* module within this package.

```python
# CRUD
from plone.jsonapi.routes.api import get_batched
from plone.jsonapi.routes.api import create_items
from plone.jsonapi.routes.api import update_items
from plone.jsonapi.routes.api import delete_items

# route dispatcher
from plone.jsonapi.core.browser.router import add_route

# GET
@add_route("/todos", "todos", methods=["GET"])
@add_route("/todos/<string:uid>", "todos", methods=["GET"])
def get(context, request, uid=None):
    """ get all todos
    """
    return get_batched("Todo", uid=uid, endpoint="todo")
```

New in version 0.3: The standard GET route returns now the results for all resoures batched. This behavior is provided by the *get_batched* function.

Note: The prior *get_items* function, which returns all items in an array, is still provided, but not recommended due to performance issues.

New in version 0.9.0: You can specify an own *query* and pass it to the *get_batched* or *get_items* funciton of the api. This gives full control over the executed query on the catalog. Please see the *docs/Readme.rst* doctest for more details.

Note: Other keywords (except *uid*) are ignored, if the *query* keyword is detected.

The upper example registers a function named *get* with the *add_route* decorator. This ensures that this function gets called when the */todos* route is called, e.g. *http://localhost:8080/Plone/@@API/todo*.

The second argument of the decorator is the endpoint, which is kind of the registration key for our function. The last argument is the methods we would like to handle here. In this case we're only interested in GET requests.

All route providers get always the *context* and the *request* as the first two arguments. The *uid* keyword argument is passed in, when a UID was appended to the URL, e.g *http://localhost:8080/Plone/@@API/todo/a3f3f9efd0b4df190d16ea63d*.

The *get_batched* function we call inside our function will do all the heavy lifting for us. We simply need to pass in the *portal_type* as the first argument, the *UID* and the *endpoint*.

To be able to create, update and delete our *Todo* content type, it is neccessary to provide the following functions as well. The behavior is analogue to the upper example but as there is no need for batching, the functions return a Python <*list*> instead of a complete mapping as above.

```
ACTIONS = "create,update,delete,cut,copy,paste"

# http://werkzeug.pocoo.org/docs/0.11/routing/#builtin-converters
# http://werkzeug.pocoo.org/docs/0.11/routing/#custom-converters
@route("/<any(" + ACTIONS + "):action>",
      "plone.jsonapi.routes.action", methods=["POST"])
@route("/<any(" + ACTIONS + "):action>/<string(maxlength=32):uid>",
      "plone.jsonapi.routes.action", methods=["POST"])
@route("/<string:resource>/<any(" + ACTIONS + "):action>",
      "plone.jsonapi.routes.action", methods=["POST"])
@route("/<string:resource>/<any(" + ACTIONS + "):action>/<string(maxlength=32):uid>",
      "plone.jsonapi.routes.action", methods=["POST"])
def action(context, request, action=None, resource=None, uid=None):
    """Various HTTP POST actions

    Case 1: <action>
    <Plonesite>/@@API/plone/api/1.0/<action>

    Case 2: <action>/<uid>
    -> The actions (cut, copy, update, delete) will performed on the object␣
↪identified by <uid>
    -> The actions (create, paste) will use the <uid> as the parent folder
    <Plonesite>/@@API/plone/api/1.0/<action>/<uid>

    Case 3: <resource>/<action>
    -> The "target" object will be located by a location given in the request body␣
↪(uid, path, parent_path + id)
    -> The actions (cut, copy, update, delete) will performed on the target object
    -> The actions (create) will use the target object as the container
    <Plonesite>/@@API/plone/api/1.0/<resource>/<action>

    Case 4: <resource>/<action>/<uid>
    -> The actions (cut, copy, update, delete) will performed on the object␣
↪identified by <uid>
    -> The actions (create) will use the <uid> as the parent folder
    <Plonesite>/@@API/plone/api/1.0/<resource>/<action>
    """

    # Fetch and call the action function of the API
    func_name = "{}_items".format(action)
    action_func = getattr(api, func_name, None)
    if action_func is None:
        api.fail(500, "API has no member named '{}'".format(func_name))

    portal_type = api.resource_to_portal_type(resource)
    items = action_func(portal_type=portal_type, uid=uid)

    return {
        "count": len(items),
        "items": items,
        "url": api.url_for("plone.jsonapi.routes.action", action=action),
    }
```

Adding a custom data adapter

The data returned by the API for each content type is extracted by the *IInfo* Adapter. This Adapter simply extracts all field values from the content.

