Sphinx Objects.inv Converter Documentation

Release 1.0

Brian Skinn

Jan 29, 2018

Contents

When documentation is built using, e.g., Sphinx's `StandaloneHTMLBuilder`, an inventory of the named objects in the documentation set is dumped to a file called `objects.inv` in the html build directory. This file is read by `intersphinx` when generating links in other documentation.

Since version 1.0 of Sphinx (~July 2010), the data in these `objects.inv` inventories is compressed by `zlib` (presumably to reduce storage requirements and improve download speeds; "version 2"), whereas prior to that date the data was left uncompressed ("version 1"). This compression renders the files non-human-readable. **It is the purpose of this package to enable quick and simple encoding/decoding of these files.**

In particular, `sphobjinv` was developed to satisfy two primary use cases:

1. Searching and inspection of `objects.inv` contents in order to identify how to properly insert `intersphinx` references.
2. Assembly of new `objects.inv` files in order to allow `intersphinx` cross-referencing of other documentation sets that were not created by Sphinx.

Sphinx Objects.inv Encoder/Decoder is available on PyPI under `sphobjinv`:

```
pip install sphobjinv
```

The package is configured for use both as a *command-line script* and as a *Python package*.

The project source repository is on GitHub: bskinn/sphobjinv.

CHAPTER 1

Command-Line Usage

On most systems, `sphobjinv` should automatically install with a command line / shell script inserted into {python}/Scripts and thus be executable from anywhere.

Argument handling for both encode and decode operations has been written in an attempt to make using the script as user-friendly as possible. The script has the following usage syntax (Windows version shown):

```
> sphobjinv --help

usage: sphobjinv-script.py [-h] {encode,decode} [infile] [outfile]

Decode/encode intersphinx 'objects.inv' files.

positional arguments:
  {encode,decode}  Conversion mode
  infile           Path to file to be decoded (encoded). Defaults to
                   './objects.inv(.txt)'. Bare paths are accepted, in which
                   case the above default input file names are used in the
                   indicated path. '-' is a synonym for these defaults.
  outfile          Path to decoded (encoded) output file. Defaults to same
                   directory and main file name as input file but with
                   extension .txt (.inv). Bare paths are accepted here as
                   well, using the default output file names.

optional arguments:
  -h, --help       show this help message and exit
```

In particular, note the default input file names (`objects.inv` and `objects.txt`), and the ability to select these default file names in a given directory by simply passing the path to that directory.

Examples

All of the below are decode operations executed on Windows; encode operations behave essentially the same, except for swapping the default input/output file extensions. Adapt the path syntax, etc. as appropriate for the relevant operating system.

To decode a file with the Sphinx-default name of `objects.inv` residing in the current directory, to the default output file of `objects.txt`:

```
> sphobjinv decode

Conversion completed.
'.\objects.inv' decoded to '.\objects.txt'.
```

To decode the same `objects.inv` file to `objects_custom.`:

```
> sphobjinv decode - objects_custom.

Conversion completed.
'.\objects.inv' decoded to '.\objects_custom.'.
```

To decode `objects_python.inv` residing in the root directory to `objects_python.txt` in the directory `\temp`:

```
> sphobjinv decode \objects_python.inv \temp

Conversion completed.
'\objects_python.inv' decoded to '\temp\objects_python.txt'.
```

To decode the same `objects_python.inv` to `new_objects.txt` in the directory `\git`:

```
> sphobjinv decode \objects_python.inv \git\new_objects.txt

Conversion completed.
'\objects_python.inv' decoded to '\git\new_objects.txt'.
```

CHAPTER 2

API

The primary `sphobjinv` API consists of two pairs of functions:

- *`readfile()`* / *`writefile()`* – Read/write files from/to disk as `bytes`, for proper behavior of `zlib` (de)compression.
- *`encode()`* / *`decode()`* – Encode/decode the object data read from disk.

Also exposed are two `re.compile()` patterns, potentially useful in parsing **decoded data only**:

- *`p_comments`* – Retrieves the #-prefixed comment lines
- *`p_data`* – Retrieves all lines not prefixed by #

The normal workflow would be:

1. Import the module; e.g.:

```
>>> import sphobjinv as soi
```

2. Read the desired file data (compressed or uncompressed) with *`readfile()`*:

```
>>> fd = soi.readfile('/path/to/file')
```

3. Decode [or encode] the file data with *`decode()`* [or *`encode()`*]:

```
>>> data = soi.decode(fd)
```

4. Write the desired file with *`writefile()`*, or otherwise use the resulting `bytes` data:

```
>>> len(soi.p_data.findall(data))    # e.g., retrieve the number of object entries
6319

>>> soi.writefile('/path/to/new/file', data)
```

Members

decode (*bstr*)
Decode a version 2 `intersphinx` `objects.inv` bytestring.

The #-prefixed comment lines are left unchanged, whereas the `zlib`-compressed data lines are uncompressed to plaintext.

Parameters **bstr** – `bytes` – Binary string containing an encoded `objects.inv` file.

Returns *out_b* – `bytes` – Decoded binary string containing the plaintext `objects.inv` content.

encode(*bstr*)

Encode a version 2 `intersphinx` `objects.inv` bytestring.

The #-prefixed comment lines are left unchanged, whereas the plaintext data lines are compressed with `zlib`.

Parameters **bstr** – `bytes` – Binary string containing the decoded contents of an `objects.inv` file.

