invenio-sipstore Documentation

Release 1.0.0a7

CERN

Aug 18, 2017

Contents

Submission Information Package store for Invenio.

This is an experimental developer preview release.

- Free software: GPLv2 license
- Documentation: https://invenio-sipstore.readthedocs.io/

CHAPTER 1

User's Guide

This part of the documentation will show you how to get started in using Invenio-SIPStore.

Installation

Invenio-SIPStore is on PyPI so all you need is:

```
$ pip install invenio-sipstore
```

Configuration

Default configuration of Invenio-SIPStore module.

invenio_sipstore.config.**SIPSTORE_AGENT_FACTORY = 'invenio_sipstore.api.SIP._build_agent_info'**
Factory to build the agent, stored for the information about the SIP.

invenio_sipstore.config.**SIPSTORE_AGENT_JSONSCHEMA_ENABLED = True**
Enable SIP agent validation by default.

invenio_sipstore.config.**SIPSTORE_ARCHIVER_DIRECTORY_BUILDER = 'invenio_sipstore.archivers.utils.default_a**
Builder for archived SIPs.

invenio_sipstore.config.**SIPSTORE_ARCHIVER_LOCATION_NAME = 'archive'**
Name of the invenio_files_rest.models.Location object, which will specify to the archive location in its URI.

invenio_sipstore.config.**SIPSTORE_ARCHIVER_METADATA_TYPES = ['json']**
List of metadata types (SIPMetadataType.name) to include in archiving.

invenio_sipstore.config.**SIPSTORE_ARCHIVER_SIPFILE_NAME_FORMATTER = 'invenio_sipstore.archivers.utils.de**
Filename formatter for the archived SIPFile.

invenio_sipstore.config.**SIPSTORE_ARCHIVER_SIPMETADATA_NAME_FORMATTER = 'invenio_sipstore.archivers.u**
Filename formatter for archived SIPMetadata.

invenio_sipstore.config.**SIPSTORE_BAGIT_TAGS = [('Source-Organization', 'European Organization for Nuclear R**
Default list of BagIt tags that will be written.

invenio_sipstore.config.**SIPSTORE_DEFAULT_AGENT_JSONSCHEMA = 'sipstore/agent-v1.0.0.json'**
Default JSON schema for extra SIP agent information.

For more examples, you can have a look at Zenodo's config: https://github.com/zenodo/zenodo/tree/master/zenodo/modules/sipstore/jsonschemas/sipstore

invenio_sipstore.config.**SIPSTORE_DEFAULT_BAGIT_JSONSCHEMA = 'sipstore/bagit-v1.0.0.json'**
Default JSON schema for BagIt archiver.

invenio_sipstore.config.**SIPSTORE_FILEPATH_MAX_LEN = 1024**
Max filepath length.

invenio_sipstore.config.**SIPSTORE_FILE_STORAGE_FACTORY = 'invenio_files_rest.storage.pyfs.pyfs_storage_facto**
Archived file storage factory.

Usage

Submission Information Package store for Invenio.

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

API Docs

Extension

Submission Information Package store for Invenio.

class `invenio_sipstore.ext.`**`InvenioSIPStore`**(*app=None*)
Invenio-SIPStore extension.

Extension initialization.

`init_app`(*app*)
Flask application initialization.

`init_config`(*app*)
Initialize configuration.

API

API for Invenio-SIPStore.

class `invenio_sipstore.api.`**`RecordSIP`**(*recordsip*, *sip*)
API for managing SIPRecords.

Constructor.

Parameters

- **`recordsip`** (*`invenio_sipstore.models.RecordSIP`*) – the RecordSIP model to manage
- **`sip`** (*`invenio_sipstore.api.SIP`*) – the SIP associated

classmethod create(*pid*, *record*, *archivable*, *create_sip_files=True*, *user_id=None*, *agent=None*)
Create a SIP, from the PID and the Record.