To customize how the data is extracted from the content, you have to register an adapter for a more specific interface on the content.

This adapter has to implement the *IInfo* interface.

```
from plone.jsonapi.routes.interfaces import IInfo

class TodoAdapter(object):
    """ A custom adapter for Todo content types
    """
    interface.implements(IInfo)

    def __init__(self, context):
        self.context = context

    def to_dict(self):
        return {} # whatever data you need

    def __call__(self):
        # just implement it like this, don't ask x_X
        return self.to_dict()
```

Register the adapter in your *configure.zcml* file for your special interface:

```
<configure
    xmlns="http://namespaces.zope.org/zope">
    <!-- Adapter for my custom content type -->
    <adapter
        for="plone.todo.interfaces.ITodo"
        factory=".adapters.TodoAdapter"
        />

</configure>
```

Adding a custom data manager

The data sent by the API for **each content type** is set by the *IDataManager* Adapter. This Adapter has a simple interface:

```
class IDataManager(interface.Interface):
    """ Field Interface
    """

    def get(name):
        """ Get the value of the named field with
        """

    def set(name, value):
        """ Set the value of the named field
        """

    def json_data(name, default=None):
        """ Get a JSON compatible structure from the value
        """
```

To customize how the data is set to each field of the content, you have to register an adapter for a more specific interface on the content. This adapter has to implement the *IDataManager* interface.

Note: The *json_data* function is called by the Data Provider Adapter (*IInfo*) to get a JSON compatible return Value, e.g.: DateTime('2017/05/14 14:46:18.746800 GMT+2') -> "2017-05-14T14:46:18+02:00"

Important: Please be aware that you have to implement security for field level access on your own.

```
from zope.annotation import IAnnotations
from persistent.dict import PersistentDict
from plone.jsonapi.routes.interfaces import IDataManager

class TodoDataManager(object):
    """ A custom data manager for Todo content types
    """
    interface.implements(IDataManager)

    def __init__(self, context):
        self.context = context

    @property
    def storage(self):
        return IAnnotations(self.context).setdefault('plone.todo', PersistentDict())

    def get(self, name):
        self.storage.get("name")

    def set(self, name, value):
        self.storage["name"] = value
```

Register the adapter in your *configure.zcml* file for your special interface:

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

    <!-- Adapter for my custom content type -->
    <adapter
        for="plone.todo.interfaces.ITodo"
        factory=".adapters.TodoDataManager"
        />

</configure>
```

Adding a custom field manager

The default data managers (*IDataManager*) defined in this package know how to *set* and *get* the values from fields. But sometimes it might be useful to be more granular and know how to *set* and *get* a value for a **specific field**.

Therefore, *plone.jsonapi.routes* introduces Field Managers (*IFieldManager*), which adapt a field.

This Adapter has a simple interface:

```
class IFieldManager(interface.Interface):
    """A Field Manager is able to set/get the values of a single field.
    """

    def get(instance, **kwargs):
        """Get the value of the field
        """

    def set(instance, value, **kwargs):
        """Set the value of the field
        """

    def json_data(instance, default=None):
        """Get a JSON compatible structure from the value
        """
```

To customize how the data is set to each field of the content, you have to register a more specific adapter to a field.

This adapter has to implement then the *IFieldManager* interface.

Note: The *json_data* function is called by the Data Manager Adapter (*IDataManager*) to get a JSON compatible return Value, e.g.: DateTime('2017/05/14 14:46:18.746800 GMT+2') -> "2017-05-14T14:46:18+02:00"

Note: The *json_data* method is defined on context level (*IDataManger*) as well as on field level (*IFieldManager*). This is to handle objects w/o fields, e.g. Catalog Brains, Portal Object etc. and Objects which contain fields and want to delegate the JSON representation to the field.

Important: Please be aware that you have to implement security for field level access on your own.

```python
class DateTimeFieldManager(ATFieldManager):
    """Adapter to get/set the value of DateTime Fields
    """
    interface.implements(IFieldManager)

    def set(self, instance, value, **kw):
        """Converts the value into a DateTime object before setting.
        """
        try:
            value = DateTime(value)
        except SyntaxError:
            logger.warn("Value '{}' is not a valid DateTime string"
                        .format(value))
            return False

        self._set(instance, value, **kw)

    def json_data(self, instance, default=None):
        """Get a JSON compatible value
        """
        value = self.get(instance)
        return api.to_iso_date(value) or default
```

Register the adapter in your *configure.zcml* file for your special interface:

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

  <!-- Adapter for AT DateTime Field -->
  <adapter
      for="Products.Archetypes.interfaces.field.IDateTimeField"
      factory=".fieldmanagers.DateTimeFieldManager"
      />

</configure>
```

Adding a custom catalog tool

New in version 0.9.1: You can specify an own *catalog* tool which performs your custom query.

