collective.jsonify

Release 1.0

July 13, 2015

Contents

`collective.jsonify` exports your Plone content to JSON.

Many packages that export data from Plone have complicated dependencies, and so only work with Plone 3.0 or higher (or not even with 3.0). `collective.jsonify`'s only dependency is simplejson. It can be installed in any Plone version as far back as:

- Plone 2.1 (or probably even Plone 2.0, but not tested)
- Zope 2.6.4 (with CMF rather than Plone)
- Python 2.2

The exported JSON is a collective.transmogrifier friendly format. Install `collective.jsonify` on a site you want to export from, and setup an import transmogrifier pipeline on the site you're importing to, using the blueprints in the collective.jsonmigrator package.

Alternatively use the provided export script by adding it to

For more information see the documentation.

> **Warning** This product may contain traces of nuts.
>
> **Author** Rok Garbas, *migrating for you since 2008*
>
> **Source** http://github.com/collective/collective.jsonify

CHAPTER 1

How to install it

Install `collective.jsonify` for your Plone site, so that it is available in your Plone site's `PYTHONPATH`, including the `simplejson` package. The easiest way is to use buildout, as for any other modern Plone project. Other options include:

1. Play with PYTHONPATH manually.
2. Use `easy_install collective.jsonify` or `pip collective.jsonify` which will also pull `simplejson`.

Note: if you are working with python 2.2, then you will need to install a 'tweaked branch of simplejson <https://github.com/simplejson/simplejson/tree/python2.2>'_.

Then run your Zope instance, go to the Zope root and create the necessary External Methods.

External method for exporting JSON files to the filesystem:

- export_content: - id: `export_content` - module name: `collective.jsonify.json_methods` - function name: `export_content`

External methods for remote access from the importing Plone instance, using `collective.jsonmigrator`:

- get_item - id: `get_item` - module name: `collective.jsonify.json_methods` - function name: `get_item`
- get_children: - id: `get_children` - module name: `collective.jsonify.json_methods` - function name: `get_children`
- get_catalog_results: - id: `get_catalog_results` - module name: `json_methods` - function name: `get_catalog_results`

It's true that External Methods are not the nicest to work with and using them makes the setup a little long. But the nice thing about External Methods is that they work in Plone 1.0 as well as in Plone 4.0, so you could potentially use `collective.jsonify` to migrate from very old Plone versions.

CHAPTER 2

How to use it

`collective.jsonify` is intended to be used in conjunction with `collective.jsonmigrator`. There you can find an example transmogrifier pipeline that connects to the Plone site running `collective.jsonify`, crawls it, extracts the content and imports it into the target site.

To see what `collective.jsonmigrator` is actually seeing you can issue "json views" on content you want to explore:

```
http://localhost:8080/Plone/front-page/get_item
http://localhost:8080/Plone/front-page/get_children
```

The first gets all content out of `front-page`; the second lists all content contained inside this object and returns their ids.

Finally, you can use `get_catalog_results` to catalog query results as a list of paths. To use it, you need to hand your query as a base64'ed Python dict string. Here's an example of doing this with curl:

```
curl --data catalog_query=$(echo '{"Type": "Slide"}' | base64 -w0) \
  'http://localhost:8080/Plone/portal_catalog/get_catalog_results
```

CHAPTER 3

Using the exporter

Instead of doing on-the-fly exporting with collective.jsonmigrator, you can also export your site's content to json files for multiple re-use. This is done by the export script and the external method, as described above. You can also batch-export the contents, if you get out of memory on your exporting machine. Here is an example on how to configure the export script for using as an external method:

```
from collective.jsonify.export import export_content as export_content_orig


def export_content(self):
    return export_content_orig(
        self,
        basedir='/tmp',  # export directory
        extra_skip_classname=['ATTopic'],
        batch_start=5000,
        batch_size=5000,
        batch_previous_path='/Plone/last/exported/path'  # optional, but saves more memory because no
    )
```

To start the export, just open the url in your browser:

```
http://localhost:8080/Plone/export_content
```

CHAPTER 4

How to extend it

We try to cover the basic Plone types to export useful content out of Plone. We cannot predict all usecases, but if you have custom requirements it's easy to extend functionality. You have a few options:

- You can pass additional wrappers to the `get_item` External Method. Of course you have to have these wrappers in your PYTHONPATH:

```
http://localhost:8080/Plone/front-page/get_item?additional_wrappers=myproject.wrapper1.Wrapper;m
```

- If you need something completely custom, you could override the `get_item` and `get_children` External Methods.

CHAPTER 5

Code

CHAPTER 6

Changelog

6.1 1.0 (2015-05-16)

- Let the wrapper test correctly for `zope.interface` and `Interface` interfaces. [thet]
- In the wrapper class, call the value in decode, if it's a callable. [thet]
- When serializing datetime, date, time or DateTime properties, just use the unicode representation which can be parsed. [thet]
- When serializing values, if there is no special handler for a field type, just try to unicode the value. [thet]
- Fix export of defaultPage and layout. Before, always the defaultPage was set now layout is always set and defaultPage only, if there is one defined. [thet]
- Handle plone.formwidget.geolocation Dexterity field types. [thet]
- Check, if wrapper methods for Zope/CMF objects are Zope/CMF only objects by testing for Archetypes and Dexterity first. [thet]
- Add `BlobField` for `get_archetypes_fields`. [thet]
- Don't try to convert ints to unicode in get_properties(). [djowett]
- Zope 2.6 support for collective.jsonify. [djowett]
- Fix setup.py to work with Python 2.2. [djowett]
- Add error type to tracebacks. [djowett]
- Fix read of NamedBlobImage, NamedFile and NamedBlobFile in dexterity objects. [djowett]
- Fix read of field for unicode transcoding in dexterity objects. [djowett]
- Make `archetypes.schemaextender` support more generic and handle probably most use cases. [thet]
- Add `_directly_provided` export field for the object's directly provided interfaces. [thet]
- Add json_methods module to own Extension folder, which makes it automatically available and unnecessary to add it to the instance's Extension folder. [thet]
- Don't skip `ComputedField` fields, but just export their computed value. Better skip them in your transmogrifier import pipeline. [thet]
- Allow a `skip_callback` function to be passed to the `export_content` function. It evaluates to `True`, if the current visited item should be excluded from exporting. [thet]
- Export a content's references as list of UID values. [thet]

- Declare the `content_type` of a field's value only for `TextField` and `StringField`. [thet]
- Add example buildouts for Plone 2.1, 2.5, 3 and 4. [thet]
- Declare `base64` encoding for _datafield_FIELDNAME structures. This is used to correctly decode in transmogrify.dexterity. [thet]
- Add export module from `collective.blueprint.jsonmigrator` and modify to use collective.jsonify wrapper. Use it in Plone 2.1 by adding it as external method. [thet]
- PEP 8. [thet]
- Fixing local roles export. [realefab]
- Make ATExtensionFields serializable. [jsbueno]
- Fixes exporting of Image types that use ATBlob. [jsbueno]

6.2 0.2 (2014-08-18)

- Support p.a.collection QueryField. [jone]
- Dexterity support. [djowett]
- Add Blob fields support. Use specific methods to retrieve filename, content type and size. [gborelli]
- Add _get_at_field_value to wrappe.Wrapper in order to use accessor method for Archetypes fields. [gborelli]
- @@jsonify view added. See README_JSONIFY_VIEW.rst for more [pieretti]

6.3 0.1 (2011-03-14)

- documentation added [garbas]
- collection of external methods from `collective.blueprint.jsonmigrator` and `collective.sync_migrator`. [garbas]
- initial release [garbas]