Apart from the SIP itself, it also creates `RecordSIP` for the SIP-PID-Record relationship, as well as `SIPFile` objects for each of the files in the record, along with `SIPMetadata` for the metadata. Those objects are not returned by this function but can be fetched by the corresponding RecordSIP attributes `sip`, `sip.files` and `sip.metadata`.

Parameters

- **pid** (*invenio_pidstore.models.PersistentIdentifier*) – PID of the published record ('recid').
- **record** (*invenio_records.api.Record*) – Record for which the SIP should be created.
- **archivable** (*bool*) – tells if the record should be archived. Usefull when `Invenio-Archivematica` is installed.
- **create_sip_files** (*bool*) – If True the SIPFiles will be created.

Returns RecordSIP object.

Return type *invenio_sipstore.api.RecordSIP*

sip
Return the SIP corresponding to this record.

Return type *invenio_sipstore.api.SIP*

class invenio_sipstore.api.**SIP**(*sip*)
API for managing SIPs.

Constructor.

Parameters **sip** (*invenio_sipstore.models.SIP*) – the SIP model associated

agent
Return the agent of the associated model.

archivable
Tell if the SIP should be archived.

archived
Tell if the SIP has been archived.

attach_file(*file*)
Add a file to the SIP.

Parameters **file** – the file to attach. It must at least implement a *key* and a valid *file_id*. See `invenio_files_rest.models.ObjectVersion`.

Returns the created SIPFile

Return type *invenio_sipstore.models.SIPFile*

attach_metadata(*type*, *metadata*)
Add metadata to the SIP.

Parameters

- **type** (*str*) – the type of metadata (a valid *invenio_sipstore.models.SIPMetadataType* name)
- **metadata** (*str*) – the metadata to attach.

Returns the created SIPMetadata

Return type *invenio_sipstore.models.SIPMetadata*

classmethod create(*archivable*, *files=None*, *metadata=None*, *user_id=None*, *agent=None*)
Create a SIP, from the PID and the Record.

Apart from the SIP itself, it also creates `SIPFile` objects for each of the files in the record, along with `SIPMetadata` for the metadata. Those objects are not returned by this function but can be fetched by the corresponding SIP attributes 'files' and 'metadata'. The created model is stored in the attribute 'model'.

Parameters

- **archivable** (*bool*) – tells if the SIP should be archived or not. Usefull if `Invenio-Archivematica` is installed.
- **files** – The list of files to associate with the SIP. See *invenio_sipstore.api.SIP.attach_file()*
- **metadata** (*dict*) – A dictionary of metadata. The keys are the type (valid *invenio_sipstore.models.SIPMetadataType* name) and the values are the content (string)
- **user_id** – the ID of the user. If not given, automatically computed
- **agent** – If not given, automatically computed

Returns API SIP object.

Return type *invenio_sipstore.api.SIP*

files
Return the list of files associated with the SIP.

Return type list(*invenio_sipstore.models.SIPFile*)

classmethod get_sip(*uuid*)
Get a SIP API object from the UUID if a model object.

id
Return the ID of the associated model.

metadata
Return the list of metadata associated with the SIP.

Return type list(*invenio_sipstore.models.SIPMetadata*)

user
Return the user of the associated model.

Models

Invenio-SIPStore database models.

class invenio_sipstore.models.**RecordSIP**(***kwargs*)
An association table for Records and SIPs.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

pid
Relation to the PID associated with the record SIP.

pid_id
Id of the PID pointing to the record.

sip
Relation to the SIP associated with the record.

sip_id
Id of SIP.

class invenio_sipstore.models.**SIP**(***kwargs*)
Submission Information Package model.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

agent
Agent information regarding given SIP.

archivable
Boolean stating if the SIP should be archived or not.

archived
Boolean stating if the SIP has been archived or not.

classmethod create(*user_id=None*, *agent=None*, *id_=None*, *archivable=True*, *archived=False*)
Create a Submission Information Package object.