All search is done through a catalog adapter. This adapter has to provide at least a *search* method. The others are optional, but recommended.

```python
class ICatalog(interface.Interface):
    """ Plone catalog interface
    """
    def search(query):
        """ search the catalog and return the results
        """

    def get_catalog():
        """ get the used catalog tool
        """

    def get_indexes():
        """ get all indexes managed by this catalog
        """

    def get_index(name):
        """ get an index by name
        """

    def to_index_value(value, index):
        """ Convert the value for a given index
        """
```

To customize the catalog tool to get full control of the search, you have to register an catalog adapter for a more specific interface on the portal. This adapter has to implement the *ICatalog* interface.

```
from zope import interface
from plone.jsonapi.routes.interfaces import ICatalog
from plone.jsonapi.routes import api

class Catalog(object):
    """Plone catalog adapter
    """
    interface.implements(ICatalog)

    def __init__(self, context):
        self._catalog = api.get_tool("portal_catalog")

    def search(self, query):
        """search the catalog
        """
        catalog = self.get_catalog()
        return catalog(query)
```

Register the adapter in your *configure.zcml* file for your special interface:

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

    <!-- Adapter for a custom catalog adapter -->
    <adapter
        for=".interfaces.ICustomPortalMarkerInterface"
        factory=".catalog.Catalog"
        />

</configure>
```

Adding a custom catalog query adapter

New in version 0.9.1: You can specify an own *query* adapter, which builds a query for the catalog adapter.

All search is done through a catalog adapter. The *ICatalogQuery* adapter provides a suitable query usable for the *ICatalog* adapter. It should at least provide a *make_query* method.

```
class ICatalogQuery(interface.Interface):
    """ Plone catalog query interface
    """

    def make_query(**kw):
        """ create a new query or augment an given query
        """
```

To customize a custom catalog tool to perform a search, you have to register an catalog adapter for a more specific interface on the portal. This adapter has to implement the *ICatalog* interface.

```
from zope import interface
from plone.jsonapi.routes.interfaces import ICatalogQuery

class CatalogQuery(object):
    """Catalog query adapter
    """
    interface.implements(ICatalogQuery)

    def __init__(self, catalog):
        self.catalog = catalog

    def make_query(self, **kw):
        """create a query suitable for the catalog
        """
        query = {"sort_on": "created", "sort_order": "descending"}
        query.update(kw)
        return query
```

Register the adapter in your *configure.zcml* file for your special interface:

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

    <!-- Adapter for a custom query adapter -->
    <adapter
        for=".interface.ICustomCatalogInterface"
        factory=".catalog.CatalogQuery"
        />

</configure>
```

CHAPTER 6

CRUD

Each content route provider shipped with this package, provides the basic CRUD *Operations* functionality to *get*, *create*, *delete* and *update* the resource handled.

New in version 0.8.1: Added route providers to *cut*, *copy* and *paste* contents

Unified API

URL Schema `<BASE URL>/<OPERATION>/<uid:optional>`

There is a convenient and unified way to fetch the content without knowing the resource. This unified resource is directly located at the *Base URL*.

Response Format

The response format of the unified *get* API differs from the default *Response Format* and omits the *items* list. The content information is directly provided in the root of the returned JSON object. Therefore it is only suitable to return a single object.

```
{
    _runtime: 0.00381,
    uid: "7455c9b14e3c48c9b0be19ca6a142d50",
    tags: [ ],
    portal_type: "Document",
    id: "front-page",
    description: "Welcome to Plone",
    api_url: "http://localhost:8080/Plone/@@API/plone/api/1.0/documents/
↪7455c9b14e3c48c9b0be19ca6a142d50",
    effective: "1000-01-01T00:00:00+02:00",
    title: "Welcome to Plone",
    url: "http://localhost:8080/Plone/front-page",
    created: "2014-10-14T20:22:19+02:00",
    modified: "2014-10-14T20:22:19+02:00",
    type: "Document"
}
```

GET

The *get* route will return the content located at the given UID.

http://localhost:8080/Plone/@@API/plone/api/1.0/get/<uid:optional>

The given optional UID defines the target object to get. You can omit this UID and specify the path to the object with a request parameter.