Returns *out_b* – `bytes` – Binary string containing the encoded `objects.inv` content.

p_comments = re.compile(b'^#.*$', re.MULTILINE)

Bytestring regex pattern for comment lines in decoded `objects.inv` files

p_data = re.compile(b'^[^#].*$', re.MULTILINE)

Bytestring regex pattern for data lines in decoded `objects.inv` files

readfile(*path*, *cmdline=False*)

Read file contents and return as binary string.

Parameters

- **path** – `str` – Path to file to be opened.
- **cmdline** – `bool` – If `False`, exceptions are raised as normal. If `True`, on raise of any subclass of `Exception`, the function returns `None`.

Returns *b* – `bytes` – Binary contents of the indicated file.

writefile(*path*, *contents*, *cmdline=False*)

Write indicated file contents (with clobber).

Parameters

- **path** – `str` – Path to file to be written.
- **contents** – `bytes` – Binary string of data to be written to file.
- **cmdline** – `bool` – If `False`, exceptions are raised as normal. If `True`, on raise of any subclass of `Exception`, the function returns `None`.

Returns *p* – `str` – If write is successful, echo of the *path* input `str` is returned. If any `Exception` is raised and *cmdline* is `True`, `None` is returned.

CHAPTER 3

Sphinx Objects.inv Syntax

Plaintext Sphinx `objects.inv` files follow a syntax that, to the best of this author's ability to determine, is completely undocumented. The below syntax is believed to be accurate as of May 2016 (`objects.inv` "version 2" files, Sphinx v1.4.1). Based upon a quick `git diff` of the Sphinx repository, it is thought to be accurate for all Sphinx>=1.0b1.

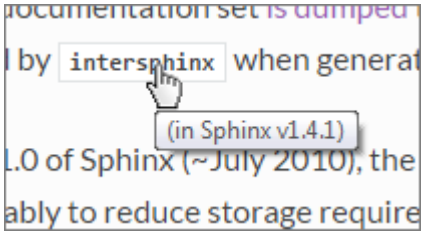
The first line must be exactly:

```
# Sphinx inventory version 2
```

The second and third lines must obey the template:

```
# Project: <project name>
# Version: <full version number>
```

The above project name and version are used to populate mouseovers for the `intersphinx` cross-references:



The fourth line must contain the string 'zlib' somewhere in it, but for the purposes of consistency it should be exactly:

```
# The remainder of this file is compressed using zlib.
```

All remaining lines of the file are the objects data, each laid out in the following syntax:

```
{name} {domain}:{role} {priority} {uri} {dispname}
```

`{name}` The object name used when cross-referencing the object (falls between the backticks)

{domain} The Sphinx domain used when cross-referencing the object (falls between the first and second colons; omitted if using the default domain)

{role} The Sphinx role used when cross-referencing the object (falls between the second and third colons; or, between the first and second colons if using the default domain)

{priority} Flag for placement in search results. Most will be 1 (standard priority) or -1 (omit from results)

{uri} Relative URI for the location to which cross-references will point. The base URI is taken from the relevant element of the `intersphinx_mapping` configuration parameter of `conf.py`.

{dispname} Default cross-reference text to be displayed in compiled documentation.

For illustration, the following is the entry for the `join()` method of the `str` class in the Python 3.5 `objects.inv`, broken out field-by-field:

```
str.join py:method 1 library/stdtypes.html#$ -

{name}      = str.join
{domain}    = py
{role}      = method
{priority}  = 1
{uri}       = library/stdtypes.html#$
{dispname}  = -
```

The above illustrates two shorthand notations that were introduced to shrink the size of the inventory file:

1. If `{uri}` has an anchor (technically a "fragment identifier," the portion following the `#` symbol) and the tail of the anchor is identical to `{name}`, that tail is replaced with `$`.
2. If `{dispname}` is identical to `{name}`, it is stored as `-`.

Thus, a standard `intersphinx` reference to this method would take the form (the leading `:py` could be omitted if `py` is the default domain):

```
:py:meth:`str.join`
```

The cross-reference would show as `str.join()` and link to the relative URI:

```
library/stdtypes.html#str.join
```

Other intersphinx Syntax Examples

To show as only `join()`:

```
:py:meth:`~str.join`
```

To suppress the hyperlink as in `str.join()`:

```
:py:meth:`!str.join`
```

To change the cross-reference text and omit the trailing parentheses as in `This is join!`:

```
:py:obj:`This is join! <str.join>`
```

CHAPTER 4

Creating and Using a Custom objects.inv

1. Identify the head of the URI to the documentation.
2. Create the custom `objects.inv` file.

- Create the header per the required *syntax*, entering the project name and version as appropriate.
- Create lines of object data, again per the required *syntax*.
  - Be sure only to use the relative portion of the URI for the `{uri}` field.
  - Choose an appropriate domain/role for each object. If necessary to avoid conflicts, a custom domain can be created; otherwise, one of the built-in domains may suffice.

3. Encode the file with `sphobjinv`.
4. Transfer the encoded file to its distribution location.

- If only local access is needed, it can be kept local.
- If external access needed, upload to a suitable host.

5. Add an element to the `intersphinx_mapping` parameter in `conf.py`.

- The element key is arbitrary.
- The first element of the value tuple is the head URI for the repository.
- The second element of the value tuple is the address of the distribution location of the encoded file.

Python Module Index

S

Index