Parameters

- **user_id** (*int*) – Id of the user responsible for the SIP.
- **agent** (*dict*) – Extra information on submitting agent in JSON format.
- **archivable** (*bool*) – Tells if the SIP should be archived or not.
- **archived** (*bool*) – Tells if the SIP has been archived.

id
Id of SIP.

user
Relation to the User responsible for the SIP.

user_id
User responsible for the SIP.

class invenio_sipstore.models.**SIPFile**(***kwargs*)
Extra SIP info regarding files.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

checksum
Return the checksum of the file.

file
Relation to the SIP along which given file was submitted.

file_id
Id of the FileInstance.

filepath
Filepath of submitted file within the SIP record.

sip
Relation to the SIP along which given file was submitted.

sip_id
Id of SIP.

size
Return the size of the file.

storage_location
Return the location of the file in the current storage.

validate_key(*filepath*, *filepath_*)
Validate key.

class invenio_sipstore.models.**SIPMetadata**(***kwargs*)
Extra SIP info regarding metadata.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

content
Text blob of the metadata content.

sip
Relation to the SIP along which given metadata was submitted.

sip_id
Id of SIP.

type
Relation to the SIPMetadataType.

type_id
ID of the metadata type.

class invenio_sipstore.models.**SIPMetadataType**(***kwargs*)
Type of the metadata added to an SIP.

The type describes the type of file along with an eventual schema used to validate the structure of the content.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

format
The format of the metadata (xml, json, txt...).

This is used as the extension of the created file during an export.

classmethod get(*id*)
Return the corresponding SIPMetadataType.

classmethod get_from_name(*name*)
Return the corresponding SIPMetadataType.

classmethod get_from_schema(*schema*)
Return the corresponding SIPMetadataType.

id
ID of the SIPMetadataType object.

name
The unique name tag of the metadata type.

schema
URI to a schema that describes the metadata (json or xml schema).

title
The title of type of metadata (i.e. 'Invenio JSON Record v1.0.0').

Errors

Errors for Submission Information Packages.

exception invenio_sipstore.errors.**SIPError**
Base class for SIPStore errors.

exception invenio_sipstore.errors.**SIPUserDoesNotExist**(*user_id*, **args*, ***kwargs*)
User ID for SIP does not exist.

Initialize exception.

Proxies

Proxy definitions.

invenio_sipstore.proxies.**current_sipstore = <LocalProxy unbound>**
Helper proxy to access the SIPStore state object.

Signals

Signals for the module.

invenio_sipstore.signals.**sipstore_archiver_status = <blinker.base.NamedSignal object at 0x7f0fd0f680d0; 's**
Signal sent during the archiving processing.

Sends a dict with the following information inside: - total_files: the total number of files to copy - total_size: the total size to copy - copied_files: the number of copied files - copied_size: the size copied - current_filename: the name of the last copied file - current_filesize: the size of the last copied file

invenio_sipstore.signals.**sipstore_created = <blinker.base.NamedSignal object at 0x7f0fd0f68350; 'sipstore_cre**
Signal sent each time a SIP has been created.

Send the SIP as a parameter: *invenio_sipstore.api.SIP*

Example subscriber

```
def listener(sender, *args, **kwargs):
    # sender is the SIP being archived
    for f in sender.files:
        print(f.filepath)

from invenio_sipstore.signals import sipstore_created
sipstore_created.connect(listener)
```

Utilities

SIPStore utility functions.

invenio_sipstore.utils.**load_or_import_from_config**(*key*, *app=None*, *default=None*)
Load or import value from config.

Returns The loaded value.

invenio_sipstore.utils.**obj_or_import_string**(*value*, *default=None*)
Import string or return object.

Params value Import path or class object to instantiate.

Params default Default object to return if the import fails.

Returns The imported object.

Utilities for SIPStore archivers.

invenio_sipstore.archivers.utils.**chunks**(*iterable*, *n*)
Yield iterable split into chunks.