Example

Getting an object by its **physical path**:

http://localhost:8080/Plone/@@API/plone/api/1.0/get?path=/Plone/folder

Or you can specify the **parent path** and the **id** of the object

http://localhost:8080/Plone/@@API/plone/api/1.0/get?parent_path=/Plone&id=folder

New in version 0.4: Adding 0 or the string *portal* as UID returns the portal Object.

New in version 0.9.1: The *get* route is now obsolete. Please use the base url to retrieve a content by uid, e.g.: http://localhost:8080/Plone/@@API/plone/api/1.0/<uid>

CREATE

The *create* route will create the content inside the container located at the given UID.

http://localhost:8080/Plone/@@API/plone/api/1.0/create/<uid:optional>

The given optional UID defines the target container. You can omit this UID and specify all the information in the HTTP POST body.

Example

This example shows possible variations of a HTTP POST body sent to the JSON API with the header **Content-Type: application/json** set.

```
{
    portal_type: "Document", // mandatory
    id: "test", // mandatory if title is not set
    title: "test", // mandatory if id is not set
    parent_uid: "7455c9b14e3c48c9b0be19ca6a142d50", // you can specify the UID for␣
↪the parent folder
    parent_path: "/Plone/folder", // or the physical path of the parent container
    ...
}
```

UPDATE

The *update* route will update the content located at the given UID.

http://localhost:8080/Plone/@@API/plone/api/1.0/update/<uid:optional>

The given optional UID defines the object to update. You can omit this UID and specify all the information in the HTTP POST body.

Example

```
{
    uid: "7455c9b14e3c48c9b0be19ca6a142d50", // you can either specify the UID
    path: "/Plone/folder/test", // or the physical path to the object
    id: "test", // or the id and the path of the parent container
    parent_path: "/Plone/folder",
    ...
}
```

DELETE

The *delete* route will delete the content located at the given UID.

http://localhost:8080/Plone/@@API/plone/api/1.0/delete/<uid:optional>

The given optional UID defines the object to delete. You can omit this UID and specify all the information in the HTTP POST body.

Example

Delete an object by its **physical path**:

http://localhost:8080/Plone/@@API/plone/api/1.0/delete?path=/Plone/folder

Or you can specify the **parent path** and the **id** of the object

http://localhost:8080/Plone/@@API/plone/api/1.0/delete?parent_path=/Plone&id=folder

Or you can specify these information in the request body:

```
{
    uid: "7455c9b14e3c48c9b0be19ca6a142d50", // you can either specify the UID
    path: "/Plone/folder/test", // or the physical path to the object
    id: "test", // or the id and the path of the parent container
    parent_path: "/Plone/folder",
    ...
}
```

CUT

The *cut* route will cut the content located at the given UID.

http://localhost:8080/Plone/@@API/plone/api/1.0/cut/<uid:optional>

The given optional UID defines the object to cut. You can omit this UID and specify all the information either in the HTTP POST body or as request arguments.

Example

Cut an object by its **physical path**:

http://localhost:8080/Plone/@@API/plone/api/1.0/cut?path=/Plone/folder

Or you can specify the **parent path** and the **id** of the object

http://localhost:8080/Plone/@@API/plone/api/1.0/cut?parent_path=/Plone&id=folder

Or you can specify these information in the request body:

```
{
    uid: "7455c9b14e3c48c9b0be19ca6a142d50", // you can either specify the UID
    path: "/Plone/folder/test", // or the physical path to the object
    id: "test", // or the id and the path of the parent container
    parent_path: "/Plone/folder",
    ...
}
```

COPY

The *copy* route will copy the content located at the given UID.

http://localhost:8080/Plone/@@API/plone/api/1.0/copy/<uid:optional>

The given optional UID defines the object to copy. You can omit this UID and specify all the information either in the HTTP POST body or as request arguments.

Example

Copy an object by its **physical path**:

http://localhost:8080/Plone/@@API/plone/api/1.0/copy?path=/Plone/folder

Or you can specify the **parent path** and the **id** of the object

http://localhost:8080/Plone/@@API/plone/api/1.0/copy?parent_path=/Plone&id=folder

Or you can specify these information in the request body:

```
{
    uid: "7455c9b14e3c48c9b0be19ca6a142d50", // you can either specify the UID
    path: "/Plone/folder/test", // or the physical path to the object
    id: "test", // or the id and the path of the parent container
    parent_path: "/Plone/folder",
    ...
}
```

PASTE

The *paste* route will paste the previous cutted/copied content to the location identified by the given UID.

http://localhost:8080/Plone/@@API/plone/api/1.0/paste/<uid:optional>

The given optional UID defines the target object (usually a folder). You can omit this UID and specify all the information either in the HTTP POST body or as request arguments.

