# ssf Documentation

*Release 0.2.2*

**Joe Cool**

**Dec 10, 2020**

# CONTENTS:

CHAPTER

ONE

# SSF

Spreadsheet Number Format processor - a Python port of SheetJS/ssf.js

- Free software: Apache Software License 2.0
- Documentation: https://ssf.readthedocs.io.

## 1.1 Features

ssf (Spreadsheet Format) is a pure python library to format data using ECMA-376 spreadsheet format codes (used in popular spreadsheet software packages). It is derived from the JavaScript version available at https://github.com/SheetJS/ssf. All listed issues in that package, up to #80, have been fixed in this version and support for colors, widths, and localization including alternative calendars have also been implemented.

## 1.2 Python Version and Required Libraries

A modern version of Python is required to use *ssf*: version 3.6 or better. Also, these libraries are required by *ssf*: *Babel*, *python-dateutil*, *pytz*, *pyYAML*, *six*, *ummalqura*, *convertdate*.

## 1.3 Example

- Basic Demo

## 1.4 Credits

This package is a Python port of the similarly named JavaScript library (https://github.com/SheetJS/ssf).

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

# INSTALLATION

## 2.1 Stable release

To install ssf, run this command in your terminal:

```
$ pip install ssf
```

This is the preferred method to install ssf, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

## 2.2 From sources

The sources for ssf can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/snoopyjc/ssf
```

Or download the tarball:

```
$ curl -OJL https://github.com/snoopyjc/ssf/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

# USAGE

## 3.1 Simple Usage

To use ssf in a project:

```
>>> from ssf import SSF

>>> ssf = SSF()
>>> ssf.format('#,##0', 1000)
'1,000'
>>> ssf.format('Currency', 1000.98)
'$1,000.98'
>>> ssf.format('Currency', 1000.98, locale='de-DE')
'1.000,98 €'
>>> ssf.format('General', 100000, width=5)
'1E+05'
>>> ssf.format('?/?', 3.14159)
'22/7'
>>> ssf.format('Long Date', '1/1/2001', locale='fr-FR')
'lundi 01 janvier 2001'
>>> ssf.format('[DBNum1][$-804]General', 12.3456789)
'.'

>>> ssfc = SSF(color_pre='<span style="color:#{rgb}">', color_post='</span>')
>>> ssfc.format('0;[Red]0;0;@', -2)
'<span style="color:#FF0000">2</span>'
```

See the API reference for the options you can pass to *SSF()*. Most options can be also passed to the *ssf.format()* method, enabling you to create one instance of the *SSF* object.

## 3.2 Manipulating the Internal Format Table

Binary spreadsheet formats store cell formats in a table and references them by index. This library uses a table implemented as a dict per object instance. You can use *ssf.load(fmt)* to load additional entries into unused slots in the table - it returns the index it chooses. You can also specify the index by using *ssf.load(fmt, ndx)*. For compatibility with the XLS and XLSB file formats, custom indices should be in the valid ranges *5-8*, *23-26*, *41-44*, *63-66*, *164-382* (see *[MS-XLSB] 2.4.655 BrtFmt*)

*ssf.get_table()* gets the internal table as a dict (number to format string mapping).

*ssf.load_table(table)* sets the internal table from a dict mapping ints to format strings.

## 3.3 Other Utilities

Static method *SSF.is_date(fmt)* returns *True* if *fmt* encodes a date format.

*ssf.get_format(type, ...)* returns a format appropriate for the type and locale. The *type* should be (*General*, *Number*, *Currency*, *Accounting*, *Date*, *Short Date*, *Long Date*, *Time*, *Percentage*, *Fraction*, *Scientific*, or *Text*). See the API reference for more information.