If 'n' is an integer, yield the iterable as n-sized chunks. If 'n' is a list of integers, yield chunks of sizes: n[0], n[1], ..., len(iterable) - sum(n)

```
>>> from invenio_sipstore.archivers.utils import chunks
>>> list(chunks('abcdefg', 3))
['abc', 'def', 'g']
>>> list(chunks('abcdefg', [1, ]))
['a', 'bcdefg']
>>> list(chunks('abcdefg', [1, 2, 3]))
['a', 'bc', 'def', 'g']
```

invenio_sipstore.archivers.utils.**default_archive_directory_builder**(*sip*)
Build a directory structure for the archived SIP.

Creates a structure that is based on the SIP's UUID. 'abcdefgh-1234-1234-1234-1234567890ab' -> ['ab', 'cd', 'efgh-1234-1234-1234-1234567890ab']

Parameters **sip** – SIP which is to be archived

Returns list of str

invenio_sipstore.archivers.utils.**default_sipfile_name_formatter**(*sipfile*)
Default generator the SIPFile filenames.

Writes doen the file in the archive under the original filename.

WARNING: This can potentially cause security and portability issues if the SIPFile filenames come from the users.

invenio_sipstore.archivers.utils.**default_sipmetadata_name_formatter**(*sipmetadata*)
Default generator for the SIPMetadata filenames.

invenio_sipstore.archivers.utils.**secure_sipfile_name_formatter**(*sipfile*)
Secure filename generator for the SIPFiles.

Since the filenames can be potentially dangerous, not compatible with the underlying file system, or not portable across operating systems this formatter writes the files as a generic name: UUID-<secure_filename>, where <secure_filename> is the original filename which was stripped from any malicious parts (UNIX directory navigation '.', '..', '/'), special protocol parts ('ftp://', 'http://'), special device names on Windows systems, etc. and for maximum portability contains only ASCII characters. Since this operation can cause name collisions, the UUID of the underlying FileInstance is appended as prefix of the filename. For more information on the `secure_filename` function visit: `http://werkzeug.pocoo.org/docs/utils/#werkzeug.utils.secure_filename`

Archivers

Archivers for SIPStore module.

An archiver is an controller that can serialize a SIP to disk according to a specific format. Currently Invenio-SIPStore comes with a BagIt archiver that can write packages according to "The BagIt File Packaging Format (V0.97)".

New formats can be implemented by subclassing *BaseArchiver*.

Base archiver

Base archiver for SIPs.

The base archiver implements a basic API that allows subclasses to not having to worry about e.g. files to disk.

class invenio_sipstore.archivers.base_archiver.**BaseArchiver**(*sip*, *data_dir=u'files'*, *metadata_dir=u'metadata'*, *extra_dir=u''*, *storage_factory=None*, *filenames_mapping_file=None*)
Base archiver.

The archiving is done in two steps:

1. Generation of a list containing file information which contains all relevant information for writing down each file.
2. Actual IO operation on the storage, which takes the previously generated list as input and writes it down to disk.

The first step contains all archiver specific information on the archive structure and all relevant archive metadata that is to be written in addition to the "core" files, which are *SIPFile* and *SIPMetadata* files.

The first step does not produce any side effects to the system. Specific archivers which inherit from this class are expected to primarily overwrite the *BaseArchiver.get_all_files()* method to implement the, archiver-specific structure and any additional archived files.

Relevant public method:

- *BaseArchiver.get_all_files()*

Relevant protected methods:

- *BaseArchiver._get_data_files()*

- *BaseArchiver._get_metadata_files()*
- *BaseArchiver._get_extra_files()*

The second step writes down the generated file information to disk using the configured storage class. By default it uses the file storage factory specified in *SIPSTORE_FILE_STORAGE_FACTORY* configuration variable. This behaviour is overwritable by `storage_factory` parameter that can be provided to the constructor of this class.