Example

Paste to a target identified by its **physical path**:

http://localhost:8080/Plone/@@API/plone/api/1.0/paste?path=/Plone/folder

Or you can specify the **parent path** and the **id** of the object

http://localhost:8080/Plone/@@API/plone/api/1.0/paste?parent_path=/Plone&id=folder

Or you can specify these information in the request body:

```
{
    uid: "7455c9b14e3c48c9b0be19ca6a142d50", // you can either specify the UID
    path: "/Plone/folder/test", // or the physical path to the object
    id: "test", // or the id and the path of the parent container
    parent_path: "/Plone/folder",
    ...
}
```

CHAPTER 7

Dexterity Content

New in version 0.7: Added Dexterity Data Manager (see: datamanagers.py)

New in version 0.9.2: Added Zope Schema Field Manager (see: fieldmanagers.py)

Dexterity Content Types are handled by a Data Manager (*IDataManager*) and a Field Manager (*IFieldManager*) respectively. Also see *Adding a custom data manager*. Integrators just have to add route providers for their Dexterity Content Types (see: *Adding a custom route provider*) and the JSON API should handle the heavy lifting.

Security

plone.jsonapi.routes checks only the permission on the content object to be at least *cmf.ModifyPortalContent*. Checks on the field level will be added in future versions.

CHAPTER 8

Example

This section provides some common examples how to use *plone.jsonapi.routes*.

Get content by uid

Getting an object by its **UID**:

http://localhost:8080/Plone/@@API/plone/api/1.0/3762908a5d2c4917b9d2dbaf2a9be1cc

Get content by path

Getting a content by its **physical path**:

http://localhost:8080/Plone/@@API/plone/api/1.0/get?path=/Plone/folder

Get content by parent_path and id

Get a content by specifying the **parent path** and the **id**:

http://localhost:8080/Plone/@@API/plone/api/1.0/get?parent_path=/Plone&id=folder

Copy/Cut/Paste content

Contents can be copied or cutted to the clipboard to paste it later somewhere else. This example shows how to copy a folder located in Plone with the physical path */Plone/folder*.

You can either cut or copy by the *parent_path* & *id* pair:

http://localhost:8080/Plone/@@API/plone/api/1.0/copy?parent_path=/Plone&id=folder

http://localhost:8080/Plone/@@API/plone/api/1.0/cut?parent_path=/Plone&id=folder

Or you can simply take the physical path by specifying the *path*:

http://localhost:8080/Plone/@@API/plone/api/1.0/copy?path=/Plone/folder

http://localhost:8080/Plone/@@API/plone/api/1.0/cut?path=/Plone/folder

Or if you know the UID, you can use the *uid* parameter:

http://localhost:8080/Plone/@@API/plone/api/1.0/copy/3762908a5d2c4917b9d2dbaf2a9be1cc

http://localhost:8080/Plone/@@API/plone/api/1.0/cut/3762908a5d2c4917b9d2dbaf2a9be1cc

After you cutted or copied the content, you can paste it by providing the target folder in the known way.

If you would like to paste it in the portal root, just use the UID 0 or the path of your Plone site:

http://localhost:8080/Plone/@@API/plone/api/1.0/paste/0

http://localhost:8080/Plone/@@API/plone/api/1.0/paste?path=/Plone

Search contents

To search **all** contents of the portal, you can utilize the *search* route:

http://localhost:8080/Plone/@@API/plone/api/1.0/search

The search results can be refined by using request parameters, e.g.:

http://localhost:8080/Plone/@@API/plone/api/1.0/search?q=test

http://localhost:8080/Plone/@@API/plone/api/1.0/search?q=test&limit=10

http://localhost:8080/Plone/@@API/plone/api/1.0/search?q=test&limit=10&portal_type=Folder

http://localhost:8080/Plone/@@API/plone/api/1.0/search?q=test&limit=10&portal_type=Folder&Creator=admin

Basically, you can use any defined index of your Plone site.

There are some convenience keys like *q* for the SearchableText index. See *Parameters* for further details.

Delete contents

It is possible to delete contents from your Plone site with the *delete* route. See *CRUD* for details.