## 3.4 References

- *MS-XLSB*: Excel (.xlsb) Binary File Format
- *ECMA-376*: Number Format Specification

CHAPTER

FOUR

# SSF

## 4.1 ssf package

### 4.1.1 Submodules

### 4.1.2 ssf.ssf module

**class** ssf.ssf.**SSF**(*tzinfo=None*, *date1904=False*, *dateNF=None*, *table=None*, *color_pre=None*, *color_post=None*, *locale_support=True*, *locale=None*, *default_width=None*, *decimal_separator=None*, *thousands_separator=None*, *errors='warn'*)

Bases: object

Spreadsheet Formatter (number format). Formats values according to spreadsheet-style format codes. If date1904 is True, then the base date is January 1, 1904, which was used on some spreadsheet programs for Mac. The default (False), means that the base date is December 31, 1899 (which spreadsheet programs call the 1900 date system). The dateNF if not None, replaces the default Short Date format of *m/dd/yyyy*. The table if not None, replaces the entire translation from ints to formats table.

The color_pre and color_post specify formats for values that are provided before and after the results that have a [ColorN] or color name e.g. [Red] specifier in the formats. Any {} in the specified format are replaced by the specified color (in Title Case). Any {rgb} in the formats are replaced with the hex color number (without a #).

If locale_support is True, then handle the international decimal point and thousands separator changes, and the language-based month names. To use this, you can pass the locale here, or when you call ssf.format(). If locale is None, then the default local locale is used. The default_width, if not None, gives the width to use on every ssf.format() call, if not otherwise specified. The decimal_separator and thousands_separator are used to override the defaults as specified by the locale. The errors parameter specifies what to do on locale (and other) errors. The default is to warn using the warnings module, then ignore the error. The other choices are 'ignore', which completely ignores the error, 'pounds', which fills the result with '#' characters, and 'raise', which will raise a ValueError exception.

**SSF_js_version = '0.11.2'**

**autocorrect_format**(*fmt*)

Run some automatic corrections on the given format, and return the corrected format

**static fmt_is_date**(*fmt*)

Returns True iff this fmt a date format

**format**(*fmt*, *v*, *width=None*, *align=None*, *locale=None*, *decimal_separator=None*, *thousands_separator=None*)

Format a value v according to the spreadsheet format in fmt with field width with alignment align. If width is not specified, then the *default_width* from the *ssf* object is used. The align can be specified as

'left', 'right', 'center' or None. If `align` is None, the alignment is defaulted by the type of the value and the format. Text is left aligned, numbers and dates are right aligned, and bool's are centered. If the format is a text format (`@`), then the default is left aligned for all types of values. If `locale` is not None and the `ssf` object supports locale, then this locale is used as the default locale if none is otherwise specified in the format. The `decimal_separator` and `thousands_separator` come from the specified locale, or the locale of the *ssf* object if None. If specified, they override the default. If they are specified as *inherit*, then the corresponding values of the *ssf* object are used even if a `locale` is specified here. Note that any locale specified in the format itself does not change these separator values, to be consistent with spreadsheet implementations.

**getTime**(*dt*)
JavaScript style: Milliseconds since an epoch

**getTimezoneOffset**(*dt*)
JavaScript style: Minutes from UTC

**get_day_names**()
Returns a 7-tuple containing 2-tuples of the abbreviation and full-day name, with Monday first

**get_format**(*type='General'*, *places=None*, *use_thousands_separator=None*, *negative_numbers=None*, *fraction_denominator=- 1*, *positive_sign_exponent=True*, *locale=None*)
Get an appropriate format for the `locale` either specified here or the locale of the `ssf` object. The `type` is one of General, Number, Currency, Accounting, Date, Short Date, Long Date, Time, Percentage, Fraction, Scientific, or Text (in any case). If `places` is not None and this is a number format, then this specifies the number of decimal places, else a default is used. Also for number formats, `use_thousands_separator` determines if the locale-specified thousands separator is used. For currency and accounting formats, `use_thousands_separator` defaults to *True*. In addition, `negative_numbers` specifies how to format negative numbers. It can be None, which uses a default format depending on the type (normally '-'), or -, which always uses a minus sign, *Red*, which formats in red without a minus sign, *parens* which formats in parenthesis, or *Redparens*, which does both red and parenthesis. You can also specify *()* as a synonym for *parens*.