Relevant public method:

- *BaseArchiver.write_all_files()*

Relevant protected methods:

- *BaseArchiver._write_sipfile()*
- *BaseArchiver._write_sipmetadata()*
- *BaseArchiver._write_extra()*

Base archiver constructor.

Parameters

- **sip** (*invenio_sipstore.api.SIP*) – the SIP to archive.
- **data_dir** – Subdirectory in archive where the SIPFiles will be written.
- **metadata_dir** – Subdirectory in archive where the SIPMetadata files will be written.
- **extra_dir** – Subdirectory where all any extra files, that are specific to an archive standard, should be written.
- **storage_factory** – Storage factory used to create a new storage class instance.
- **filenames_mapping_file** – Mapping of file names.

_generate_extra_info(*content*, *filename*)
Generate the file information dictionary from a raw content.

_generate_sipfile_info(*sipfile*)
Generate the file information dictionary from a SIP file.

_generate_sipmetadata_info(*sipmetadata*)
Generate the file information dictionary from a SIP metadata.

_get_data_files()
Get the file information for all the data files.

The structure is defined by the JSON Schema `sipstore/file-v1.0.0.json`.

Returns list of dict containing file information.

_get_extra_files(*data_files*, *metadata_files*)
Get file information on any additional files in the archive.

Return any additional files that are to be written. If `filenames_mapping_file` was set in the constructor, this method will generate a file containing the SIP filenames mapping.

The structure is defined by the JSON Schema `sipstore/file-v1.0.0.json`.

Parameters

- **data_files** – File information on the SIP files.
- **metadata_files** – File information on the SIP metadata files

Returns list of dict containing any additional files information.

_get_metadata_files()
Get the file information for the metadata files.

The structure is defined by the JSON Schema `sipstore/file-v1.0.0.json`.

Returns list of dict containing file information.

_get_sipfile_filename_mapping(*filesinfo*)
Generate filename mapping for SIPFiles.

Due to archive file system specific issues, security reasons and archive package portability reasons, one might want to write down the SIP file under a different name than the one that was provided in the system (often by the user). In that case it is important to generate a mapping file between the original *invenio_sipstore.models.SIPFile.filepath* entries and the archived filenames. It is important to include this mapping in the archive if `SIPSTORE_ARCHIVER_SIPFILE_NAME_FORMATTER` was set to anything other than the default formatter.

See `default_sipfile_name_formatter()` and `secure_sipfile_name_formatter()`.

_write_extra(*fileinfo=None*, *content=None*, *filename=None*)
Write any extra file to the archive.

Requires EITHER 'fileinfo' or ('content' AND 'filename').

Parameters

- **fileinfo** (*dict*) – File information on the custom file.
- **content** (*str*) – Text content of the file
- **filename** (*str*) – Filename of the file.

_write_sipfile(*fileinfo=None*, *sipfile=None*)
Write a SIP file to disk.

***Requires** either *fileinfo* or *sipfile* to be passed.

Parameter *fileinfo* with the file information ('file_uuid' key required) or *sipfile* - the *SIPFile* instance, in which case the relevant file information will be generated on the spot.

Parameters

- **fileinfo** (*dict*) – File information on the SIPFile that is to be written.
- **sipfile** (`invenio_sipstore.models.SIPFile`) – SIP file to be written.

_write_sipmetadata(*fileinfo=None*, *sipmetadata=None*)
Write SIPMetadata file to disk.

get_all_files()
Get the complete list of files in the archive.

Returns the list of all relative final path

get_archive_base_uri()
Get the base URI (absolute path) for the archive location.

To configure the URI, specify the relevant configuration variable *SIPSTORE_ARCHIVER_LOCATION_NAME*, with the name of the `Location` object which will be used as the archive base URI.

Returns the absolute path to the archive location, e.g.:

- `/data/archive/`
- `root://eospublic.cern.ch//eos/archive`

get_archive_subpath()
Generate the relative directory path of the archived SIP.