You can either delete by the *parent_path* & *id* pair:

http://localhost:8080/Plone/@@API/plone/api/1.0/delete?parent_path=/Plone&id=folder

Or you can simply take the physical path by specifying the *path*:

http://localhost:8080/Plone/@@API/plone/api/1.0/delete?path=/Plone/folder

Or if you know the UID, you can use the *uid* parameter:

http://localhost:8080/Plone/@@API/plone/api/1.0/delete/3762908a5d2c4917b9d2dbaf2a9be1cc

It is even possible to delete multiple objects. This can be done by sending the data in a list within the POST body. See *CRUD* for mode details.

Note: The API do not allow to delete the portal object (UID=0)

Get the portal object

To fetch the portal object you can use the *plonesite* route:

http://localhost:8080/Plone/@@API/plone/api/1.0/plonesite

Get folder contents

If you are interested in the contents of a folderish content type, you can append the *children=yes* request parameter to the url:

http://localhost:8080/Plone/@@API/plone/api/1.0/plonesite?children=yes

this will add a *children* list to the response which includes all contents of the requested object. This can actually be done with any route provider.

Get the full object

The API is designed in a two step architecture, see the API doc:*Concept*. Therefore only the catlog brain results are returned in the first step.

You can bypass this step by specifying the *complete=yes* request parameter.

Note: The *complete=yes* parameter also affects the child nodes

Note: It is not recommended to use the complete flag, as it is significant slower.

CHAPTER 9

Useful Links

This section contains a collection of useful links to the project and its dependent packages.

Plone

- Plone Website
- plone.api @ pypi

plone.jsonapi.routes

- plone.jsonapi.routes @ pypi
- plone.jsonapi.routes @ GitHub
- plone.jsonapi.routes Issues

plone.jsonapi.core

- plone.jsonapi.core @ pypi
- plone.jsonapi.core @ GitHub
- plone.jsonapi.core Issues

CHAPTER 10

Contributors

Thanks to all the contributors!

- José Dinuncio [jdinuncio]
- Jian Aijun [jianaijun]
- Jan Müller [jan-mue]
- Percy Barboza
- Gagaro [Gagaro]
- Mauro Amico [mamico]
- Thomas Clement Mogensen [tmog]
- Alessandro Pisa [ale-rt]

CHAPTER 11

License

CHAPTER 12

Changelog

0.9.3 - 2017-05-14

Changes

- *IDataManager* contain now a *json_data* method to return a JSON suitable return structure or delegate to the *IFieldManager.json_data* method. Please see section “Customizing” in the documentation for more details.
- Added support for *z3c.relationfield.interfaces.IRelationList* fields
- Added support for *plone.app.textfield.interfaces.IRichText* fields
- Added support for *plone.app.blob.interfaces.IBlobField* fields
- More code cleanup and refactoring (coming closer to a robust 1.0.0 release!)

0.9.2 - 2017-05-12

Changes

- Added *IFieldManager* adapter to *get* and *set* the value on field level.
- Removed *build* number from version route JSON response.
- Content route improved.
- API refactored.
- Improved *users* route.
- Updated documentation

0.9.1 - 2017-04-20

Changes

- Added *ICatalog* and *ICatalogQuery* adapter for full search control. See docs for usage.
- Removed *query* module in favor of the new adapters.
- Removed multiple catalog query functionality. Please define a custom *ICatalog* adapter if you need it.
- Added generic route provider for all *portal_types*. N.B. The old-style route providers, e.g. *folders*, *documents* etc., are now obsolete. Please use the lower portal type name instead, e.g. *folder*, *docuememt* ...
- The *users* route shows now more details of the users and results are now batched.
- Removed default *getObjPositionInParent* sorting. Please specify explicitly via *sort_on*.
- UID of the plone site is now '0' instead of 0.
- Huge code refactoring and cleanup.

0.9.0 - 2017-01-12

Changes

- API mthods *get_items* and *get_batched* accept now keyword paramters. Keywords can be catalog indexes, e.g. *id=document-1* or a complete catalog query objejt, e.g. *query={'portal_type': 'Document'}*.
- Changed *get_contents* method to use the *search* functionality from the *query* module.
- More doctests added

0.8.9 - 2017-01-11

Changes

- Catalog to query can now be set via the *catalog* request parameter.
- Optimized search logic
- Fixed issue with multiple *portal_type* parameters in request
- Code Refactoring
- More tests

0.8.8 - 2017-01-10

Changes

- Handle catalog queries for multiple contents, which might be located in different catalogs.
- Fixed an issue where the batch navigation did not show more results when using multiple *portal_type* request parameters.