For currencies, these additional `negative_numbers` formats are supported:

- <<− The sign should precede the value and currency symbol (− does this too)
- >>− The sign should follow the value and currency symbol
- <− The sign should immediately precede the value
- >− The sign should immediately follow the value

For Fraction formats, the `fraction_denominator` specifies the denominator to be used for the fraction. If it is negative, then it instead specifies how many digits to use in the numerator. If it is zero, then a ValueError is raised.

For Scientific formats, `positive_sign_exponent` determines if a positive sign is displayed for positive exponents. The default is True.

**get_month_names**()
Returns a 13-tuple containing 3-tuples of the single-letter abbreviation, the abbreviation, and the full-month name. The entry at index 0 is None.

**get_table**()
Returns the mapping table (a dict) from ints to format strings

**static is_date**(*fmt*)
Returns True iff this `fmt` a date format

**static is_text_fmt**(*fmt*)
Is this a text format?

**load**(*fmt*, *idx=None*)
Loads a single format entry specified by `fmt` into the mapping table. If `idx` is specified, then that is used as the table index, else the first free entry is used. The index used is returned.

**load_entry**(*fmt*, *idx=None*)
Loads a single format entry specified by `fmt` into the mapping table. If `idx` is specified, then that is used as the table index, else the first free entry is used. The index used is returned.

**load_table**(*tbl*)
Given a dict of table indexes and values in `tbl`, load it for use by the formatter

**locale_prefix**(*locale*)
Returns the appropriate [$-zzzz] locale prefix for the given `locale`.

**static round**(*number*, *places=0*)
JavaScript style: Round 0.5 always up - not to even like python 3

**static round_to_precision**(*v*, *p*)
Like toPrecision, except returns a float

**set_day_names**(*days*)
Given an iterable of length 7 as `days`, each of which containing a tuple of the abbreviation and full-day name, with Monday first, set this as the values to be returned for ddd and dddd formats, respectfully.

**set_month_names**(*months*)
Given an iterable of length 13 as `months`, each of which containing a 3-tuple of the single-letter abbreviation, the abbreviation, and the full-month name, set this as the values to be returned for mmmmm, mmm, and mmmm formats, respectfully. The first element is not used, so that the first month has index = 1.

**static toPrecision**(*v*, *p*)
Emulates JavaScript's flt.toPrecision(p)

**static to_str**(*v*)
Emulate the ""+val in JavaScript. If val is float but is an integer value, then the decimal is removed.

**class** ssf.ssf.**SSF_CALENDAR**(*calendar=0*)
Bases: `object`

Handle alternative calendars for ssf. This shouldn't be used directly.

**CHINESE_LUNAR = 19**

**GREGORIAN_ARABIC = 10**

**GREGORIAN_LOCAL = 1**

**GREGORIAN_MIDDLE_EASTERN_FRENCH = 9**

**GREGORIAN_TRANSLITERATED_ENGLISH = 11**

**GREGORIAN_TRANSLITERATED_FRENCH = 12**

**GREGORIAN_US = 2**

**HIJRI = 6**

**JAPANESE = 3**

**JEWISH = 8**

**KOREAN = 5**

**LUNAR_x0E = 14**

**LUNAR_x11 = 17**

**LUNAR_x12 = 18**

**SYSTEM_DEFAULT = 0**

**TAIWAN = 4**

**THAI_BUDDHIST = 7**

**UM_AL_QURA = 23**

**day_month_map = {}**

**day_names** (*locale_name*)
Return the day names for this calendar in the given `locale_name`. The result is an array of day names, starting with Sunday in index 0, or None if we have no day names for this locale

**ecclesiastical_leap_to_civil = {1: 8, 2: 9, 3: 10, 4: 11, 5: 12, 6: 13, 7: 1, 8**

**ecclesiastical_to_civil = {1: 7, 2: 8, 3: 9, 4: 10, 5: 11, 6: 12, 7: 1, 8: 2,**

**era_list = None**

**fixup_special** (*ymd*)

**property has_leap_month**
Could this calendar have a leap month?

**lunar_bin = None**

**month_names** (*locale_name*, *isleap=False*)
Return the month names for this calendar in the given `locale_name`. The result is an array of month names, starting with January in index 0, or None if we have no month names for this locale. If isleap is True and this calendar has a leap month, then the result contains 13 entries, else it has the normal 12.