The behaviour of this method can be changed by changing the *SIPSTORE_ARCHIVER_DIRECTORY_BUILDER* configuration variable.

Generates the relative directory path for the archived SIP, which should be unique for given SIP and is usually built from the SIP information and/or its assigned objects, e.g.:

- `/ab/cd/ab12-abcd-1234-dcba-123412341234` (3-level chunk of SIP UUID identifier).
- `/12345/r/5` (/<PID value>/r/<record revision id>)

The return value of this method is a location that is *relative* to the base archive URI, the full path that is constructed later can look as follows: (based on examples from *BaseArchiver.get_archive_base_uri()*):

- `/data/archive/ab/cd/ab12-abcd-1234-dcba-123412341234`
- `root://eospublic.cern.ch//eos/archive/12345/r/5`

get_fullpath(*filepath*)
Generate the absolute (full path) to the file in the archive system.

Parameters **filepath** (*str*) – path to the file, relative to archive subdirectory e.g. `data/myfile.dat`.

Returns Absolute path, e.g. `root://eospublic.cern.ch//eos/archive/12345/data/myfile.dat`

Return type str

write_all_files(*filesinfo=None*)
Write all files to the archive.

The only parameter of this method *filesinfo* is a list of dict, each containing information on the files that are to be written. There are three types of file-information dict that are recognizable:

- SIPFile-originated, which copy the related FileInstance bytes.
- SIPMetadata-originated, which write down the content of the metadata to the archive.
- Extra files, which writes down short text files, that are usually specific to the archiver format, e.g.: manifest file, README, archive creation timestamp, etc.

By the default when 'filesinfo' is omitted, the base archiver will generate the file info for all attached SIPFiles and SIPMetadata files (but only those which SIPMetadata.type.name was specified in the *SIPSTORE_ARCHIVER_METADATA_TYPES*). Specific archivers are expected to overwrite the *self.get_all_files* method, or craft the *filesinfo* parameter of this method externally.

For more information on the structure of the file-info dict, see JSON Schema: invenio_sipstore.jsonschemas.sipstore.file-v1.0.0.json.

Parameters **filesinfo** – A list of dict, specifying the file information that is to be written down to the archive. If not specified, will execute the *self.get_all_files* to build the files list.

BagIt archiver

Archivers for SIP.

class invenio_sipstore.archivers.bagit_archiver.**BagItArchiver**(*sip*, *data_dir=u'data/files'*, *metadata_dir=u'data/metadata'*, *extra_dir=u''*, *patch_of=None*, *include_all_previous=False*, *tags=None*, *filenames_mapping_file=u'data/filenames.txt'*)

BagIt archiver for SIPs.

Archives the SIP in the BagIt archive format (v0.97). For more information on the BagIt standard visit: https://tools.ietf.org/html/draft-kunze-bagit

Constructor of the BagIt Archiver.

When specifying 'patch_of' parameter the 'include_all_previous' flag determines whether files that are missing in the archived SIP (w.r.t. the SIP specified in 'patch_of') should be treated as explicitly deleted (include_all_previous=False) or if they should still be included in the manifest.

Example:

include_all_previous = True

SIP_1: SIPFiles: a.txt, b.txt BagIt Manifest: a.txt, b.txt

SIP_2 (Bagged with patch_of=SIP_1): SIPFiles: b.txt, c.txt BagIt Manifest: a.txt, b.txt, c.txt fetch.txt: a.txt, b.txt

include_all_previous = False

SIP_1: SIPFiles: a.txt, b.txt BagIt Manifest: a.txt, b.txt

SIP_2 (Bagged with patch_of=SIP_1): SIPFIles: b.txt, c.txt BagIt Manifest: b.txt, c.txt fetch.txt: b.txt