0.8.7 - 2017-01-10

Changes

- Handle Reference Fields: Reference fields containing a reference can be updated with a dictionary, e.g.:

```
{
  uid: <UID of a content containing a reference field>,
  ReferenceField: {
    "title": "New Title"
  }
}
```

- Added module *underscore* to the tests suite
- Validation for the entire object added
- Get the catalog to query from Archtype Tool and default to *portal_catalog*
- Use explicit namespace in route providers
- Handle Reference Fields (Fields containing and *ImplicitAcquisitionWrapper* object)
- Added ZCML directive to enable/disable route registrations (default enabled):

```
<!-- Disable route registration -->
<plone:jsonapi
    register_api_routes="False"
/>
```

- Version route is now part of the standard route providers
- Dropped AdvancedQuery handling

0.8.6 - 2016-04-08

Fix for broken release 0.8.5

0.8.5 - 2016-04-08

CLOSED ISSUES

- https://github.com/collective/plone.jsonapi.routes/issues/59: API URL for non standard content types
- https://github.com/collective/plone.jsonapi.routes/issues/60: Add a namespace to the route registrations
- https://github.com/collective/plone.jsonapi.routes/issues/63: handle richtext fields
- https://github.com/collective/plone.jsonapi.routes/issues/82: Plone 5 CSFR Protection
- https://github.com/collective/plone.jsonapi.routes/issues/80: Tests for Plone 5
- https://github.com/collective/plone.jsonapi.routes/issues/77: Problem with creating files
- https://github.com/collective/plone.jsonapi.routes/issues/62: 'reference_catalog' not found
- https://github.com/collective/plone.jsonapi.routes/pull/75: Fix api invocation on the zope root

- https://github.com/collective/plone.jsonapi.routes/pull/74: Reuse and improve code to check if a parameter in the request has a True value
- https://github.com/collective/plone.jsonapi.routes/pull/73: Using specifiers to format string (helps compatibility with Python 2.6, improves code readability)

0.8.4 - 2016-01-14

CLOSED ISSUES

- https://github.com/collective/plone.jsonapi.routes/pull/66: api routes: sharing (docs)
- https://github.com/collective/plone.jsonapi.routes/pull/65: api routes: sharing (code)
- https://github.com/collective/plone.jsonapi.routes/pull/61: Use IConstrainTypes adapters for dexterity content

API CHANGES

- Sharing information can be displayed for objects. Use *?sharing=yes*

0.8.3 - 2015-09-14

CLOSED ISSUES

- https://github.com/collective/plone.jsonapi.routes/issues/58: Unit tests: add tests for adapter module
- https://github.com/collective/plone.jsonapi.routes/issues/57: API Change: workflow data optional
- https://github.com/collective/plone.jsonapi.routes/issues/54: Let complete flag overrule "uid rule"
- https://github.com/collective/plone.jsonapi.routes/issues/53: Unit tests: add tests for api module

API CHANGES

- File data **not** included by default anymore. Use *?filedata=yes*
- Workflow data **not** included by default anymore. Use *?workflow=yes*
- Workflow data is now located at the key *workflow*
- The complete flag can be now negated, even if the full object is displayes *?complete=no*
- The *state* key is removed – use *review_state* instead
- Parent URL data included now for brain results

0.8.2 - 2015-09-09

CLOSED ISSUES

- https://github.com/collective/plone.jsonapi.routes/issues/52: Handle field unauthorized errors in the GET API
- https://github.com/collective/plone.jsonapi.routes/issues/51: Default Data Adapters missing

0.8.1 - 2015-09-06

CLOSED ISSUES

- https://github.com/collective/plone.jsonapi.routes/issues/50: API route throws error
- https://github.com/collective/plone.jsonapi.routes/pull/37: Include custom metadata columns
- https://github.com/collective/plone.jsonapi.routes/pull/37: Include custom metadata columns
- https://github.com/collective/plone.jsonapi.routes/issues/49: Setting the ID throws a traceback
- https://github.com/collective/plone.jsonapi.routes/issues/48: Implement cut/copy/paste routes
- https://github.com/collective/plone.jsonapi.routes/issues/46: Route Provider *portal* throws TypeError
- https://github.com/collective/plone.jsonapi.routes/issues/47: ZCML directive to enable AdvancedQuery if installed

ENHANCEMENTS

- API actions to cut/copy/paste contents
- New route provider *plonesites*
- Support for catalog brain schema