**to_chinese_lunar** (*ymd*)

**to_default** (*ymd*)

**to_hijri** (*ymd*)

**to_japanese** (*ymd*)

**to_jewish** (*ymd*)

**to_korean** (*ymd*)

**to_local** (*ymd*)
Convert a tuple containing (year, month, day) to a SimpleNamespace containing (year, month, day, isleap, era)

**to_lunar_x0e** (*ymd*)

**to_lunar_x11** (*ymd*)

**to_lunar_x12** (*ymd*)

**to_taiwan** (*ymd*)

**to_thai_buddhist** (*ymd*)

**to_um_al_qura** (*ymd*)

**um_bin = None**

**x0D = 13**

**x0F = 15**

**x10 = 16**

**x14 = 20**

**x15 = 21**

**x16 = 22**

**x18 = 24**

**x19 = 25**

**x1A = 26**

**x1B = 27**

**x1C = 28**

**x1D = 29**

**x1E = 30**

**x1F = 31**

**class** ssf.ssf.**SSF_LOCALE**(*locale=None*, *locale_support=True*, *locale_currency=True*, *decimal_separator=None*, *thousands_separator=None*, *calendar_code=None*)

Bases: object

Handle locale support for SSF. This shouldn't be used directly.

**GANNEN = ''**

**MAX_AMPM = 6**

**am_pm_map = None**

**commaify**(*s*)

**currency_map = None**

**dbnum_map = None**

**era_map = None**

**lc_all_map = None**

**lcid_map = None**

**lcid_max = 0**

**lcid_reverse_map = None**

**normalize_locale**(*locale*)

Normalize locale based on examples in the lcid/locale map

**numbers_map = None**

**table_map = None**

### 4.1.3 Module contents

Top-level package for ssf.

CHAPTER

FIVE

# CONTRIBUTING

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

You can contribute in many ways:

## 5.1 Types of Contributions

### 5.1.1 Report Bugs

Report bugs at https://github.com/snoopyjc/ssf/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.

### 5.1.2 Fix Bugs

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

### 5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

### 5.1.4 Write Documentation

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

### 5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/snoopyjc/ssf/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 :)

## 5.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/ssf.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 ssf
$ cd ssf/
$ python setup.py develop
```

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 the tests, including testing other Python versions with tox:

```
$ python setup.py test or pytest
$ tox
```

   To get tox, just pip install them into your virtualenv.

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

```
$ git add .
$ git commit -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.

## 5.3 Pull Request Guidelines

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

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/snoopyjc/ssf/pull_requests and make sure that the tests pass for all supported Python versions.

## 5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_ssf
```

## 5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

SIX

# CREDITS

## 6.1 Development Lead

- Joe Cool <snoopyjc@gmail.com>

Based on the work done by SheetJSDev (https://sheetjs.com/) community edition SpreadSheet Format (https://github.com/SheetJS/ssf).

## 6.2 Contributors

None yet. Why not be the first?

CHAPTER

SEVEN

# HISTORY

## 7.1 0.2.2 (2020-10-29)

- Fix issues #13, #14, #15

## 7.2 0.2.1 (2020-10-26)

- Fix issues #5, #10, #11, #12

## 7.3 0.2.0 (2020-10-25)

- Fix issue #6: Add support for other calendars
- Fix issues #7-9

## 7.4 0.1.5 (2020-10-14)

- Fix issues #1, #3, and JS#79, JS#80.
- Add demo doc and encode/decode the results so unicode comes thru

## 7.5 0.1.2 (2020-10-08)

- Add data files to distro

## 7.6 0.1.0 (2020-10-05)

- First release on PyPI.

CHAPTER

EIGHT

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## S

# INDEX

## N



## R



## S



## T



## U



## X