Parameters

- **sip** (*invenio_sipstore.api.SIP* or *invenio_sipstore.models.SIP*) – API instance of the SIP that is to be archived.
- **data_dir** – directory where the SIPFiles will be written.
- **metadata_dir** – directory where the SIPMetadata will be written.
- **extra_dir** – directory where all extra files will be written, including the BagIt-specific files.
- **patch_of** (*invenio_sipstore.api.SIP* or *invenio_sipstore.models.SIP*) – Write a 'lightweight' bag, which will archive only the new SIPFiles, and refer to the repeated ones in "fetch.txt" file. The provided argument is a SIP API, which will be taken as a base for determining the "diff" between two bags.
- **tags** – a list of 2-tuple containing the tags of the bagit, which will be written to the 'bag-info.txt' file.
- **filenames_mapping_file** – filepath of the file in the archive which contains all of SIPFile mappings. If this parameter is boolean-resolvable as False, the file will not be created.

_generate_bagging_date()

Generate the bagging date timestamp.

classmethod _get_bagit_metadata_type()
Return the SIPMetadataType for the BagIt metadata files.

static _get_checksum(*checksum*, *expected=u'md5'*)
Return the checksum if the type is the expected.

static _is_fetched(*file_info*)
Determine if file info specifies a file that is fetched.

archiver_version = u'SIPBagIt-v1.0.0'
Specification of the SIP bag structure.

This name will be formatted as `External-Identifier` tag:

```
External-Identifier: <SIP.id>/<archiver_version>
```

bagit_metadata_type_name = u'bagit'
Name of the SIPMetadataType for internal use of BagItArchiver.

get_all_files()
Create the BagIt metadata object.

get_baginfo_file(*filesinfo*)
Create the bag-info.txt file from the tags.

get_bagit_file()
Create the bagit.txt file which specifies the version and encoding.

Returns File information dictionary

Return type dict

classmethod get_bagit_metadata(*sip*, *as_dict=False*)
Fetch the BagIt metadata information (SIPMetadata).

Parameters **sip** – SIP for which to fetch the metadata.

Returns Return the BagIt metadata information (SIPMetadata) instace or None if the object does not exist.

get_fetch_file(*filesinfo*)
Generate the contents of the fetch.txt file.

get_manifest_file(*filesinfo*)
Create the manifest file specifying the checksum of the files.

Returns the name of the file and its content

Return type tuple

get_tagmanifest_file(*filesinfo*)
Create the tagmanifest file using the files info.

Returns the name of the file and its content

Return type tuple

save_bagit_metadata(*filesinfo=None*, *overwrite=False*)
Generate and save the BagIt metadata information as SIPMetadata.

write_all_files()
Write all of the archived files to the archive file system.

Admin

CHAPTER 3

Additional Notes

Notes on how to contribute, legal information and changes are here for the interested.

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

Types of Contributions

Report Bugs

Report bugs at https://github.com/inveniosoftware/invenio-sipstore/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Invenio-SIPStore could always use more documentation, whether as part of the official Invenio-SIPStore docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/inveniosoftware/invenio-sipstore/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *invenio* for local development.

1. Fork the *invenio* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/invenio-sipstore.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv invenio-sipstore
$ cd invenio-sipstore/
$ pip install -e .[all]
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass tests:

```
$ ./run-tests.sh
```

   The tests will provide you with test coverage and also check PEP8 (code style), PEP257 (documentation), flake8 as well as build the Sphinx documentation and run doctests.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -s -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests and must not decrease test coverage.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 2.7, 3.3, 3.4 and 3.5. Check https://travis-ci.com/inveniosoftware/invenio-sipstore/pull_requests and make sure that the tests pass for all supported Python versions.

Changes

Version v1.0.0a7 (released 2017-08-18)

- Initial public release.

License

Authors

Submission Information Package store for Invenio.

- CERN <info@inveniosoftware.org>
- Rémi Ducceschi <remi.ducceschi@gmail.com>

Python Module Index

i

Index

Symbols

A

B

C

D

T

U

V

W