0.8 - 2015-07-20

CLOSED ISSUES

- https://github.com/collective/plone.jsonapi.routes/issues/45: Add authentication routes
- https://github.com/collective/plone.jsonapi.routes/issues/44: Add the filename to the JSON data
- https://github.com/collective/plone.jsonapi.routes/issues/43: API: Intermediate Folder creation
- https://github.com/collective/plone.jsonapi.routes/issues/41: Field Type Validation
- https://github.com/collective/plone.jsonapi.routes/issues/42: ContentType for Dexterity Files CT

0.7 - 2015-07-09

CLOSED ISSUES

- https://github.com/collective/plone.jsonapi.routes/issues/9: Handle Dexterity Behavior fields
- https://github.com/collective/plone.jsonapi.routes/issues/38: Filename handling
- https://github.com/collective/plone.jsonapi.routes/issues/36: Mime Type handling

OTHER CHANGES

- Updated Documentation
- Request module: Added helper functions
- Travis CI integration

0.6 - 2015-02-22

CLOSED ISSUES

- https://github.com/collective/plone.jsonapi.routes/issues/33: Image detail URL throws error
- https://github.com/collective/plone.jsonapi.routes/issues/34: Failed POST request return HTTP 200
- https://github.com/collective/plone.jsonapi.routes/issues/35: DataManager does not check field permissions

0.5 - 2015-02-20

CLOSED ISSUES

- https://github.com/collective/plone.jsonapi.routes/issues/32: Add documentation for the new ZPublisher record behavior
- https://github.com/collective/plone.jsonapi.routes/issues/31: Change default sort order to ascending
- https://github.com/collective/plone.jsonapi.routes/pull/30: fix standard query ignoring sort_on and sort_order
- https://github.com/collective/plone.jsonapi.routes/issues/27: querying does not support ZPublisher record format
- https://github.com/collective/plone.jsonapi.routes/issues/25: Add support for Plone 4.2

OTHER CHANGES

- Added batch adapter
- Added more tests

0.4 - 2015-01-13

FIXED ISSUES

- https://github.com/collective/plone.jsonapi.routes/issues/22: Absoulte url is missing in update/create response
- https://github.com/collective/plone.jsonapi.routes/issues/21: Image Route throws an error

ENHANCEMENTS

- https://github.com/collective/plone.jsonapi.routes/issues/20: Support query for DateTime Indexes
- https://github.com/collective/plone.jsonapi.routes/issues/23: Support query for created/modified DateTime ranges

OTHER CHANGES

- added *IDataManager* field data manager
- added */auth* route to enforce a basic auth
- added a custom exception class to set the right response status
- added *recent_modified* and *recent_created* handling
- added unittests for the *api* and *request* module
- no more request passing anymore - all handled by the request module now

0.3 - 2014-10-14

FIXED ISSUES

- https://github.com/collective/plone.jsonapi.routes/issues/16: Files can not be created/updated with base64 encoded data
- https://github.com/collective/plone.jsonapi.routes/issues/10: Fails on NamedBlobFile dexterity fields
- https://github.com/collective/plone.jsonapi.routes/pull/11: Typo in brain adapter
- https://github.com/collective/plone.jsonapi.routes/issues/14: Missing UIDs for complete objects

ENHANCEMENTS

- https://github.com/collective/plone.jsonapi.routes/issues/12: Add batching
- https://github.com/collective/plone.jsonapi.routes/issues/13: Add a flag to return the full fledged object results immediately
- https://github.com/collective/plone.jsonapi.routes/issues/19: Need to do a GET on a file using file path without using uid
- https://github.com/collective/plone.jsonapi.routes/issues/18: destination handling
- https://github.com/collective/plone.jsonapi.routes/issues/3: Add buildout configs inside package

DOCUMENTATION

- https://github.com/collective/plone.jsonapi.routes/issues/2: Sphinx documentation started

0.2 - 2014-03-05

FIXED ISSUES

- https://github.com/ramonski/plone.jsonapi.routes/issues/5: Dexterity support
- https://github.com/ramonski/plone.jsonapi.routes/issues/4: Update on UID Urls not working
- https://github.com/ramonski/plone.jsonapi.routes/issues/1: Started with some basic browsertests

API CHANGES

- API root url provided.
- Image and file fields are now rendered as a nested structure, e.g:

```
{
  data: b64,
  size: 42,
  content_type: "image/png"
}
```

- Workflow info is provided where possible, e.g:

```
{
  status: "Private",
  review_state: "private",
  transitions: [
    {
      url: ".../content_status_modify?workflow_action=submit",
      display: "Puts your item in a review queue, so it can be published on the␣
↪site.",
      value: "submit"
    },
  ],
  workflow: "simple_publication_workflow"
}
```

0.1 - 2014-01-23

- first public release